ESCOLA NAVAL

talante ♔ bien faire

João Francisco Barracosa Santos

eVentos – Desenvolvimentos no veleiro autónomo Barlavento

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Marinha

Alfeite

2019

ESCOLA NAVAL

talanteviefaire

João Francisco Barracosa Santos

eVentos – Desenvolvimentos no veleiro autónomo Barlavento

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Marinha

Orientação de: Professor Doutor Victor Lobo

Coorientação de: 2TEN EN-MEC Castro Fernandes

O Aluno Mestrando

ASPOF M Barracosa Santos

O Orientador

Professor Doutor Victor Lobo

Alfeite

2019

Agradecimentos

Esta dissertação de mestrado só foi possível de ser realizada, com o auxílio de inúmeras pessoas, às quais não posso deixar de reconhecer o apoio nesta navegação.

Em primeiro lugar gostaria de agradecer todo o apoio incondicional da minha família.

Agradeço de forma muito especial aos meus orientadores, Professor Doutor Victor Lobo e 2TEN EN-MEC Castro Fernandes, por toda a preocupação, amizade e motivação, sem eles não seria possível.

À Célula de Experimentação Operacional de Veículos, um também especial agradecimento por me receber de braços abertos.

Não posso deixar de gratular o todo o apoio oficinal da ODAM - Oficina D'Almada, imprescindível neste trabalho, na pessoa do João Catarino.

Quero também agradecer à guarnição e em particular à câmara de oficiais do NRP *Almirante Gago Coutinho*, navio onde estagiei e que me fez crescer.

Aos meus camaradas do Curso Jorge Álvares e em particular ao camarada ASPOF EN-MEC Jorge da Cruz pelo apoio e troca de ideias no decorrer do projeto.

Resumo

De entre os veículos de superfície não tripulados existentes, os veleiros autónomos são uma tecnologia promissora para missões persistentes e de longa duração. Esta dissertação de mestrado incluiu o melhorar e construir constituintes estruturais e sensores do veleiro autónomo "Barlavento" da Escola Naval, com recurso à modelação e impressão 3D e maquinação por comando numérico computorizado. Paralelamente, foi implementado e validado um algoritmo de comando e controlo, bem como incrementada a fiabilidade do mesmo durante a sua navegação autónoma. Foram integrados novos sensores e *hardware*, e estabelecidas comunicações externas no protocolo NMEA[1] 0183, por forma a garantir a compatibilidade com *software opensource.* Por último são apresentados testes e resultados das provas de mar.

Palavras-chave: Veleiro autónomo, Impressão 3D, comando numérico computorizado, comando e controlo, NMEA

[1] *National Marine Electronics Association*

Abstract

Among existing unmanned surface vehicles, autonomous sailboats are a promising technology for persistent and long-term missions. This work included the improvement and construction of structural components and sensors of the Naval Academy "Barlavento" autonomous sailboat, using 3D modeling and printing besides computer numerical command machining. At the same time, a command and control algorithm was implemented and validated, as well as its reliability during its autonomous navigation. New sensors and hardware have been integrated and external communications have been established using the NMEA 0183 protocol to ensure compatibility with opensource software. Lastly, tests are presented and results from sea trials.

Keyword: Autonomous sailboat, 3D printing, Computer numeric command, Command and control, NMEA

Índice

Índice de figuras

Índice de tabelas

Índice de equações

Introdução

Esta dissertação apresenta o trabalho feito para melhorar um veleiro autónomo que começou a ser desenvolvido na Escola Naval em 2015 (Fernandes, 2016), e que já foi usado em competições de vela robótica. O trabalho consistiu em melhorar a estrutura do veleiro, com ênfase especial nas suas velas rígidas, bem como reformular o sistema de sensores, atuadores, e algoritmo de controlo do veleiro. Entre as melhorias está a utilização do protocolo NMEA 0183 em todas as comunicações externas, a utilização extensiva de impressão 3D e máquinas de controlo numérico para construir peças, a integração de software de acesso aberto para visualização, recolha, e análise de dados, e estudo e construção das novas velas.

Navegação é "o processo de planear, gravar e controlar o movimento de um veículo de um sítio para outro" (Bowditch, 2002, p. 799), e neste caso essa função é executada por um microcomputador/microcontrolador, a partir de objetivos ou pré-definidos no código ou enviados de uma estão de controlo em terra (GCS).

A vela é influenciada por uma interação complexa de forças geradas pelo vento e pela água com ação direta na embarcação. A vela robótica é então o conjunto de ações autónomas, suportadas por sensores, com o objetivo de levar uma embarcação à vela de um sítio para outro.

As características da vela robótica incluem (Stelzer & Jafarmadar, 2011a):

- O vento ser a única fonte de propulsão;
- O sistema de controlo é feito totalmente a bordo e não radio controlado;
- Serem veículos energeticamente autossuficientes (pelo menos para propulsão).

Neste capítulo pretende-se fazer uma introdução à vela robótica, fazendo um resumo do que já foi desenvolvido nesta área. Pretende-se ainda apresentar aplicações deste tipo de veículos e as motivações que levaram ao desenvolvimento deste trabalho.

História da vela robótica

Grande parte do desenvolvimento tecnológico em *veículos de superfície não tripulados* ou em inglês unmanned *surface vehicles* (USV), está confinada a veículos movidos a motores elétricos ou de combustão interna, capazes de velocidades elevadas ou médias e pouca autonomia. Estes veículos são bastante limitados, na medida em que não conseguem efetuar missões de longo curso e de grande *endurance*, pois estão limitados energeticamente. É aí que surge a grande vantagem de veleiros autónomos, que apenas necessitam de energia para esforços energéticos mínimos como manter o computador de bordo e sensores ligados ou efetuar ajustes ligeiros nas velas ou no leme. É ainda possível fazer um controlo energético, cingindo-o apenas para tarefas indispensáveis, em situações críticas que requerem a recolha do veículo.

Uma das grandes aplicações deste tipo de veículos, (USV) é a monotorização oceânica que está limitada no espaço e no tempo. Plataformas fixas como boias, têm grande vantagem por recolherem dados ao longo de grandes séries temporais, mas pecam na vertente da cobertura em área, na medida em que são fixas. Navios oceanográficos, por outro lado, conseguem cobrir grandes áreas, mas estão limitados no tempo de serviço pois têm custos elevados de missão.

Saildrone

Um excelente exemplo de USV é o *Saildrone*, desenvolvido em parceria entre a *Saildrone Inc*. e a NOAA (*National Oceanic and Atmospheric Administration*) (Meinig, Lawrence-Slavas, Jenkins, & Tabisola, 2015). Este veículo não-tripulado e autónomo é um veleiro capaz de desempenhar missões de até doze meses. É dotado de uma vela de 4 m de altura que mais se assemelha a uma asa de avião que será discutido no capítulo . Através de uma estação em terra são carregados no sistema *waypoints* de passagem e corredores que delimitam os bordos a que o veleiro está restrito, permitindo a navegação dentro deles, para qualquer direção e intensidade de vento e corrente (desde que tenha vento suficiente). Para comunicar com terra, possui modems *Iridium* SBD (*Short Burst Data*) que facilitam a troca de dados críticos para a missão, comunicando-os a cada dez minutos; e ainda um outro modem independente, o RUDICS (*router-based unrestricted*

digital) que fornece as séries temporais comprimidas dos dados recolhidos a cada seis horas. Esta plataforma oferece a vantagem de ser facilmente reconfigurável, podendo integrar novos sensores por meio de portas série. Ainda assim, carrega uma panóplia de sensores como se pode ver na Figura 1.

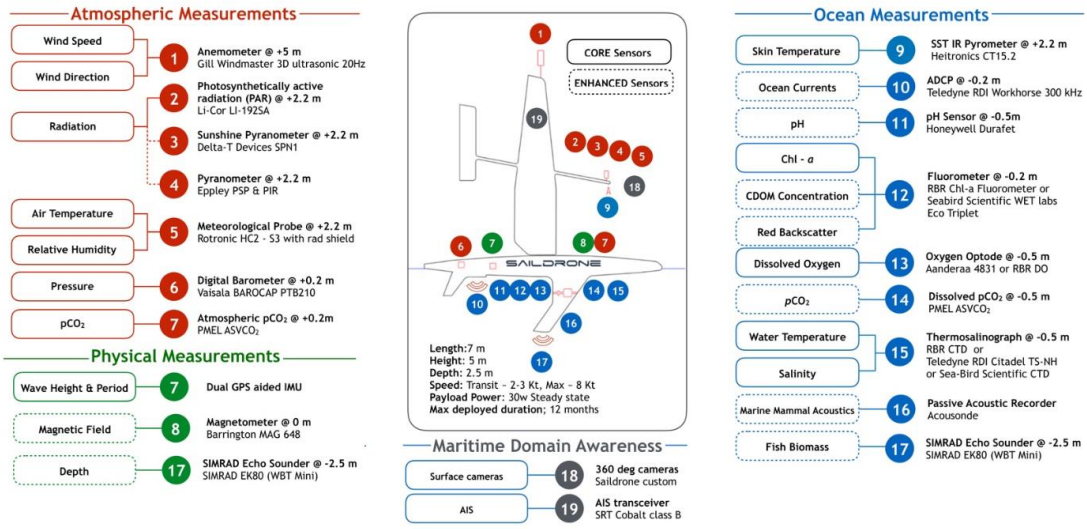

Figura 1 Sensores e capacidades do Saildrone (imagem adaptada de https://www.saildrone.com/#Technology)

Vela robótica em Portugal

FASt

A Faculdade de Engenharia da Universidade do Porto (FEUP), desenvolveu um protótipo de veleiro não tripulado capaz de navegar autonomamente, tendo ainda a flexibilidade de poder carregar *payload* adicional (Cruz & Alves, 2010). Este protótipo é inspirado em modelos de veleiros de competição, tendo sido construído com materiais compósitos e tendo um tamanho de 2,5m. O FASt está equipado com diversos sensores com GPS, agulha magnética, inclinómetro, anemómetro, indicador da direção do vento e sensor de posição da retranca. Além disso conta também com sensores de luminosidade, sensores de temperatura, alagamento e BMS (*Battery Management System*). É possível controlá-lo em modo RC (*Remote control*) ou deixá-lo navegar autonomamente, cum-

prindo com um planeamento de navegação. O leme é movimentado através de um controlador PI (proporcional-integral) e as velas são mareadas segundo um diagrama polar previamente construído, em que são determinadas proas ótimas a seguir em todas as direções de vento possíveis. Está equipado com diversos tipos de antenas para controlo e transmissão de dados, de curto alcance e de longo alcance (satélite). Dispõe ainda de autossuficiência energética com recurso a painéis solares (Marques, 2015).

Figura 2 Veleiro autónomo FASt (imagem adaptada de www.nauticapress.com/)

Vela robótica na Marinha Portuguesa

No ano de 2010, foi definido as linhas orientadoras do projeto eVentos e desde aí que têm surgido teses de mestrado no âmbito deste projeto. Em 2011, foram elaboradas as primeiras teses sobre este tema, tendo por base um *Laser* RC que foi reconvertido para navegação autónoma e onde foi implementado um algoritmo de comando e controlo. Após esta tese foi identificado que este monotipo RC não dispunha a robustez e espaço suficiente para todo o *payload* e só em 2015 foi construído um veleiro totalmente de raiz capaz de responder a estas necessidades. Este protótipo participou numa regata internacional de veleiros com estas características a *World Robotics Sailing Conference* em 2016

e em diversos exercícios de demonstração de capacidades no contexto da Marinha Portuguesa como o REX (*Robotic Exercise*). Esta dissertação baseia-se neste último veleiro, corrigindo problemas estruturais identificados na versão anterior, desenvolvendo melhorias significativas no código interno do sistema e GCS (*Ground Control Station*).

Definição do problema

Um veleiro autónomo para conseguir fazer uma navegação sem interferência contínua de um utilizador e ser energeticamente independente, tem que ter uma série de elementos. Este problema poderá ser estruturado da seguinte forma:

1. **Plataforma:** Casco, leme e velas.
2. **Computador de bordo** que corresponde ao de todo o sistema que está programado em três camadas:
    a. Aquisição de dados dos sensores;
    b. Processamento e planeamento;
    c. Acionamento de atuadores.
3. **Sensores essenciais**:
    a. Sistema de posicionamento GNSS (*Global Navigation Satellite System*);
    b. Bússola;
    c. Sensor direção do vento;
    d. Sensor de posição das velas;
4. **Atuadores**:
    a. Velas;
    b. Leme;
5. **Comunicações** com terra que numa fase inicial permitem detetar anomalias e corrigir lacunas e numa fase final servem para enviar dados de interesse, e receber informação do veleiro;
6. **Sistema de produção e armazenamento de energia**:
    a. Baterias;
    b. Produção de energia, recorrendo a painéis solares ou turbinas quer eólicas quer por efeito de arrastamento na água;

7. **Sistema de controlo em emergências**, que se sobrepõe a tudo o resto:
    a. Anticolisão;
    b. Alagamento;
    c. Nível de bateria baixo;
8. ***Payload***, que pode variar consoante a missão a desempenhar, podendo ser instrumentos com aplicações hidro-oceanográficas, *surveillance*:
    a. Transdutores acústicos;
    b. Câmara;
    c. ADCP's (*Acoustic Doppler Current Profiler*);
    d. LISST (*Optical Laser diffraction instruments*)**;**
    e. SDR

Para além de todo o hardware enunciado nos pontos anteriores, um veleiro autónomo tem que desempenhar uma série de tarefas complexas para atingir determinado objetivo. Quando se está a planear uma derrota de longa duração, a previsão atmosférica tem de ser tida em conta, assim como os perigos à navegação, como a zona de costa ou zonas de baixios. Como os veleiros operam num meio altamente dinâmico e com condições de mudança súbitas, uma plataforma deste tipo terá de responder de forma rápida às alterações das condições ambientais.

A maior parte dos sistemas de navegação comerciais dos veleiros modernos oferece piloto automático, sendo que o seu uso se encontra bastante divulgado. Estes têm a opção de poder manter a embarcação na proa desejada ou manter um ângulo constante ao vento, com a ajuda de anemómetros. No que toca ao controlo das velas, é normalmente uma função desempenhada pela tripulação dada a complexidade da tarefa. É objetivo desta tese desenvolver não só um piloto automático como também um mecanismo de controlo autónomo das velas, que não é oferecido em soluções comerciais (Ruzicka, 2017).

Existem diversos sistemas comerciais, como por exemplo, *weather routing software*, *chartplotters*, molinetes elétricos, que podem ser úteis para construir um veleiro autónomo, mas esses componentes normalmente assumem que há intervenção humana nalguma das fases, e nenhum deles compila todos estes sistemas num só, nem oferecem controlo automático das velas integrado nesse sistema. Torna-se então necessário que

uma embarcação autónoma tenha em conta todos estes fatores, interligando-os de forma harmoniosa.

Por norma, estes tipos de plataformas são controlados parametricamente, por funções escritas em *hardcode*, sendo que são despoletadas consoante os inputs gerados pelos sensores. Qualquer capacidade de adaptação a situações novas está limitada ao algoritmo escrito no microcontrolador, impedindo o protótipo de se ajustar em tempo real (Fernandes, 2016). Existem vários algoritmos de piloto automático e controlo das velas baseados em métodos como *fuzzy-sets* (Stelzer, Proll, & John, 2007), *neural networks* e outras técnicas de inteligência artificial (Aartrijk, Tagliola, & Adriaans, 2002) como algoritmos evolutivos (dos Santos & Goncalves, 2019) que dirigem a plataforma de modo a otimizar e ajustar a trajetória da embarcação e aperfeiçoá-la ao longo do tempo.

Para além do controlo do piloto automático para manter o navio no rumo desejado, é necessário também proceder à afinação das velas para tirar o máximo rendimento. Assim, a otimização de um veleiro é uma tarefa complexa que depende do planeamento da trajetória ótima, do controlo do leme e afinação das velas (Alves & Cruz, 2008).

Os sistemas de controlo de veículos autónomos podem ser divididos em dois tipos arquiteturas diferentes:

- ***Top-down planner-based*** - consistem em sistemas que conhecem o ambiente *a priori* e que em conjunto com dados adquiridos por sensores, geram um modelo que permite traçar um planeamento. Estes mecanismos de planeamento, formulam um plano detalhado para as ações do robô, de forma a alcançar um determinado objetivo definido anteriormente. Assim, após ser traçado o plano, o robô apenas o vai cumprir. Estes sistemas têm mostrado boa *performance* em ambientes complexos e estáticos, mesmo tendo em conta que muitas vezes o desenvolvimento deste plano é uma tarefa significativamente lenta. Ainda assim estes sistemas não conseguem reagir rapidamente num ambiente dinâmico e imprevisível.
- ***Bottom-up reactive*** – esta abordagem conecta diretamente os sensores aos atuadores. Assim, o robô consegue responder rapidamente a alterações no ambi-

ente, como por exemplo rajadas de vento ou objetos em movimento. Esta arquitetura tem mostrado bons resultados em ambientes de constante mudança. Partindo do pressuposto que o robô apenas reage a informação instantânea, sem conhecimento do panorama global, pode não conseguir ser eficaz, embora seja eficiente. Estão ainda limitados a tarefas mais complexas, com muitas variáveis, pois têm de processar dados rapidamente (Stelzer, Le, & Jafarmadar, 1998).

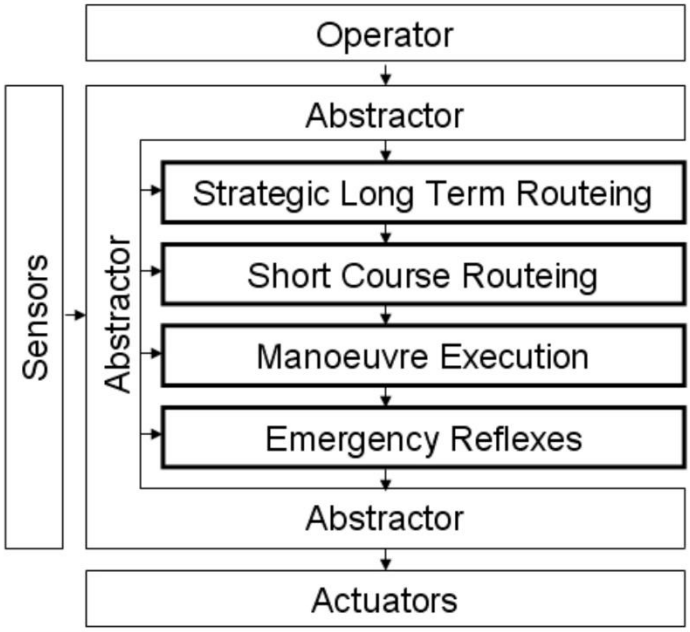

Figura 3 Arquitetura de um veleiro autónomo (Stelzer et al., 1998)

Stelzer sugere uma arquitetura de sistema demonstrado no diagrama da Figura 3. Este sistema é capaz de executar navegação completamente autónoma para um determinado *waypoint*. Só existe interação com o utilizador para introduzir determinados pré-requisitos estratégicos.

O controlo do sistema é dividido em quatro camadas. Cada camada tem acesso aos dados dos sensores através ***Abstractor*** que é nada mais que um algoritmo que compila os dados brutos, executa um pré-processamento aos mesmos, executa transformações de unidades, verifica determinadas condições e exporta-os numa semântica capaz de ser lida

pelas outras camadas. Como semântica universal para partilha de dados em ambiente marítimo, temos o exemplo do protocolo NMEA.

A camada ***Sensors***, é responsável por gerar constantemente e em tempo real dados sobre direção e intensidade do vento, atitude do veleiro nos três eixos, posição das velas e consequente ângulo de ataque ao vento, posição geográfica, rumo. Estes dados são os considerados fundamentais, mas opcionalmente podem ser adicionados sensores acústicos, radar, AIS ou mesmo sensores óticos para calcular rumos por forma a evitar a colisão, podem ser inseridos dados de batimetria e previsões meteorológicas. Baseado nestes sensores o sistema calcula a melhor posição para o leme e velas, sendo estes os únicos atuadores necessários à navegação autónoma.

A camada ***Operator*** é também necessária pois apesar destes veleiros serem completamente autónomos, torna-se necessário definir certos objetivos introduzidos pelo utilizador. Mais concretamente podem ser *waypoints* de chegada ou intermédios, que são introduzidos antes ou durante a navegação. Depois desta fase o operador deixa de ter interferência no planeamento de derrotas ou execução de manobras.

A camada ***Strategic long term routeing***, é a camada mais geral de decisão. Tem em conta dados de análise ao momento, dados de previsão a curto alcance (três a cinco dias) e dados climatológicos. Gerar uma derrota é então o procedimento onde o rumo ótimo é calculado, para um dado veleiro específico num momento específico, com base nos fatores meteorológicos e oceanográficos esperados. São eles, por exemplo, vento, corrente e estado do mar. O algoritmo de derrota gera assim um caminho ótimo, grosseiro, num contexto geral, para determinadas características de comportamento do veleiro, para determinadas previsões futuras. Este conjunto de *wapoints* passa então para a camada ***short course routeing***.

Ao contrário dos navios de propulsão mecânica, nos navios à vela, a forma mais rápida de chegar a um determinado *waypoint* de destino pode não ser uma derrota ortodrómica ou loxodrómica, dado que não é possível navegar-se contra o vento, tendo de se cumprir um planeamento de navegação complexo. A navegação depende, pois, de vários parâmetros, instantâneos e futuros, como a direção e a velocidade do vento e corrente, o período e a altura das ondas e o desempenho da navegação do navio, conjugados num ambiente

não-linear e altamente dinâmico (dos Santos & Goncalves, 2019). Além disso, devido à limitada previsibilidade das condições meteorológicas, a melhor solução tem que ser determinada ou ajustada dinamicamente durante uma viagem (Alves & Cruz, 2008). Assim surge a necessidade de criar algoritmos de ***short course routeing***. Depois de ter uma derrota grosseira gerada na camada anterior, torna-se necessário aplicar as especificidades da navegação à vela face a mudanças locais da previsão meteorológica. Nem todas as mareações de vento são navegáveis, e, nem todas as mareações têm o mesmo rendimento[2], portanto, torna-se necessário fazer alterações à primeira derrota, acrescentando manobras de viragem por d'avante ou em roda, de forma a otimizar a *velocity made good*[3] (VMG), se for esse o objetivo pretendido.

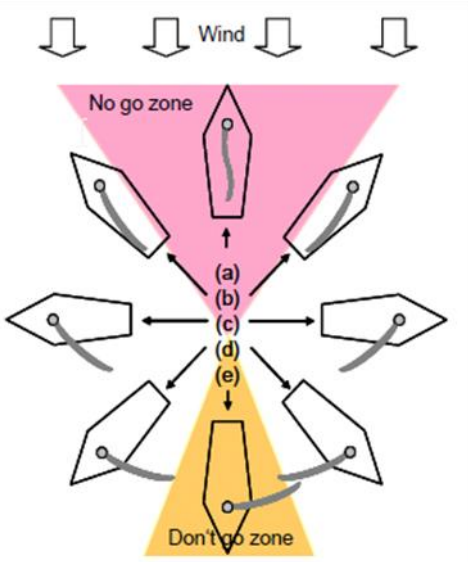

Figura 4 Mareações vento (Stelzer et al., 1998)[4]

O objetivo em ***short course routeing*** é encontrar uma maneira ótima de chegar ao próximo *waypoint* da derrota gerada na camada anterior. Para isso terá de constantemente

[2] Para as mesmas condições de vento e melhor mareação das velas possíveis, diferentes ângulos ao vento produzem diferentes velocidades (*polar chart*).

[3] Decomposição do vetor da velocidade na direção (ou a favor) do vento.

[4] A mareação "No go zone" advém do facto de ser impraticável a navegação à vela, não sendo possível mobilizar força a partir do vento. A mareação "Don't go zone" além de ser substancialmente menos eficiente que uma mareação pela alheta (na maioria dos veleiros), é ainda muito instável, sendo por isso evitada.

responder à questão de quando virar por d'avante (*tack*) quando se navega contra o vento e quando cambar (*jibe*), navegando a favor deste, Figura 5. Esta camada não necessita de previsão meteorológica, reagindo apenas a variações nas condições de vento em tempo real, alimentando a camada seguinte com a proa ótima recursivamente calculada. Os perigos à navegação têm também de ser considerados no cálculo da proa ótima.

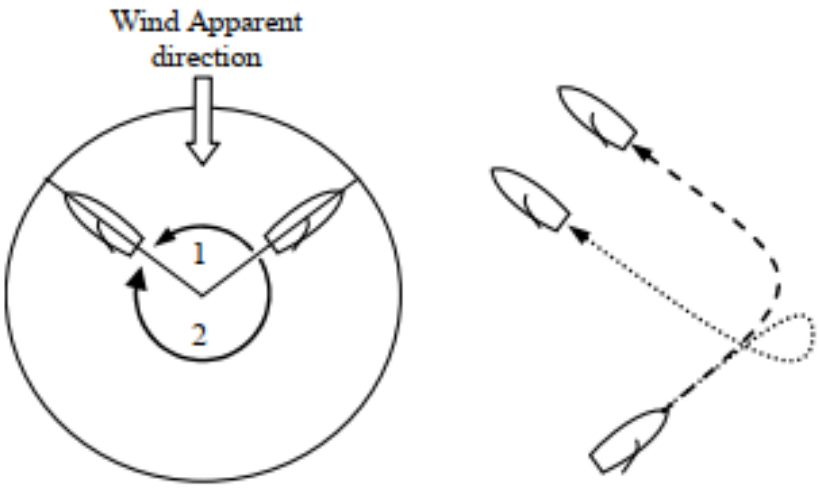

Figura 5 *Tack* (1, --) e *Jibe* (2, ..) (Rynne & Von Ellenrieder, 2010)

Após calculada a proa ótima, passamos à camada ***manouvre execution***. Se o veleiro não estiver na proa correta, o sistema faz ajustes na posição do leme, com o objetivo de trazer o veleiro à proa desejada. Em paralelo outro sistema assegura que as velas estão na posição de máximo rendimento para determinada direção e intensidade de vento.

Por fim, chegamos à camada ***Emergency reflexes***. Esta camada tem um nível hierárquico superior a todas as outras, ou seja, apenas em caso de emergência e se as manobras da camada anterior estão a pôr em risco a segurança da plataforma. Estas ações de reflexo incluem a mitigação do risco de adornamento excessivo da plataforma em caso de grandes rajadas ou ondulação inesperada, diminuição de consumos energéticos em caso de bateria fraca, risco eminente de colisão ou encalhe, entre outros (Stelzer et al., 1998).

Aplicabilidade dos veleiros autónomos

O exposto anteriormente evidencia que os veleiros autónomos não tripulados podem ter especial relevância no contexto da Marinha Portuguesa, sendo particularmente úteis para missões de longo alcance e autonomia com custos de missão muito reduzidos. Os veleiros autónomos podem incluir sensores (incluindo sensores rebocados) que transmitem dados e localização em tempo real e passam despercebidos por desenvolverem baixos níveis de ruido (Alves & Cruz, 2008). Este tipo de plataformas flutuantes "inteligentes" é geralmente direcionado para a monitorização oceânica, mas há também outras aplicações que podem ser desempenhadas, tais como o controlo de populações e rotas da mamíferos marinhos (Mordy et al., 2017), o transporte de mercadorias sem emissões de gases poluentes, desempenho de funções de navio reabastecedor ou de operações de desminagem (Stelzer & Jafarmadar, 2011b).

Os principais desafios do desenvolvimento futuro dos veleiros autónomos na Marinha Portuguesa podem passar pela integração de *hardware*, desenvolvimento de algoritmos para gerar rotas, soluções para a autossuficiência energética e sistemas anticolisão, como evidenciado por (Stelzer & Jafarmadar, 2011b). É expectável que no futuro haja uma utilização cada vez mais acentuada deste tipo de veículos, sendo de particular interesse o estudo e prototipagem de veleiros autónomos nesta instituição.

Objetivos

Esta dissertação tem como objetivo a investigação do desenvolvimento e aplicação de métodos de navegação totalmente autónomos na plataforma Barlavento, decorrente do projeto eVentos. Esta investigação passa assim a desdobrar-se em diversas etapas:

1. Planear e otimizar uma rota para um dado *waypoint* com base em medições locais do vento e corrente;
2. Previsão de derrotas de longa duração com base em previsões meteo-oceanográficas;

3. Responder a alterações das condições atmosféricas *in situ* (*short term routeing*), produzindo modificações à trajetória inicial (*strategic long term routeing*);
4. Executar de forma independente viragens de bordo por d'avante (*tack*) ou em roda (*gybe*) por forma a cumprir trajetórias predefinidas;
5. Aquisição de contactos AIS (*Automatic Identification System*) com SDR (*software defined radio*) e desenvolver algoritmos anticolisão;
6. *Surveillance* com recurso a SDR;
7. Executar navegação completamente autónoma;
8. Otimizar a gestão da energia a bordo;
9. Introduzir um novo sistema de produção de energia e controlo de carga das baterias;
10. Implementação do protocolo NMEA para transferência de dados;
11. Construção de um anemómetro;
12. Construção de novas velas;
13. Construção de novas portas de visita;
14. Organização da cablagem em *shields* para *arduino*;
15. Testar o veleiro e analisar dados.

Estrutura da Tese

Esta dissertação de mestrado está dividida em seis capítulos fundamentais. O Capítulo 1, descreve a forma como foram abordados e construídos todos os *upgrades* fundamentais que tinham sido identificados como necessários na primeira versão do protótipo assim como outros que o autor achou conveniente construir. O Capítulo 2, descreve como foi reorganizada a eletrónica e novos sensores do veleiro. No Capítulo 3, é abordada toda a parte respetiva à programação do veleiro e a forma como é executado o comando e controlo da plataforma. No Capítulo 4, são descritos de que forma foram executados testes, resultados obtidos e discussão dos mesmos. No Capítulo 5 são propostas linhas de investigação para trabalho futuro.

1 Constituintes estruturais da plataforma

Verificou-se a necessidade de efetuar melhorias ao veleiro, devido a falhas de projeto ou de construção, que colocavam em risco a operação em segurança do mesmo.

Ao nível da segurança, as duas principais melhorias implementadas foram as portas de visita e as velas. As portas de visita originais não asseguravam estanqueidade, pois, encontravam-se deformadas. Na Figura 6 encontra-se uma fotografia das portas de visita originais.

Figura 6 Primeira versão portas visita

O veio do mastro era em perfil circular de alumínio, com diâmetro de 14 mm e parede de 2 mm, fletindo bastante quando a navegar e por ação do vento. Além disso, a superfície da vela deformava, perdendo qualquer formato aerodinâmico e perdendo rendimento,

funcionando meramente por arrasto. Na Figura 7 encontra-se uma fotografia das velas originais.

Figura 7 Primeira versão das velas

Também se verificou necessário proceder a outros melhoramentos, sendo os mais importantes: anemómetro, cablagem, placas de controlo, bateria e *payload* eletro-ótico.

Neste capítulo e seguintes serão descritos os trabalhos efetuados, divididos em 3 áreas distintas: plataforma, hardware eletrónico e por fim programação.

A componente estrutural de um veleiro é muito importante, pois é este que garante a segurança da eletrónica e do *payload*, tendo uma função vital para que o mesmo consiga atingir o objetivo. Neste subcapítulo são descritas as principais melhorias efetuadas no veleiro.

1.1 Mastro e Velas

O mastro é um componente de importância indiscutível num veleiro. Queremos garantir que, para dados critérios de operação, o mesmo irá resistir intacto. O dimensionamento do novo conjunto de mastro e vela foi efetuado com recurso à ferramenta de CAD

3D e de simulação *Solidworks* 2018 *Student Edition*. A sequência de escolhas técnicas para dimensionamento do mastro foi a seguinte:

1. Escolha de perfil NACA;
2. Modelação da vela;
3. Dimensionamento do mastro;
4. Cálculo dos limites de operação do mastro projetado.

Um fluido a passar num perfil aerodinâmico gera duas forças, a força de sustentação ou *lif*t, e a força de arrasto ou *drag*. Na equação 1 e 2 encontra-se as respetivas equações.

$$F_l = \frac{1}{2}.\rho_a.V^2.A.C_l$$

Equação 1 Força *lift*

$$F_d = \frac{1}{2}.\rho_a.V^2.A.C_d$$

Equação 2 Força *drag*

Onde V é a velocidade aparente do vento, ρ_a, a densidade do ar e A, a área da superfície vélica.

Relativamente à escolha do perfil NACA (*National Advisory Committee for Aeronautics*), este tem de ser simétrico, deve ter boas capacidades aerodinâmicas com um baixo número de Reynolds e ainda uma forma espessa para permitir movimentos mais suaves quando a asa roda.

Apesar de um *airfoil* mais espesso ter mais massa, contribui com uma maior área de momento de inércia, melhorando a rigidez estrutural do aparelho.

Ao longo dos anos tem sido produzida muita investigação científica acerca de perfis aerodinâmicos que não é o objetivo desta tese. Deste modo, foi escolhido um *airfoil* NACA 0018, com base no estado da arte existente (Tretow, 2017).

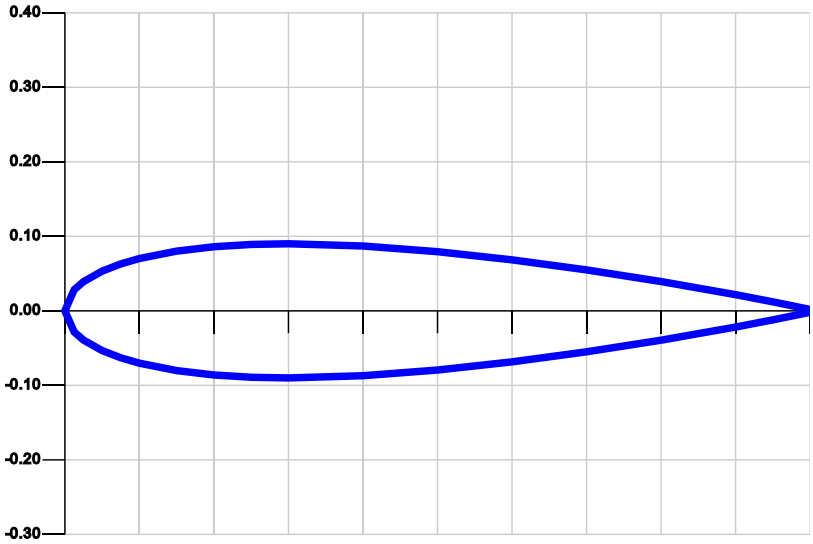

Figura 8 NACA 0018

Thickness	17.9%
Camber	0.0%
Trailing edge angle	31.1º
Lower flatness	4.4%
Leading edge radius	4.8%
Efficiency	46.7
Max C_L	1.077
Max C_L angle	15.0
Max L/D	34.864
Max L/D angle	5.5
Max L/D C_L	0.686
Stall angle	11.5
Zero-lift angle	0.0

Tabela 1 Características NACA 0018

Após escolha do perfil NACA, foi efetuada a sua modelação em CAD, encontrando-se na Figura 9 o arranjo final do mesmo.

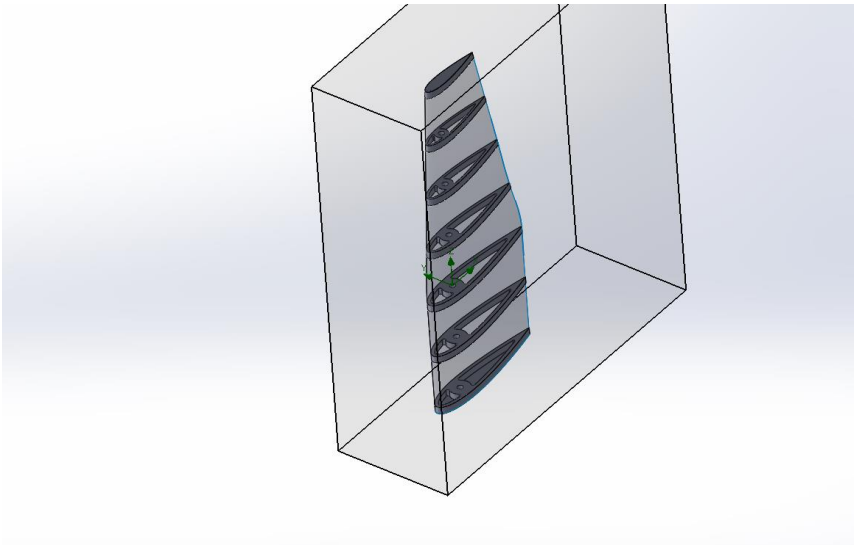

Figura 9 Modelação da vela

Consultando o mercado, verificou-se a existência de um veio circular 16mm e parede de 2mm em fibra de carbono. Simulou-se em CAD um mastro com veio com essas dimensões e características, tendo-se verificado que o mesmo seria capaz resistir até 100N de carga. Na Figura 10, encontra-se a simulação efetuada.

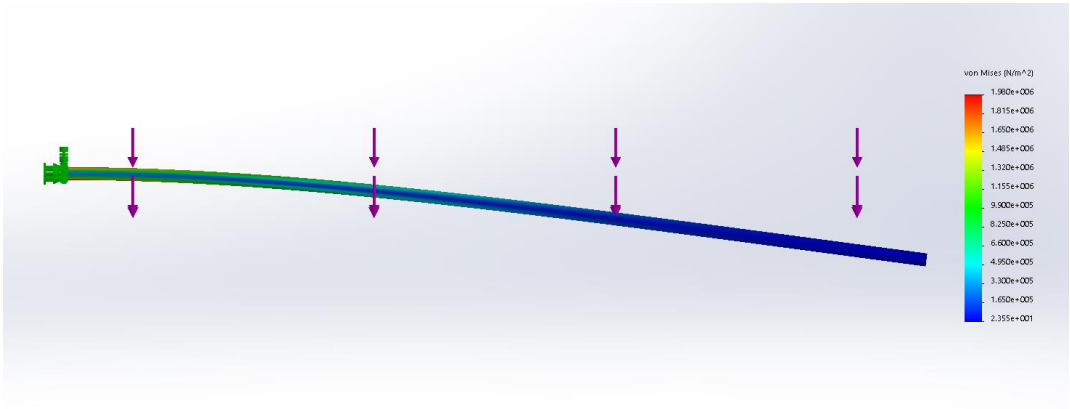

Figura 10 Simulação em *Solidworks* com força 100N

Tendo o valor da carga que o mastro é capaz de suportar, (100 N), podemos resolver a Equação 2, considerando:

1. Força de 100 N;
2. Área de 0,3855 m^2 (acordo *software* CAD);
3. Densidade do ar de 1.225 Kg/m^3;
4. Coeficiente de *drag* de valor unitário (o coeficiente de *drag* será sempre inferior a este valor).

Resolvendo a Equação 2, encontramos um valor máximo de 20.6 m/s para velocidade máxima de vento que a vela desenvolvida é capaz de resistir. Este valor não contempla forças inerciais resultantes da movimentação do veleiro, pelo que deve ser aplicado um coeficiente de segurança. Foi escolhido o valor de coeficiente de segurança 2, pelo que, as velas só poderão ser utilizadas para valores de vento inferiores a 10.3m/s, ou seja, 20 nós. É ainda de referir que o mastro foi colocado na posição a um quarto do valor da corda, por ser o ponto de aplicação das forças aerodinâmicas num *airfoil*.

Depois de efetuado o dimensionamento, foi feito o projeto de construção, o qual foi planeado e executado da seguinte forma:

1. Desenho em *Solidworks* das velas exportado em formato STL (*stereolithography*);
2. Importação dos ficheiros STL para *Vectric Aspire* com o objetivo de gerar caminhos de corte;
3. Exportação para *G-CODE*[5] para ser lido no software da fresadora, *Mach3*;
4. Maquinação, com recurso a uma fresadora CNC (*computer numerical control*), do esqueleto, construindo secções semelhante usando madeira;
5. Maquinação das secções intermédias, na mesma máquina, em espuma XPS (*Extruded polystyrene*), primeiramente com uma fresa de grande desbaste e em seguida com uma mais fina;

5 O Código G, do inglês *G-Code*, é a nome dado à linguagem de programação criada a partir da necessidade de maquinas industriais que fazem uso de sistemas de Comando Numérico Computadorizado (CNC).

6. Colagem das secções em madeira e varas guias de fibra de vidro aos tubos de fibra de carbono (*pullwinding*) com resina *epoxy*;
7. Colagem das secções em espuma XPS;
8. Passagem de lixa na estrutura completa por forma a homogeneizar a superfície;
9. Cobertura de toda a superfície com película de polipropileno que encolhe com aplicação de calor.

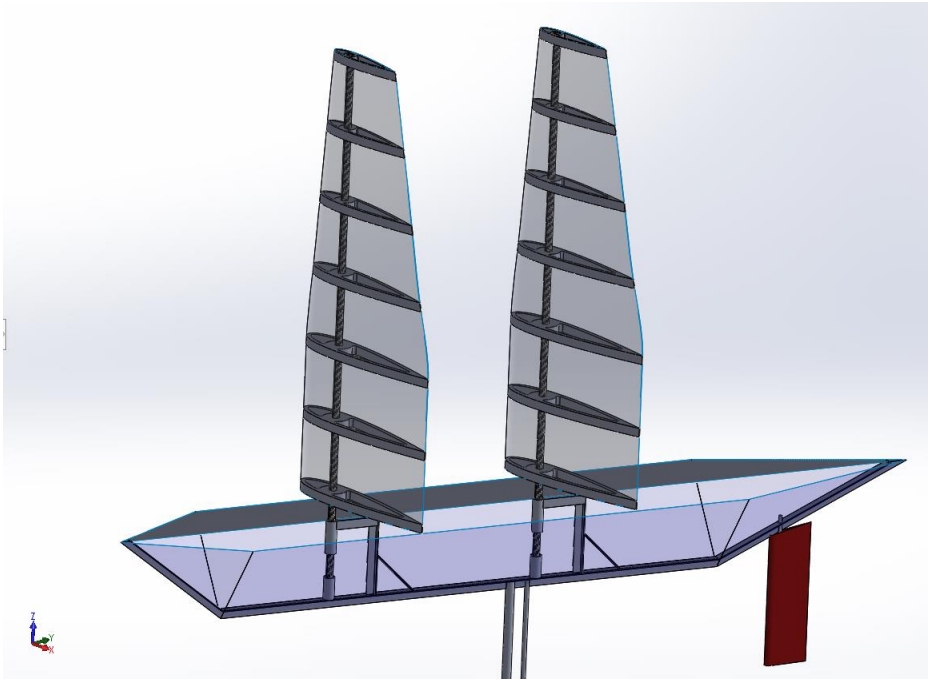

Figura 11 Modelo geral em *Solidworks*

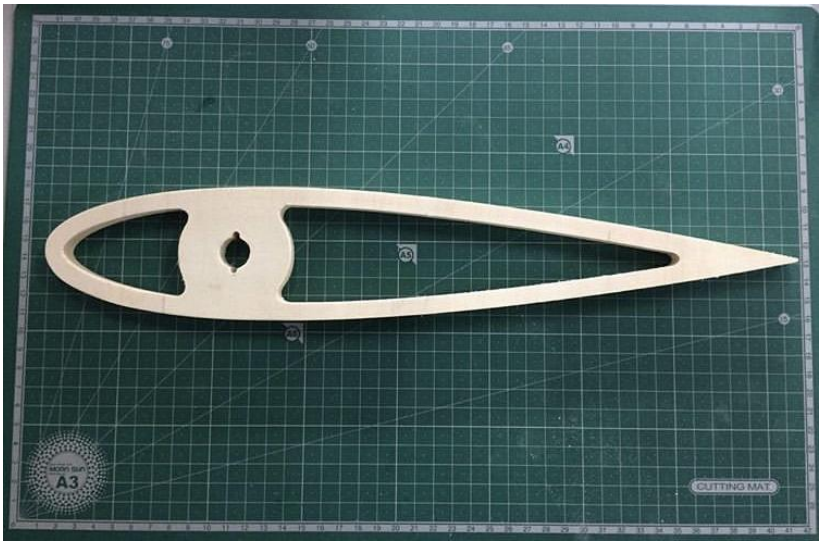

Figura 12 Primeiro teste de maquinação do esqueleto do *Airfoil* NACA 0018

Figura 13 Esqueleto da vela com secções de madeira, depois de maquinadas

Figura 14 Maquinação da espuma XPS

Figura 15 Montagem alternada das secções em madeira e espuma XPS

1.2 Portas de visita

O protótipo desenvolvido anteriormente tinha deficiências devido ao facto de ter sido utilizado PLA (*polylactic acid*), e este se ter revelado inadequado para uma utilização *offshore*, sujeito a temperaturas altas, tendo as portas de visita anteriores deformado significativamente, pondo em risco a estanqueidade do veleiro (Fernandes, 2016). O seu mecanismo de aperto também não era o mais adequado, perdendo-se muito tempo para aceder ao interior do veleiro. O mercado foi consultado, mas não foram encontradas portas de visita com um tamanho aceitável para se poder mexer livremente nos componentes da eletrónica. Identificadas as lacunas da primeira versão das portas de visita procedeu-se à substituição destas por cinco conjuntos de duas peças redondas, impressas em ABS (*acrylonitrile butadiene* styrene) e CPE+ (*co-polyester*).

Por forma a aproveitar a intrusão que se fez aquando da colocação das portas antigas, foram desenhadas em *Solidworks*, quatro conjuntos de diâmetro 228 mm e um conjunto de 155 mm, com desenho semelhante. A vedação entre as duas peças é feita através de um *O'ring*, que é esmagado quando se enrosca a tampa, Figura 16, no suporte, Figura 17.

As duas peças novas, são mais fáceis de manusear e não têm componentes metálicos sujeitos a oxidação.

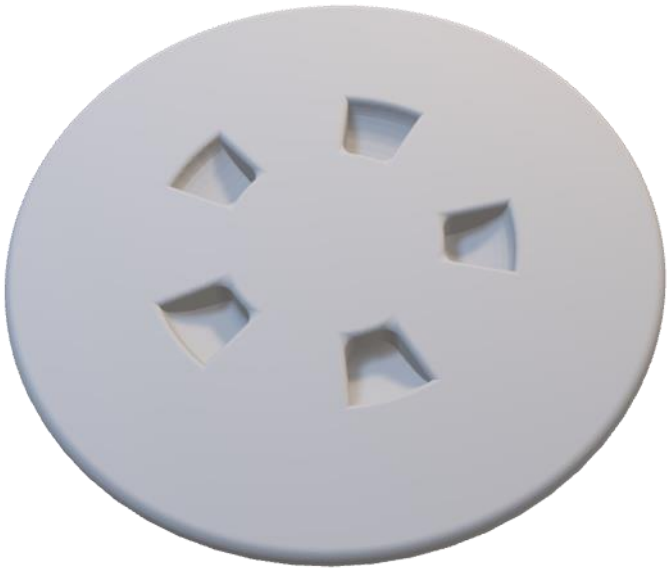

Figura 16 Tampa da porta de visita desenvolvida em *SolidWorks*

Figura 17 Suporte da porta de visita desenvolvido em *SolidWorks*

Após impressão das portas de visita, foram retiradas as portas de visita antigas e aberto um rasgo em volta da área onde estavam colocadas. Este rasgo foi aberto com o intuito

de facilitar a colocação de mais material compósito, neste caso fibra de vidro com resina de poliéster.

Figura 18 (1) Colocação de fibra de vidro após abertura do rasgo (2) Casco após ter sido lixado o excedente do material compósito (3) Colocação de Betume (4) Casco pintado

Seguidamente, o casco foi todo lixado, retirando a maior parte do excedente do material compósito e foram abertas novas aberturas redondas com vista à colocação das novas portas de visita.

Após esta fase foi colocado e posteriormente lixado, betume por cima da fibra com o objetivo de conferir maior proteção à mesma e preparar para a pintura.

Para finalizar, foi pintado o convés de cor azul e colocadas as portas de visita nas suas respetivas posições, garantindo a vedação do rebordo com silicone marítimo, **Error! Reference source not found.**.

1.3 Bucim dos mastros

Com a construção de novas portas de visita, o sistema de bucim já implementado entrava em conflito com as novas tampas das portas de visita, impedindo a rotação dos mastros tornando-se necessário produzir um novo sistema estanque. Não obstante, o sistema antigo não produzia um isolamento completo, deixando algumas fugas para a entrada de água. O novo bucim construído foi previamente desenhado em *SolidWorks* e impresso por manufatura aditiva, Figura 19.

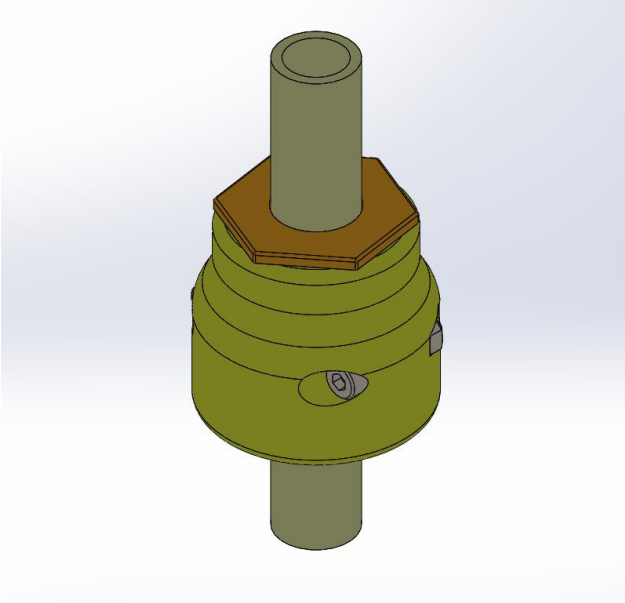

Figura 19 Vista isométrica do bucim dos mastros

Esta peça consiste na união de uma peça inferior dividida em duas metades e apertada com parafusos sextavados interiores M4x20 mm, com rosca e parte superior sextavada. A estanqueidade é garantida através de retentor de diâmetro interno 16 mm, no seu interior. Na parte inferior existe também um *O-ring* que é esmagado contra o rebordo embutido no casco, Figura 20 (a verde), que foi desenhado para efeitos de demonstração e verificação das medidas. Visto que este conjunto está exposto a fatores ambientais, foi também impresso com material mais resistente à ação UV (ultravioleta), temperatura de deformação e quimicamente mais resistente (CPE+) (Ultimaker, sem data).

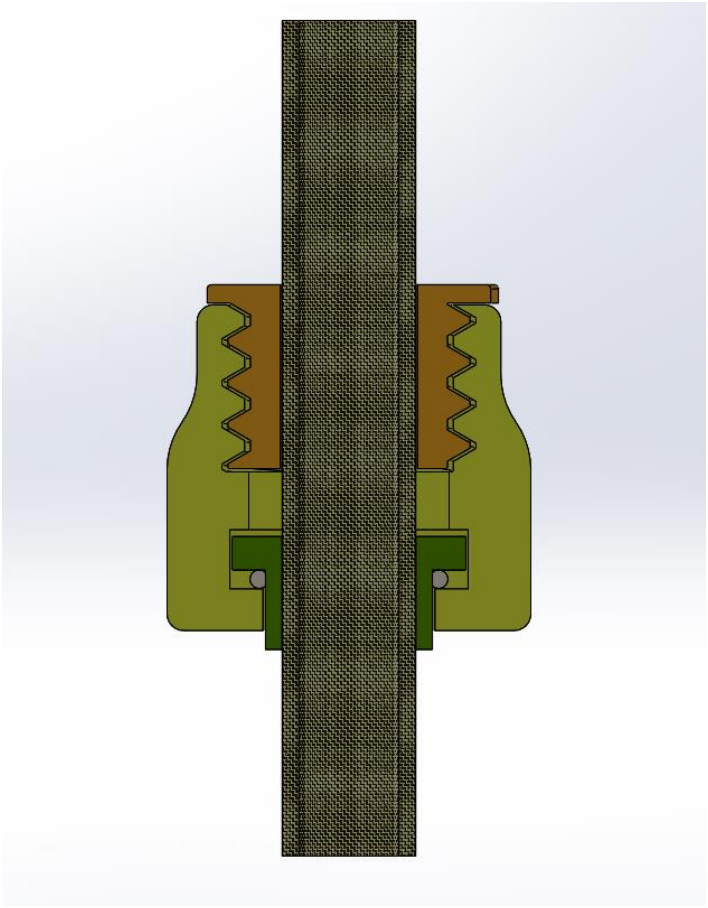

Figura 20 Corte longitudinal pelo plano yz

1.4 Suporte câmara

Para visualização do comportamento do veleiro a navegar e comparação com os dados recebidos na GCS, procedeu-se à construção por manufatura aditiva de um suporte para uma câmara, que enrosca num parafuso embutido no casco que se encontra por cima do leme. Este suporte, Figura 21, permite levar qualquer *action cam* de formato de encaixe padronizado. No caso particular do veleiro Barlavento foi colocada uma GoPro.

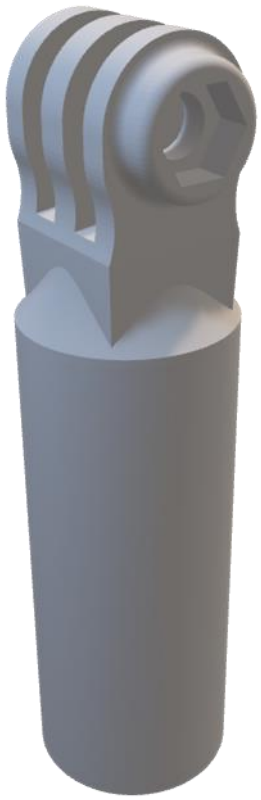

Figura 21 Suporte câmara de ação

2 Eletrónica

No início deste projeto foi verificado que os componentes eletrónicos do veleiro já não estavam presentes. Partindo deste pressuposto, foi necessário adquirir novos sensores e reconstruir toda a eletrónica do veleiro. Este capítulo descreve como foi organizada a eletrónica bem como o desenvolvimento de placas de circuito impresso, construção e calibração de um anemómetro e outros sensores periféricos. É ainda descrito o módulo de comunicações utilizado e bateria.

2.1 Arquitetura do sistema

Em termos de *hardware* o sistema é composto por:

- GPS Neo7mV2 – atualiza a posição com frequência de 10 Hz;
- IMU BNO055 – para além da proa também indica o *pitch* e *roll*;
- Servo Hitec HS-765HB – regula a posição do leme com binário máximo de 13,2 Kg/cm@6,2V;
- Transcetor E32-DTU – permite a comunicação com a estação em terra através do protocolo LoRa, a uma distância máxima de 8Km à potência de 1W;
- Anemómetro – velocidade e direção do vento;
- 1x Bateria LiPo 6S 10Ah;
- 2x Arduino Mega 2560;
- Leitor de cartões de memória;
- Controlador dos motores *Veyron* (Fernandes, 2016) – controlo da posição dos mastros;
- 2x Motores DC com caixa redutora;
- 3x Conversores DC-DC;
- Ventoinha.

O sistema é constituído por dois Arduínos MEGA 2560 sendo que o principal (*master*) atua diretamente no leme e recebe dados dos sensores:

- Posição, hora, rumo e velocidade do GPS, através do protocolo UART;
- Ângulo de adornamento, ângulo de cabeceio e proa da IMU (*inertial measurement unit*), através do protocolo I^2C;

- Velocidade do vento, contando e convertendo o número de rotações do anemómetro através de saída analógica;
- Direção do vento através do *encoder* alojado no anemómetro, através de PWM *(Pulse-Width Modulation);*

O Arduino secundário (*slave*), recebe a posição dos potenciómetros dos mastros através da leitura do seu valor analógico, lê as posições dos *waypoints* do cartão de memória e recebe do outro Arduino todos os outros dados. Em paralelo é calculada a proa a seguir e o servo do leme é atuado pelo *master* até que esteja no caminho certo. As velas são também ajustadas regularmente pelo *slave* para conseguir o ângulo de ataque ao vento relativo desejado. O Arduíno *master* comunica também com o módulo de comunicações por interface série.

A estação em terra recebe feedback de todo o sistema permitindo ver em tempo real o comportamento do veleiro. Em simultâneo é criado um log no cartão de memória e na estação em terra com o registo de todas as variáveis para poder ser analisado mais tarde.

Por forma a organizar e integrar todos os periféricos do sistema, foram desenvolvidas duas placas como veremos adiante.

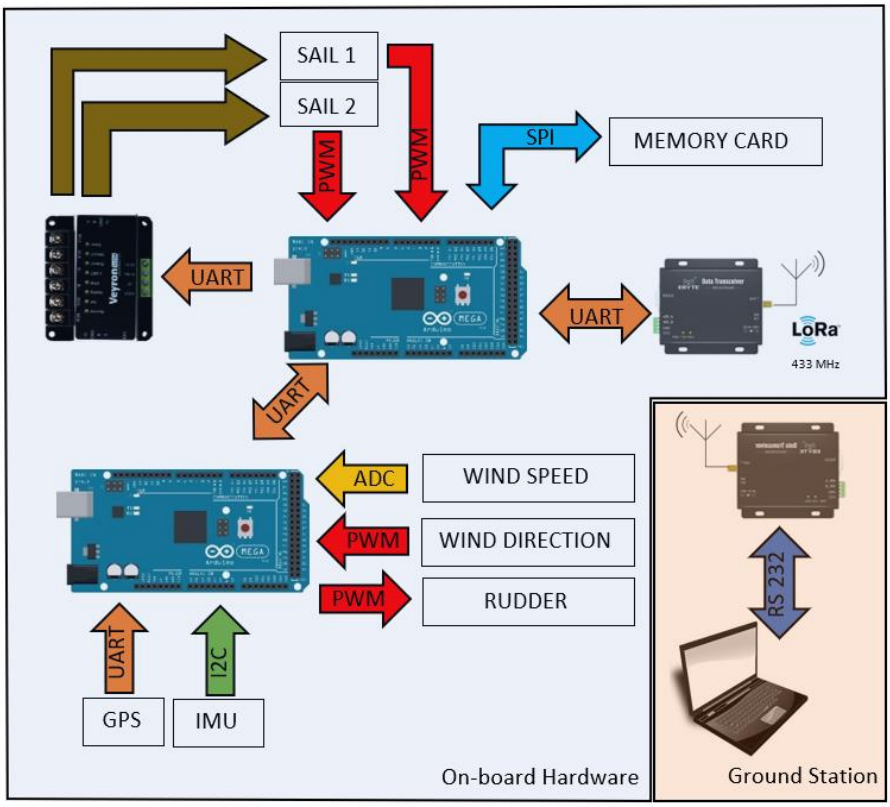

Figura 22 Arquitetura do sistema

2.2 *Shields* PCB para Arduíno

A primeira versão do veleiro autónomo Barlavento, não estava organizada e continha muitos cabos elétricos, dificultando por um lado o despiste de avarias e por outro o acondicionamento durante a navegação, soltando-se regularmente componentes. Por forma a colmatar estas necessidades foram desenvolvidas placas de circuito impresso (PCB), por forma a ligar sensores, colocar componentes eletrónicos fixos à placa e garantir a identificação e organização dos componentes.

Foram desenhadas, através do *software Eagle*, dois *shields* que encaixam nos pinos dos dois Arduinos do sistema. A primeira placa, Figura 23, foi denominada de distribuição de energia pois é o primeiro contacto com a bateria. Conta com ligações onde são encaixados cabos dos potenciómetros das velas, tem ligações série com o módulo controlador de motores DC (*Veyron*) e com o Arduíno principal. Conta também com ligações

vazias para se poderem ligar outros periféricos se necessário, interruptores e *leds* para *debug*.

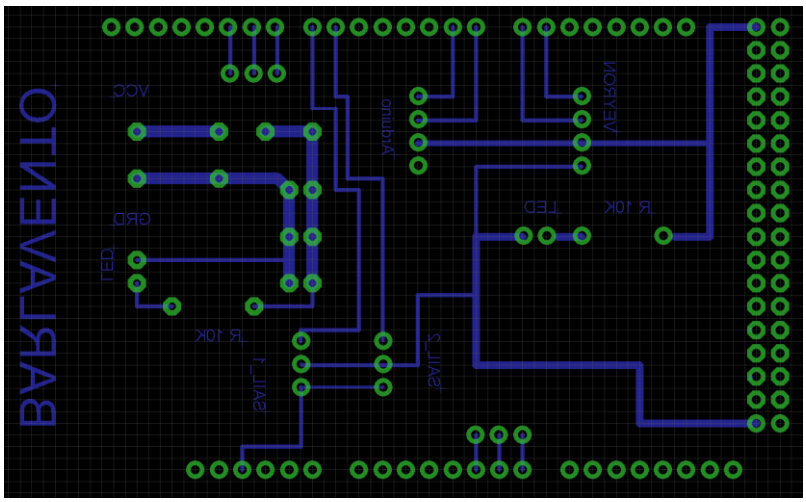

Figura 23 PCB de distribuição de energia desenvolvida em *Eagle*

A segunda placa, Figura 24, denominada placa de sensores, tem acoplados todos os outros sensores de bordo (GPS, IMU, anemómetro) bem como o leme e módulo de comunicações. Conta ainda com comunicações série com o outro Arduíno. Só o leme é alimentado pela placa, todos os outros sensores têm alimentação externa através de um módulo conversor DC-DC de 12V para 5V, para não sobrecarregar o Arduíno.

Para além das placas existem também dois conversores DC-DC que convertem a voltagem de 22,2V da bateria para 12V, alimentando depois os dois Arduínos, módulo de comunicações e também uma ventoinha que arrefece o Arduíno dos sensores.

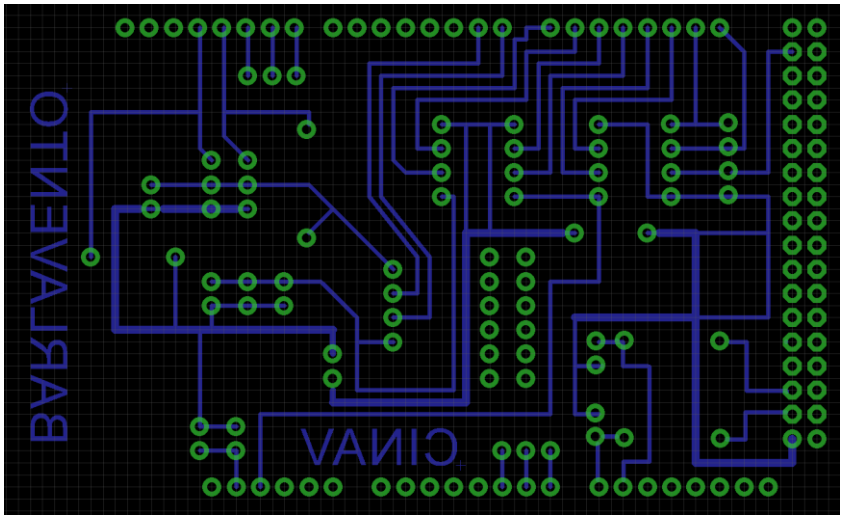

Figura 24 PCB de sensores desenvolvida em *Eagle*

Relativamente ao processo de construção, dividiu-se nas seguintes etapas:

1. Desenho das placas PCB no software *Eagle*;
2. Impressão em papel de acetato do circuito;
3. Gravação do circuito numa placa de cobre coberta com película fotossensível à luz ultravioleta, com recurso a uma lâmpada ultravioleta;
4. Revelação do circuito, mergulhando a placa numa solução de bicarbonato de sódio e água;
5. Corrosão do cobre excedente aos circuitos numa solução de percloreto de ferro, aquecida;
6. Furação dos pinos dos circuitos com uma broca adequada.

2.3 Sensor de vento

Para calcular o rumo ótimo e a posição das velas relativas ao vento, é fulcral adquirir dados sobre a direção do vento. A velocidade do vento não é tão importante, mas poderá ser útil para a construção de gráficos polares de *performance* ou para desencadear mecanismos de proteção em caso de vento excessivo.

Existem essencialmente três tipos de instrumentos com capacidade de medir o vento (Carlos Manuel Moreira Alves, 2010):

1. Sensores puramente mecânicos que usam potenciómetros para medir a direção do vento;
2. Sensores mecânicos sem contacto que usam imanes e sensores de hall para calcular a direção do vento;
3. Sensores ultrassónicos que medem a direção do vento detetando o movimento das partículas de ar dentro do sensor

Contudo foi escolhido um anemómetro com cata-vento mecânico sem contacto pela facilidade de construção por manufatura aditiva, sem custos elevados.

Foi construído por manufatura aditiva, um anemómetro por forma a acomodar um sensor de hall bipolar TLE4945L e um *magnetic rotary encoder*, AS5048A.

O sensor AS4048A, Figura 25, é um sensor de posição de ângulo de 360°, alimentado a 3,3V ou 5V, com resolução de 14 bits. O circuito integrado, mede a posição absoluta do ângulo de rotação do íman através de vários sensores de Hall[6]. A informação da posição absoluta do íman é acessível diretamente através de uma saída de PWM. O sensor tolera desalinhamentos do íman, lacunas entre o sensor e o íman de 0,5mm a 2,5mm, variações de temperatura e campos magnéticos externos. Foi escolhido um íman de neodímio por forma a entrar dentro da gama de valores admissíveis pelo sensor, sendo a força do campo magnético perpendicular à superfície admitida de 30mT a 70mT. (Austria Mikro Systeme, 2016).

[6] Um Sensor de Efeito Hall é um transdutor que, quando sob a aplicação de um campo magnético, responde com uma variação na tensão de saída.

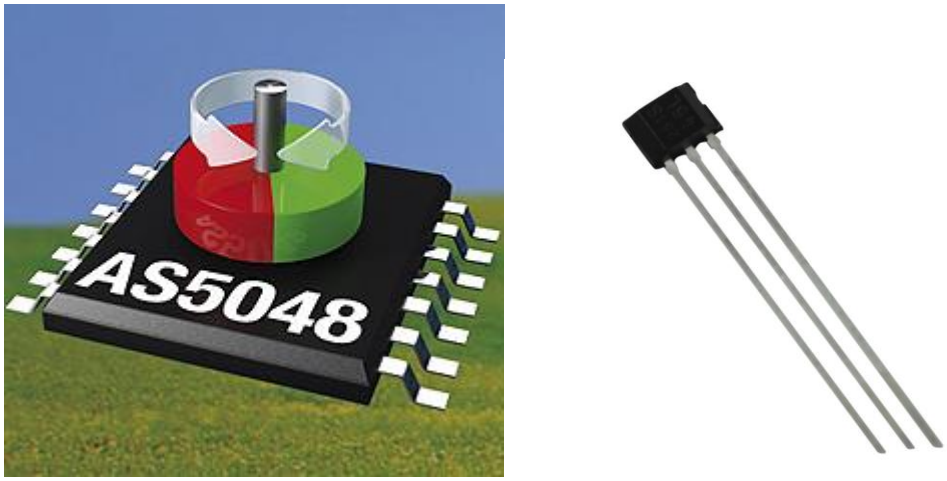

Figura 25 *Magnetic rotary encoder* AS5048A (esquerda) e Sensor de hall bipolar TLE4945L (direita)

O sensor TLE4945L, é um circuito integrado que inclui um gerador de impulsos por efeito de Hall, um amplificador e um *Schmitt-Trigger* num único *chip*. Quando existe um campo magnético perpendicular à superfície do *chip*, este induz uma tensão no sensor de Hall que é amplificada, despoletando o *Schmitt-Trigger* com coletor aberto, Figura 26, que é diretamente proporcional à força do campo magnético, isto é, depois de amplificado o sinal, o *Schmitt-Trigger*, compara este valor com um limiar padrão, se este valor de entrada for maior que o limiar, a saída assume o valor analógico alto, quando a entrada está abaixo do limiar a saída assume valor analógico baixo. Para funcionar nesta configuração, foi necessário instalar uma resistência *pull-up* de 10kΩ entre a saída do sinal e a alimentação (Infineon technologies, 2017).

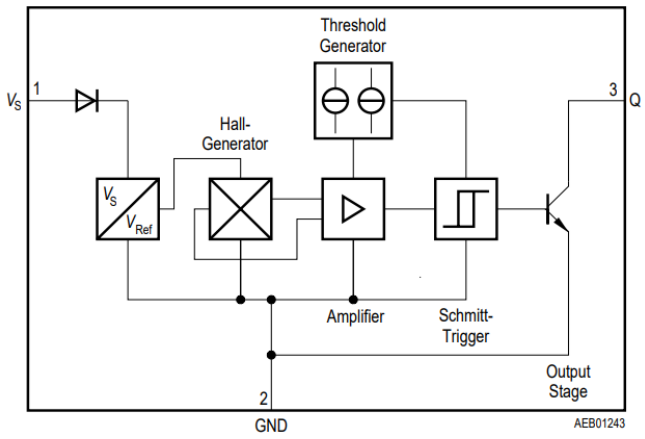

Figura 26 Diagrama de blocos do sensor hall bipolar TLE4945L (Infineon technologies, 2017)

Figura 27 Anemómetro com cata-vento desenvolvido em *Solidworks*

O instrumento da Figura 27, divide-se em duas partes:

1. Na parte superior, um cata-vento que roda um veio, onde está um íman por cima de um *magnetic rotary encoder*, e que nos dá a posição do campo magnético perpendicular ao sensor e, consequentemente, a direção do vento.

2. Na parte inferior, um anemómetro rotativo que faz rodar um veio que por sua vez aloja quatro ímanes dispostos antagonicamente com o objetivo de, ao passar pelo sensor de hall, este produzir alternadamente uma tensão maior ou menor que o valor médio predefinido, consoante esteja próximo de um polo positivo ou negativo, respetivamente. Assim, contando o tempo em que ocorre esta variação é possível calcular o número de rotações por minuto do anemómetro.

Adicionalmente todos os sensores foram cobertos com resina *epoxy*, de modo a garantir uma proteção adicional em contacto com a água.

2.3.1 Calibração

Visto que a parte superior do instrumento nos dá diretamente um valor PWM, é apenas necessário calibrar para o ângulo correspondente, impondo o ângulo zero a meia-nau.

A parte inferior do instrumento acarretou maior complexidade na calibração. Foi necessário corresponder os valores calculados de RPM (rotações por minuto) em velocidade relativa do vento em nós. De forma mais empírica e simples, foi utilizado o túnel de vento *Leybold* da Escola Naval representado na Figura 29 e um anemómetro de bolso, Figura 29, para aferir o valor da velocidade do vento à saída do túnel. Este anemómetro apresenta um erro de leitura de 3% e está limitado aos 81 nós. Para diferentes valores de velocidade do vento do túnel, foram retirados 50 valores de RPM correspondentes a cada velocidade do vento. Seguidamente interpolaram-se esses valores com uma reta, de onde se extraiu a sua equação que por fim foi implementada no código do Arduíno principal, por forma a obter a velocidade do vento em nós. É também de notar que o anemómetro só apresentou sensibilidade de leitura (por inércia dos componentes mecânicos) a partir dos 3,5 nós e que o valor máximo medido foi de 27,5 nós por limitação do túnel.

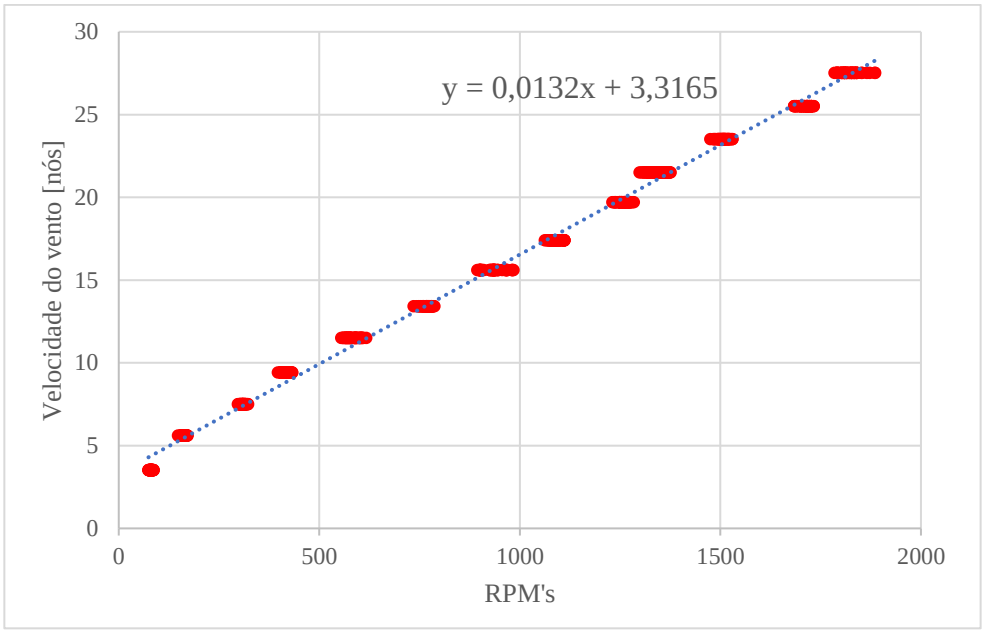

Figura 28 Relação entre RPM's e velocidade do vento

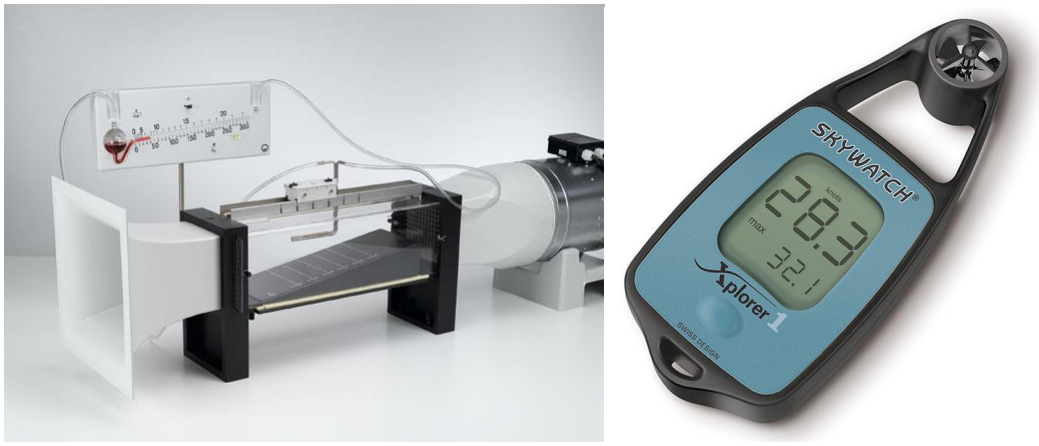

Figura 29 Túnel de vento *Leybold* (esquerda) e Anemómetro *Skywatch Xplorer* 1 (direita)

2.4 IMU

A IMU (*Inertial Measurement Unit*) utilizada neste protótipo foi o BNO055, que consiste num SiP (*System in Package*) que integra um acelerómetro triaxial de 14-bit, um

giroscópio triaxial de 16-bit e um magnetómetro. É possível comunicar através de I^2C ou UART (Sensortec, 2014), alimentando-o a 5V. Este SIP contém bibliotecas próprias que efetuam a calibração do mesmo cada vez que é ligado que não foram exploradas por não ser esse o objetivo da tese. Assim, depois de calibrado corretamente, temos *output* da proa *pitch* e *roll*.

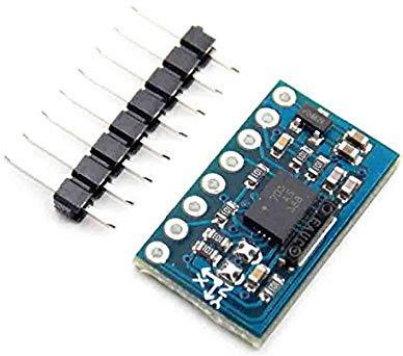

Figura 30 BNO055 imagem adaptada de https://forum.arduino.cc/

2.5 GPS

O *Global Positioning System* (GPS), é um sensor basal em muitos dos veículos, quer sejam eles tripulados ou não. É com base neste sistema que o veleiro adquire a sua posição, rumo e velocidade. Sem estas variáveis a navegação torna-se praticamente impossível, pois é a partir destas que é calculado o azimute e distância para o *waypoint* a atingir, a velocidade verdadeira praticada e o rumo que está a descrever.

O módulo de GPS utilizado foi GPS Neo7m da marca uBlox, que integra também uma antena de cerâmica. Este módulo é alimentado de 3 a 5V, utiliza comunicação série a 9600 *baud* demorando cerca de 30 segundos a obter uma posição fidedigna se ligado "a frio" e quase instantânea se ligado depois de já ter estado a trabalhar.

Figura 31 GPS Neo7m imagem adaptada de https://forum.arduino.cc/

2.6 Comunicações

2.6.1 LoRa

O espectro de dispersão LoRa é uma modulação patenteada desenvolvida pela Semtech baseado no *chirp spread spectrum* (CSS). LoRa (abreviação inglesa de "longo alcance") permite comunicações a longo alcance e baixo consumo de energia a baixa taxa de dados e segurança na transmissão, Figura 32. A tecnologia LoRa pode facilmente integrar-se com as redes existentes sendo usada em aplicações de baixo custo, alimentadas por baterias como é o caso das IoT (*Internet of Things*) (Seneviratne, 2019).

O módulo de comunicações LoRa utilizado foi o E32-DTU, que numa fase inicial do protótipo serve principalmente para fazer o *datalog* de todos os dados gerados no veleiro, sendo um elemento fulcral no *debug* de erros. Numa fase posterior poderá ser substituído por um sistema de transmissão de longo alcance (satélite) assim que seja atingida a fiabilidade necessária do sistema.

Este módulo dispõe de uma entrada RS232[7] para comunicação, mas como o Arduíno apenas consegue comunicar por UART (*Universal Asynchronous Receiver/Transmitter*)

[7] Protocolo padrão de troca série de dados entre um terminal e um comunicador.

TTL (*Transistor to Transistor Logic*), foi necessário converter o sinal usando um módulo MAX232[8] e um *null modem* para trocar o *pinout*.

Frequência	**Potência TX**	**Distância**	***Baud rate (default)***	***Air rate (default)***
433 MHz	1 W	8 km	9600	2.4 Kbps

Tabela 2 Características do módulo E32-DTU (433L30)

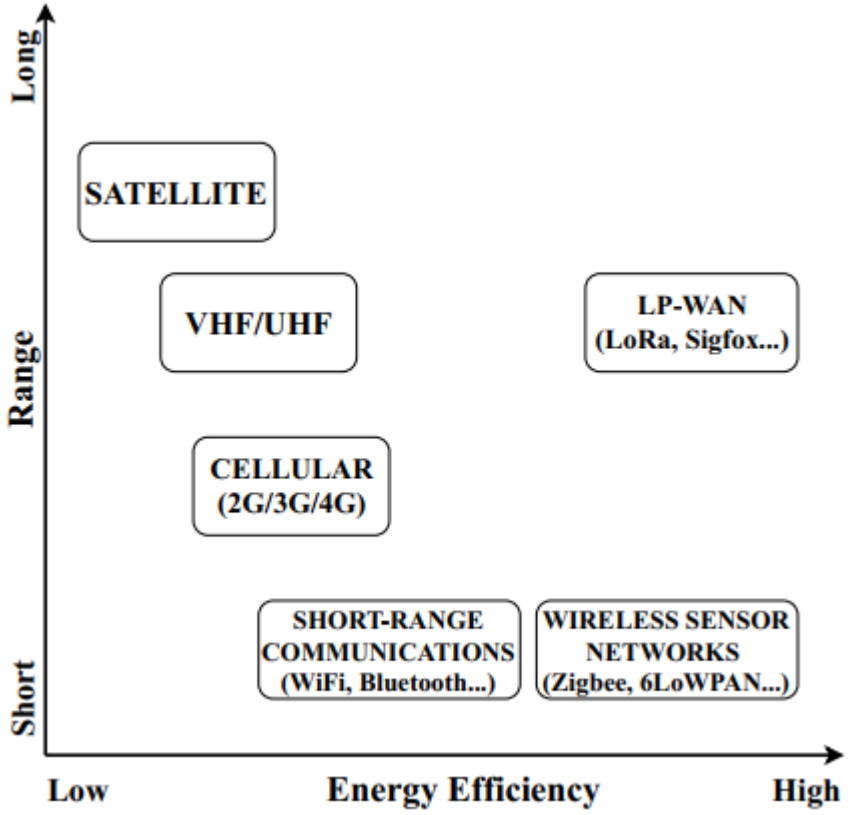

Figura 32 Comparação entre alcance e eficiência energética fornecida por diferentes tecnologias de comunicação empregadas em cenários marítimos (Sanchez-Iborra, Liaño, Simoes, Couñago, & Skarmeta, 2019)

[8] Circuíto integrado que converte os níveis de tensão de sinal RS-232 para TTL lógica («MAX232 Data Sheet», 2010).

2.7 Bateria

Por forma a maximizar a permanência do veleiro nos testes de mar, substituiu-se a bateria utilizada por uma de maior capacidade. A nova bateria é uma bateria LiPo 6s de 10Ah. Tendo como referência o consumo energético calculado por Fernandes (Fernandes, 2016), estimou-se um consumo médio de 3,40W. Para esta nova arquitetura do sistema, e, observando a Tabela 3, obtemos um consumo de 4,86W. Apesar do consumo energético do sistema ter aumentado, foi utilizada uma bateria com maior capacidade, que a 12V permite ter uma autonomia de aproximadamente de 24 horas, duas vezes e meia superior à primeira bateria utilizada. É ainda de referir que o rendimento dos conversores *step down* DC-DC foi considerado 100% no cálculo.

Item	Corrente (mA)	Tensão (V)	*not idle*	Potencia *idle* (W)
Anemómetro	41	5	100%	0.205
GPS	35	5	100%	0.175
Módulo de comunicações	100	12	90%	1.08
IMU	22	5	100%	0.11
Arduíno *master*	65	12	100%	0.78
Servo leme	285	5	50%	0.7125
Motores de controlo mastros (x2)	200	12	50%	1.2
Arduíno *slave*	50	12	100%	0.6
			TOTAL	**4.8625**

Tabela 3 Cálculo de consumo energético

3 Comando e controlo

Este capítulo apresenta os algoritmos desenvolvidos no âmbito desta dissertação. Todo o código foi desenvolvido na plataforma Arduíno IDE (*Integrated Development Environment*) em C++. O Arduíno dos sensores, que se designou como o *master*, compila todos os dados dos sensores e calcula a proa a seguir, usando trigonometria esférica (Fernandes, 2016). Paralelamente é verificada a distância ao *waypoint*; se for inferior ao valor padrão, o algoritmo calcula a proa para o *waypoint* seguinte. Este é também o responsável por decidir o ângulo de ataque das velas, como veremos adiante, o código deste Arduíno está disponível do Apêndice A ao Apêndice F.

O Arduíno controlador das velas, denominado *slave*, lê o valor dos potenciómetros e atua diretamente no módulo *Veyron*. É possível consultar o código no Apêndice G.

3.1 Algoritmo Arduíno dos sensores

3.1.1 Integração protocolo NMEA

A *National Marine Electronics Association* (NMEA) é uma associação sem fins lucrativos de fabricantes, distribuidores, revendedores, instituições de ensino e outros interessados em periféricos com aplicação em eletrónica marítima. O padrão NMEA 0183 define uma interface elétrica e um protocolo de dados para comunicações entre instrumentação marítima. Todos os dados são transmitidos na forma de frases. Apenas caracteres ASCII são permitidos, além de CR (*carriage return*) e LF (*line feed*). Cada frase começa com um sinal de "$" e termina com <CR> <LF> (Betke, 2000).

A título de exemplo, na Figura 33, é mostrado um excerto do código NMEA que foi aplicado a diferentes variáveis tais como: vento relativo (direção e velocidade), rumo, velocidade verdadeira, proa, *pitch*, *roll*, hora, posição e cobertura GPS.

Neste caso é intenção gerar uma frase NMEA do tipo *$RSA,40,A,,*05*, que define a posição angular do leme. O código começa por criar um *array* de 23 caracteres, que foi definido para o máximo de caracteres que a frase NMEA poderia ter. Em seguida é declarada a varável *cs_ruder* que irá ser usada no *checksum*. A biblioteca *PString* permite concatenar a frase com partes que vão sendo geradas sucessivamente. É então gerado a *string* $RSA, que é o código padrão para a posição angular do leme, seguido do valor da variável calculada na função *Rudder.cpp*, do caractere "A" que dá como válido o valor da varável. Existe a possibilidade de declarar a posição de dois lemes mas como não é aplicável para este protótipo é deixado um caractere vazio entre vírgulas. Por fim a frase termina com um asterisco que indica o início do *checksum*. A função *checksum* soma os valores decimais de todos os caracteres da frase inclusive o valor do asterisco e converte para formato hexadecimal. Por fim, a frase é enviada para a porta série 2, que está ligada ao módulo de comunicações.

```
//RUDDER NMEA ex: $RSA,40,A,,*05
    char NMEA_rudder [23];
    byte cs_rudder; //checksum compass
    PString strm3(NMEA_rudder, sizeof(NMEA_rudder));
    strm3.print("$RSA,");
    strm3.print(rudder.NMEA_rudder);
    strm3.print(",A,,*");
    cs_rudder = checksum(NMEA_rudder);
    if (cs_rudder < 0x10) strm3.print('0');
    strm3.print(cs_rudder, HEX);
    Serial2.println(NMEA_rudder);
```

Figura 33 Exemplo excerto código NMEA para posição angular do leme

```
byte checksum(char* str) //Checksum function
{
  byte cs = 0;
  for (unsigned int n = 1; n < strlen(str) - 1; n++)
  {
   cs ^= str[n];
  }
  return cs;
}
```

Figura 34 Função *checksum* NMEA

3.1.2 Direção do vento

O ficheiro *Wind.cpp* começa por ler os valores PWM diretos do *encoder* montado no anemómetro. Seguidamente procede-se à normalização dos valores de acordo com a Equação 3, sendo respetivamente a e b, o valor máximo e mínimo que se quer normalizar, max, o valor máximo PWM lido pelo Arduíno, min, o valor mínimo lido pelo Arduíno e x, o valor a normalizar. É de referir que o valor máximos e mínimos de PWM dependem da intensidade do campo magnético gerado pelo íman. Ao efetuar a média aritmética com ângulos a oscilar entre valores não consecutivos, por exemplo, 355 e 10, resultaria numa média de 182,5, que não se pretende. Torna-se necessário efetuar uma mudança de coordenadas, por forma a obter uma função contínua, convertendo coordenadas polares em cartesianas. Após conversão de coordenadas é efetuada a média para um *array* com um número de posições definidas pelo utilizador (predefinido a 50 valores), por forma a colmatar a perda de sinal registada pelo comprimento excessivo do cabo do anemómetro. Por fim a média é reconvertida em coordenadas polares, sendo *output* para a função *main.ino*. Na Figura 35 podemos consultar o fluxograma do ficheiro *Wind.cpp*.

$$f(x) = \frac{(b-a)(x-\min)}{max-min} + a$$

Equação 3 Normalização valores PWM

Figura 35 Fluxograma do código da direção do vento do ficheiro *Wind.cpp*

3.1.3 Velocidade do vento

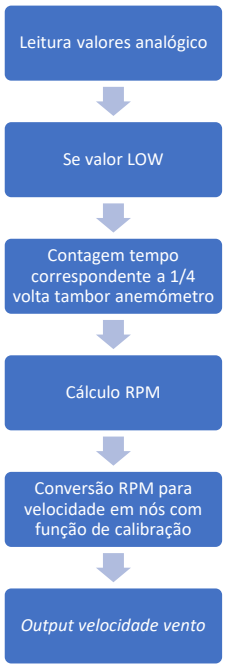

Figura 36 Fluxograma do código da velocidade do vento do ficheiro *Wind.cpp*

Ainda no ficheiro *Wind.cpp* após efetuado o cálculo da direção do vento, é calculada a intensidade. O código começa por ler o valor analógico do sensor de *hall*. Se este valor for abaixo de 1,5V, é assumido com *LOW*, ou seja, está a passar pelo sensor um íman com o polo sul perto deste. O Arduíno começa a contar o tempo e como os polos dos quatro imanes estão trocados sequencialmente, ao passar perto do sensor o íman seguinte (de polo Norte), o Arduíno assume que o valor analógico passou para cima de 1,5V (*HIGH*). É então calculado o tempo desta mudança de estado que corresponde a ¼ de volta. Com este valor é possível calcular o número de rotações por minuto. Com recurso à função de calibração calculada no capítulo 2.3.1, o valor é convertido em velocidade (nós).

3.1.4 Velas

Por terem sido utilizadas velas diferentes, foi necessário ajustar o algoritmo das velas. Tendo as velas um valor máximo de *lift* a 15º ao vento (discutido no capítulo 1.1), foi criada uma condição que define que para valores entre 0º e 135º, a vela faz uso do efeito

de *lift* e assume 15º a menos que o vento. Para valores entre 135º e 225º, a vela comporta-se fazendo uso do efeito de *drag*. Acima de 225º até 360º, assume 15º a mais que o vento.

Segundo (Rynne & Von Ellenrieder, 2010), e utilizando um *airfoil* semelhante e desprezando o facto das velas construídas afunilarem em altura, é mais eficiente usar o efeito de *drag* a partir dos 135º (até 225º), como mostra a Figura 37.

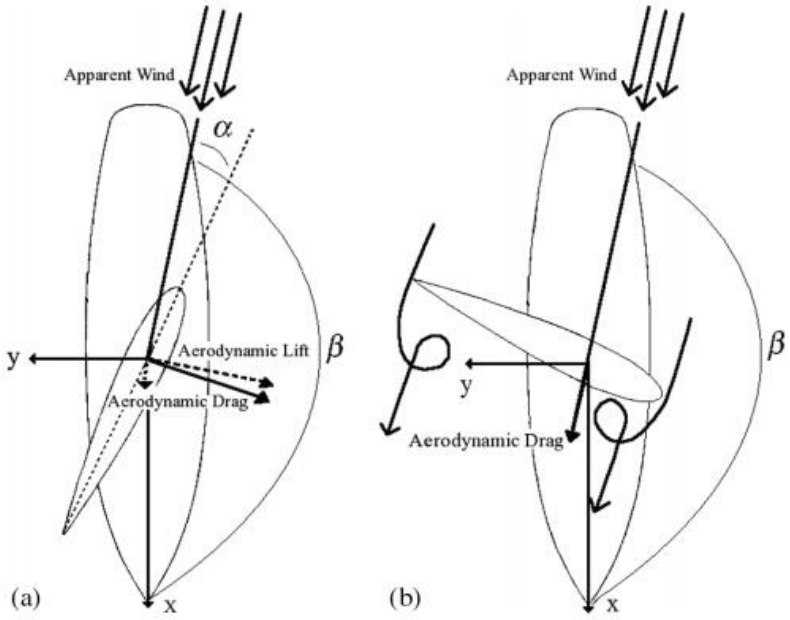

Figura 37 A navegar a favor do vento a vela rígida pode ser mareada (a) produzindo *lift* com um baixo ângulo de ataque ou (b) produzindo *drag* com um grande ângulo de ataque (Rynne & Von Ellenrieder, 2010)

```
if (autonomous == true){
  if (WDirection > 180 && WDirection < 225)
    angle = WDirection + 90;
  if (WDirection > 135 && WDirection < 180)
    angle = WDirection - 90;
  if (WDirection > 0 && WDirection < 135)
    angle = WDirection - 15;
  if (WDirection > 225 && WDirection < 360)
    angle = WDirection + 15;
}
```

Figura 38 Excerto do código de controlo das velas em função da direção do vento

3.2 Algoritmo Arduíno controlador das velas

Durante a programação do Arduíno das velas, é carregado o ficheiro *Codigo_velas_V2.ino*. Este ficheiro começa por incluir as bibliotecas do módulo controlador de motores DC *Veyron*, através da função *#include.* São também definidos valores máximos e mínimos para os potenciómetros das velas para evitar que estas cheguem ao limite mecânico dos potenciómetros, como veremos adiante.

Seguidamente são declaradas utilizadas dentro *void loop()* e depois são abertas as portas série referentes à comunicação com o Arduino principal e com o módulo *Veyron*. O *void loop()* começa por ler os valores analógicos dos potenciómetros e verifica através de uma condição *if* se estes estão dentro dos valores definidos anteriormente, Figura 39.

```
void loop()
{
  sail_read1 = analogRead(A6);
  sail_read2 = analogRead(A5);
  if (sail_read1 > VALOR_MAX_POTENCIOMETRO_VELA || sail_read1 <
VALOR_MIN_POTENCIOMETRO_VELA || sail_read2 > VALOR_MAX_POTENCIOMETRO_VELA ||
sail_read2 < VALOR_MIN_POTENCIOMETRO_VELA)
  safe();
  Serial.print("sail 1: "); Serial.println(analogRead(A6));
  Serial.print("sail 2: "); Serial.println(analogRead(A5));
```

Figura 39 *Void loop*

Se a condição for falsa, o código é direcionado para a função *safe()*, que imprime na porta série de ligação com o computador: *"erro a vela saiu do sitio"* seguido dos valores analógicos dos potenciómetros, Figura 40.

```
void safe(){
  VeyronBrushed.setSpeed(id, 1, 0);
  VeyronBrushed.setSpeed(id, 2, 0);
       while(true) {
        Serial.println("erro a vela saiu do sitio");
        Serial.print("sail 1: "); Serial.println(analogRead(A6));
        Serial.print("sail 2: "); Serial.println(analogRead(A5));
        delay(500);
        }
       }
```

Figura 40 *Void safe*

Se a condição *if* for verdadeira, o código entra num ciclo *while* que verifica se o utilizador está a inserir manualmente ângulos no monitor de série e passa diretamente para a

função *normal* atuando diretamente nas velas. Se o computador não estiver ligado por cabo ao Arduíno, o utilizador deixa de ter controlo direto nas velas, passando o controlo para o Arduíno dos sensores (onde está ligado o anemómetro) estando neste momento este algoritmo dependente do valor da direção do vento.

```
while (Serial.available()) {
        aux = Serial.parseInt();
        Serial.print("novo angulo: "); Serial.println(aux);
        if(aux > 5 && aux < 355){
        angle = aux;
        Normal = true;
        }
         else{
           if(aux == 666){
             Normal = false;
           }
           else{
         Serial.print("not a valid angle! Try again mate. Value: "); Serial.println(aux);
         delay(1000);
         }
      }
  }
  if (Normal == true){
        normal();
        }
        else{
        borboleta();
        }
  }
```

Figura 41 Introdução manual de ângulos das velas

Em situação autónoma o Arduíno dos sensores diz ao Arduíno controlador das velas qual é o ângulo que é pretendido para as velas e que depende diretamente do vento (ver capítulo 3.1.4). A função *normal*, Figura 42, começa por verificar se o ângulo se encontra

dentro dos valores de ângulo de vela admissíveis (20º a 340º), e, se a condição for verdadeira, o algoritmo aloca na varável *speed*, um valor inteiro que corresponde ao valor de um parâmetro de calibração, multiplicado pelo valor do angulo pretendido e somando outro valor de calibração, ou seja, a função de posicionamento do mastro é uma equação do tipo $y = mx + b$, em que y é o angulo da vela, m é um valor de correção que depende da desmultiplicação mecânica das rodas dentadas para o potenciómetro e b um valor de correção, que permite calibrar a posição zero do mastro. Por fim se o ângulo de vela pretendido for maior ou menor que o simétrico de um erro definido pelo utilizador (*sail_error*), as velas posicionam-se de maneira a cumprir com esta condição. Existe uma função de normalização, *speed_normalization* que impede que o motor rode demasiado devagar sem surtir qualquer efeito o que poderia causar o seu sobre aquecimento. Esta função garante que o *speed* nunca baixa de 110. É possível ainda ajustar o *speed* multiplicando-o por um fator de correção, ajustado depois de montadas as velas e bucim dos mastros (ver capítulo 1.3) caso exista muita prisão mecânica, sem modificar a função *speed_normalization*.

```
void normal(){
    if(angle > 20 && angle < 340){
        speed = - (0.8633*angle+380 - sail_read1); //0.8633*angle+330.2 - sail_read1
        Serial.print("speeed :    "); Serial.println(speed);
    if ( speed > sail_error ||  speed < -sail_error){
       speed_normalization();
       speed = speed*1.1;
       VeyronBrushed.setSpeed(id, 1, speed);
          }
        else{
        VeyronBrushed.setSpeed(id, 1, 0);
        }

         speed = - (0.8578*angle+520 - sail_read2);   //Update sail 2 //0.8578*angle+307 -
sail_read2
     if ( speed > sail_error ||  speed < -sail_error){
       speed_normalization();
       speed = speed*1.0;
       VeyronBrushed.setSpeed(id, 2, speed);
       }
        else{
        VeyronBrushed.setSpeed(id, 2, 0);
        }

  }
}
```

Figura 42 Função *normal*

4 Resultados e discussão

Neste capítulo são apresentados os resultados obtidos nas melhorias implementadas no veleiro e nas diversas provas de mar efetuadas ao mesmo.

Pelo facto de se ter utilizado o protocolo NMEA 0183 extensivamente nas comunicações para o exterior, foi possível enviar dados possíveis de serem lidos por software *open-source* como é o caso do *OpenCPN*. Este software permite visualizar em tempo real o que é transmitido pelo veleiro e ainda ver e guardar todo o fluxo NMEA recebido. Pela Figura 43, é possível visualizar graficamente a posição GPS, o rumo, velocidade verdadeira, direção e velocidade do vento aparente, ângulo do leme, *pitch*, *roll* e proa. No futuro poderá ser de interesse integrar mais sensores também aceites por este programa como sonda, barómetro, odómetro, radares ou termómetro atmosférico.

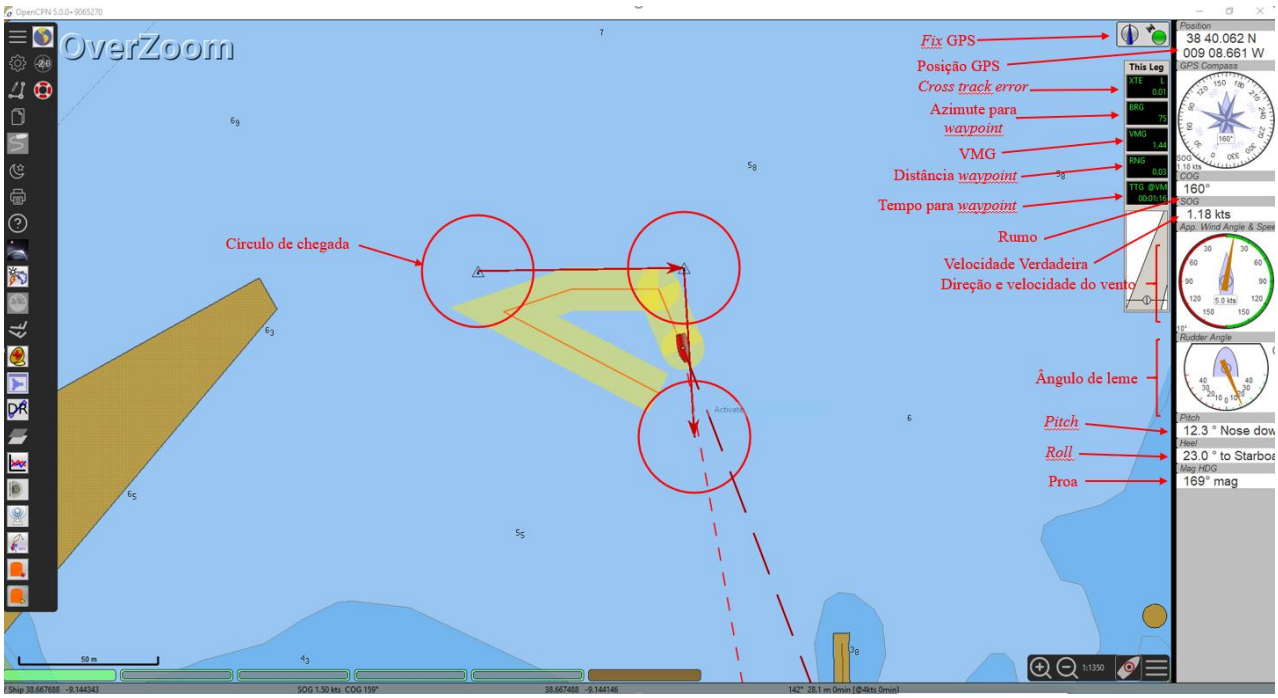

Figura 43 Visualização em tempo real no *OpenCPN*

Este *software* permite ainda instalar diversos *plug-ins*, no âmbito da *strategic long term routeing* como *wheather routing* tendo por base gráficos polares de performance do veleiro, vento *in situ* e previsão meteorológica. Permite ainda fazer anticolisão através da informação recebida por AIS ou radar.

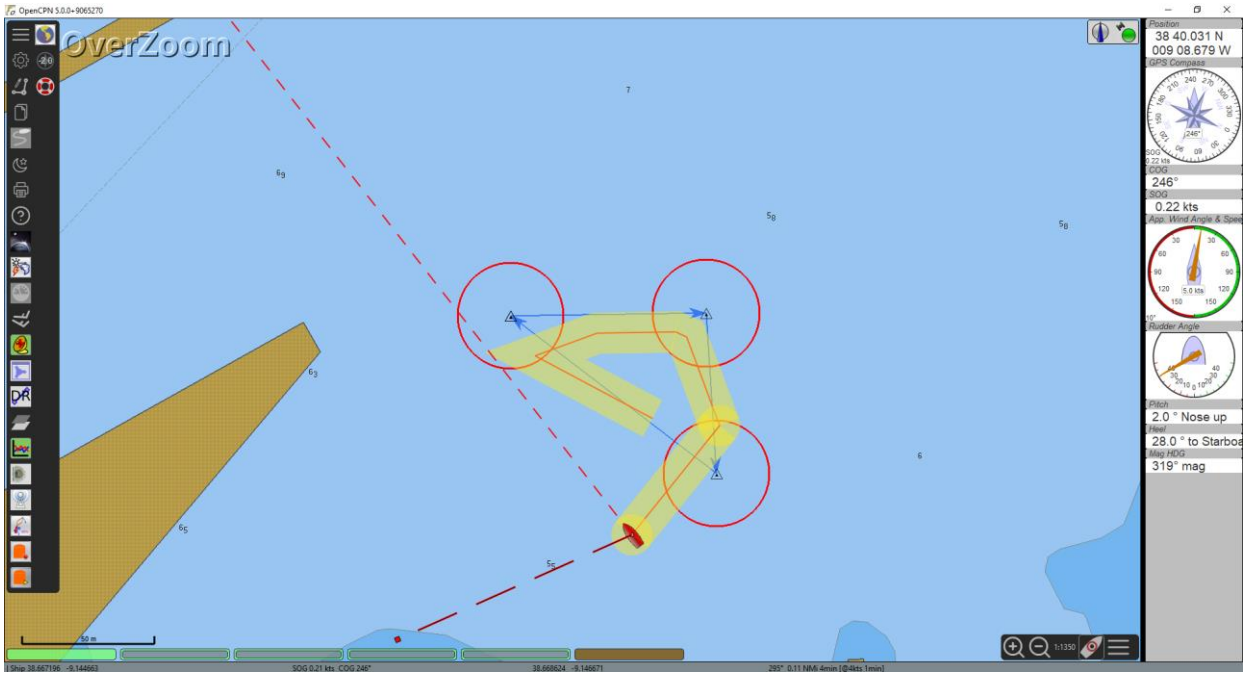

Figura 44 *Tracking* executado pelo Barlavento nas provas de mar

Durante as provas de mar, o anemómetro sofreu uma avaria e o sistema bloqueou num *loop* infinito. Mesmo assim, foram retomados os testes nos quais foi bloqueada a direção do vento para um valor médio das condições no local, ou seja, o veleiro passou a assumir o vento aparente como vento verdadeiro. Apesar da avaria, o veleiro comportou-se de forma exemplar, conseguindo marear as suas velas, descrevendo novamente o percurso entre os três *waypoints* definidos. É verificável também que este assume o próximo *waypoint* quando entra no raio de distância mínima (círculo de chegada) definido pelo utilizador que por definição é de 20 m.

É ainda apreciável o facto deste sistema de navegação ter sido reconvertido e utilizado em diversos sistemas não tripulados desenvolvidos na Célula de Experimentação Operacional de Veículos (CEOV), provando o funcionamento deste algoritmo.

Figura 45 Imagem aérea fotografada por *drone* nos primeiros testes à plataforma (ainda com as velas antigas)

Figura 46 Barlavento a navegar à bolina com as suas novas velas

No cômputo geral, os resultados dos testes revelaram-se bastante positivos, apesar das avarias que são muito comuns na prototipagem de sistemas e que servem de motivação

para colmatar pontos negativos. Os principais progressos no desenvolvimento do veleiro autónomo Barlavento podem-se evidenciar nos seguintes pontos:

1. O potenciómetro multivolta utilizado revelou-se bastante mais preciso que o anterior, permitindo um melhor controlo da posição das velas e um intervalo maior de voltas das velas sem que estas chegassem ao limite mecânico deste.
2. O CPE+ revelou-se ser muito mais resistente à ação do calor e ultravioletas que o ABS, que como consequência fez deformar a porta de visita mais pequena, construída com o último material. Comparativamente com a versão anterior do veleiro, o anemómetro construído permite receber a velocidade do vento e revela valores muito mais estáveis e precisos, mesmo numa posição perto da linha de água. Contudo, necessita de ser reformulada a sua estanqueidade, utilizando mais resina *epoxy* na proteção dos sensores.
3. As comunicações com a estação em terra sofreram melhorias significativas, tendo produzido alcances muito maiores sem nunca perder pacotes.
4. A IMU revelou-se mais precisa que o modelo anteriormente utilizado, mas ao contrário da última precisa de ser calibrada sempre que o sistema é iniciado.
5. Por ter sido usado um protocolo padrão (NMEA 0183) de troca de dados é possível utilizar diferentes tipos de plataformas que aceitem este protocolo

5 Trabalho futuro

Sendo o Barlavento um protótipo experimental, há sempre diversos pontos que podem ser melhorados em projetos futuros, por forma a aumentar a performance do veleiro. Começando pelo casco, é claramente visível que o seu *design* não é o melhor. Dever-se-á realizar-se um estudo hidrodinâmico, possibilitando o veleiro atingir velocidades maiores, melhorando assim o seu rendimento. É de notar que a velocidade do veleiro depois dos melhoramentos introduzidos que resultaram desta tese, nunca ultrapassou os dois nós, que são valores semelhantes à versão anterior.

As velas podem também sofrer alterações, introduzindo o conceito de *morphing wings* (Navaratne, Dayyani, Woods, & Friswell, 2015) com recurso à manufatura aditiva. Umas velas que conseguem alterar o seu *camber* têm um maior rendimento, apesar da adição de mais atuadores, o que implica um maior consumo de energia. Um estudo aerodinâmico da *performance* das velas é também essencial.

Um sistema de produção de energia é também fulcral a uma plataforma que seja projetada para ter continuidade no tempo. É também essencial ter um controlo na GCS da bateria disponível no sistema. Através da introdução de painéis solares ou uma turbina juntamente com um sistema de controlo de carga da bateria, BMS (*Battery Management System*), é possível coletar e armazenar eficientemente energia do meio, além de informar a GCS sobre a carga da bateria e consumos; não obstante de todas as vantagens a nível ambiental inerentes a um sistema deste tipo.

Em relação à arquitetura do sistema, propõe-se a utilização de microcontroladores com maior capacidade de processamento ou a utilização de um microprocessador em paralelo que permitisse desempenhar tarefas mais complexas como *wheather routing* ou fazer análise de dados dos sensores de bordo, enquanto que as funções básicas de navegação (*short course routeing*) continuavam a ser desempenhadas por microcontroladores. A baixa fiabilidade do sistema pode ainda ser colmatada com o desenvolvimento de PCB's mais

compactos, diminuindo a quantidade de cabos elétricos e consequentemente a probabilidade de estes se deligarem durante a navegação. Depois de atingir a fiabilidade necessária à navegação, poder-se-á adicionar mais *payload* de sensores por forma a recolher dados e executar missões.

Sendo o *OpenCPN* uma plataforma de código aberto e reprogramável, pode servir de ponto de partida futuro para uma plataforma inteiramente dedicada a veleiros autónomos, que poderá ser instalada diretamente num sistema como uma *raspberry pi* e simultaneamente como GCS, utilizando também todas as suas ferramentas de *plug-ins*.

Deverão ser efetuados mais testes para otimizar o código interno do veleiro. A recolha de dados da velocidade em função de diferentes proas e regimes de intensidade e direção do vento, torna-se essencial na construção de um gráfico polar para otimizar o controlo das velas e a variável VMG. Ainda no intuito de melhorar o código interno, deverão ser introduzidas mais funções de segurança e redundância no caso de alguns dos componentes falharem. Tendo o veleiro duas velas, é possível navegar sem utilizar o leme (no caso deste falhar), utilizando o binário entre as duas. Algoritmos de anticolisão com informação AIS podem vir a ser muito proveitosos (Jaulin & Bars, 2012), (Alves & Cruz, 2016).

Seria também muito útil no futuro ter uma GCS inteiramente dedicada a veleiros autónomos que pudesse integrar vários veículos, permitindo estruturar e planear missões bem como ter um total comando e controlo da navegação.

Depois de garantir a fiabilidade e robustez necessária para navegação em mar abeto torna-se necessário implementar sistemas de comunicações de longo alcance quer para controlo direto do veleiro quer para envio de séries temporais de dados. Um sistema de comunicações satélite é essencial para plataformas deste tipo.

A adição de outros tipos de sensores como por exemplo um SDR permitiria efetuar escuta passiva das emissões eletromagnéticas no ar e ainda receber informação meteorológica ou alvos AIS. Sensores acústicos passivos podem também ser implementados (no bolbo no patilhão), sendo esta uma área de enorme interesse para a Marinha.

Conclusão

Esta dissertação inicia-se começando por resolver problemas estruturais da primeira versão do veleiro Barlavento, dando início à construção de novas velas, portas de visita, e bucim estanque dos mastros, com recurso a ferramentas CAD. Em seguida, foi reformulada a arquitetura do sistema, integrando novos sensores, comunicações, bateria e construído e calibrado um sensor de vento. Em seguida, é explicada toda a parte de comando e controlo, explicitando mais detalhadamente toda a programação desenvolvida, começando pela integração do protocolo NMEA nas comunicações, o cálculo da direção e velocidade do vento bem como o algoritmo de controlo das velas. Por fim, são mostrados e discutidos os resultados obtidos nas provas de mar.

No final da dissertação, o veleiro Barlavento revelou-se como uma plataforma fiável e robusta e de baixo custo, podendo ser a base de trabalhos futuros em linhas de investigação muito variadas. A prototipagem desta plataforma resultou também numa experiência prática de utilização extensiva de impressão 3D e CNC para fabricar componentes complexos.

A prototipagem de sistemas autónomos é uma tarefa que engloba muitas áreas diferentes do conhecimento. É, por conseguinte, uma tarefa altamente complexa que exige a integração constante de várias disciplinas, e que por vezes requerem uma curva de aprendizagem muito acentuada.

Bibliografia | Referências

Aartrijk, M. Van, Tagliola, C., & Adriaans, P. (2002). AI on the Ocean: the RoboSail Project. *ECAI*, 653–657. Obtido de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.8172&rep=rep1&type=pdf%0Ahttp://frontiersinai.com/ecai/ecai2002/pdf/p0653.pdf

Alves, J. C., & Cruz, N. A. (2008). FASt - An autonomous sailing platform for oceanographic missions. Em *OCEANS 2008*. https://doi.org/10.1109/OCEANS.2008.5152114

Alves, J. C., & Cruz, N. A. (2016). AIS-Enabled Collision Avoidance Strategies for Autonomous Sailboats. Em *Robotic Sailing 2015*. https://doi.org/10.1007/978-3-319-23335-2_6

Austria Mikro Systeme. (2016). AS5048A/AS5048B datasheet. Obtido de https://www.mouser.com/ds/2/588/AS5048_DS000298_3-00-263460.pdf

Betke, K. (2000). The NMEA 0183 Protocol. Obtido de http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf

Bowditch, N. (2002). *The New American Practical Navigator. Deffense Mapping Agency Hydrographic/Topographic*.

Carlos Manuel Moreira Alves. (2010). Sailbot: Autonomous Marine Robot of Eolic Propulsion, (December), 1–121.

Cruz, N. A., & Alves, J. C. (2010). Auto-heading controller for an autonomous sailboat. *OCEANS'10 IEEE Sydney, OCEANSSYD 2010*. https://doi.org/10.1109/OCEANSSYD.2010.5603882

dos Santos, D. H., & Goncalves, L. M. G. (2019). A gain-scheduling control strategy and short-term path optimization with genetic algorithm for autonomous navigation of a sailboat robot. *International Journal of Advanced Robotic Systems*. https://doi.org/10.1177/1729881418821830

Fernandes, P. M. de C. (2016). *Projeto e Construção de um Veleiro Autónomo, utilizando materiais compósitos, Impressão a 3D e Aprendizagem Máquina*. Escola Naval. Obtido de http://hdl.handle.net/10400.26/15068

Infineon technologies. (2017). Uni- and Bipolar Hall IC Switches for Magnetic Field Applications. Obtido de http://www.produktinfo.conrad.com/datenblaetter/150000-174999/153775-da-01-en-HALL_SENSOR_TLE_4935L.pdf

Jaulin, L., & Bars, F. Le. (2012). A simple controller for line following of sailboats. *5th International Robotic Sailing Conference*, 107–119.

Marques, P. (2015). Monitorização e gestão de energia no veleiro autónomo FASt. Obtido de http://scholar.google.comhttps//comum.rcaap.pt/handle/10400.26/11243

MAX232 Data Sheet. (2010). Em *SD Card Projects Using the PIC Microcontroller*. https://doi.org/10.1016/b978-1-85617-719-1.00015-4

Meinig, C., Lawrence-Slavas, N., Jenkins, R., & Tabisola, H. M. (2015). The use of

Saildrones to examine spring conditions in the Bering Sea: Vehicle specification and mission performance. Em *OCEANS 2015 - MTS/IEEE Washington*. https://doi.org/10.23919/OCEANS.2015.7404348

Mordy, C., Cokelet, E., De Robertis, A., Jenkins, R., Kuhn, C., Lawrence-Slavas, N., … Wangen, I. (2017). Advances in Ecosystem Research: Saildrone Surveys of Oceanography, Fish, and Marine Mammals in the Bering Sea. *Oceanography*. https://doi.org/10.5670/oceanog.2017.230

Navaratne, R., Dayyani, I., Woods, B. K. S., & Friswell, M. I. (2015). Development and Testing of a Corrugated Skin for a Camber Morphing Aerofoil, (January), 1–10.

Ruzicka, T. (2017). *Model based Design of a Sailboat Autopilot*. Halmstad University.

Rynne, P. F., & Von Ellenrieder, K. D. (2010). Development and preliminary experimental validation of a wind- and solar-powered autonomous surface vehicle. *IEEE Journal of Oceanic Engineering*, *35*(4), 971–983. https://doi.org/10.1109/JOE.2010.2078311

Sanchez-Iborra, R., Liaño, I. G., Simoes, C., Couñago, E., & Skarmeta, A. F. (2019). Tracking and Monitoring System Based on LoRa Technology for Lightweight Boats. *Electronics*, *8*(1), 15. https://doi.org/10.3390/electronics8010015

Seneviratne, P. (2019). *Beginning LoRa Radio Networks with Arduino*. *Beginning LoRa Radio Networks with Arduino*. https://doi.org/10.1007/978-1-4842-4357-2

Sensortec, B. (2014). Intelligent 9-axis absolute orientation sensor, (November).

Stelzer, R., & Jafarmadar, K. (2011a). History and Recent Developments in Robotic Sailing. Em *Robotic Sailing*. https://doi.org/10.1007/978-3-642-22836-0_1

Stelzer, R., & Jafarmadar, K. (2011b). History and Recent Developments in Robotic Sailing. Em *Robotic Sailing* (pp. 3–23). https://doi.org/10.1007/978-3-319-10076-0

Stelzer, R., Le, G. B. L., & Jafarmadar, K. (1998). A Layered System Architecture to Control an Autonomous Sailboat. *Sensors And Actuators*.

Stelzer, R., Proll, T., & John, R. I. (2007). Fuzzy Logic Control System for Autonomous Sailboats. *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, *00*, 1–6.

Tretow, C. (2017). Design of a free-rotating wing sail for an autonomous sailboat autonomous sailboat.

Ultimaker. (sem data). Technical data sheet CPE+.

Apêndice A

```
#include <Adafruit_BNO055.h>
#include <math.h>
#include <Servo.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <Servo.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <PString.h>

#include "Main_GPS.cpp"
GPS gps;
//#include "compass.cpp"
//Compass compass;
#include "WayPoints.cpp"
WP wp;
#include "Wind.cpp"
Wind wind;
#include "Rudder.cpp"
Rudder rudder;
#include "Sail.cpp"
Sail sail;
```

```
#define RUDDER_PIN 11 // the rudder is connected to pint 12

#define WIND_DIR_PIN A7

#define WIND_SPEED_PIN 2


#define BNO055_SAMPLERATE_DELAY_MS (100)

Adafruit_BNO055 bno = Adafruit_BNO055(55,0x29);



//*****************************************

float DistCalc = 0; // float dist2wp = 0; // dist2wp is the distance (in meters) to the next Waypoint2

float AziCalc = 270;

float target_head = 180;  // target_head is the aymuth that we should follow to reach the WP (which may be different

// from azi2wp because we may have to tack.

float head = 0;

float roll = 0;

float pitch = 0;


bool autonomous = true;

bool manual = false;

bool sail_stop = false;

bool sail_auto = true;

bool sail_upwind = false;

long double millisBS = 0;

long double millis_sail = 0;
```

```
String Comand = "autonomous";
int wp_number = 0;
int wp_min_dist = 20;

int AziManual = 180;
int sailManual = 180;
int sail_teste_value = 135;

File myFile;
Servo servo_rudder;

//*****************************************
void setup() {

  Serial.begin(9600); // connect serial
  Serial2.println("Sistema a iniciar");
  Serial1.begin(9600); // connect gps sensor
  Serial2.begin(9600); // connect Comunications
  Serial3.begin(57600); // mast control arduino
  !bno.begin();
  delay(500);

  pinMode(22, OUTPUT); // PIN to enable wind sensor
  digitalWrite(22, HIGH);
  pinMode(50, OUTPUT); // PIN to enable wind sensor
  digitalWrite(50, HIGH);
  pinMode(WIND_DIR_PIN, INPUT);
  pinMode(WIND_SPEED_PIN, INPUT);
```

```
wp.Update();
  servo_rudder.attach(RUDDER_PIN);    // the rudder is connected to pin 12
  int rudder_center_value = 98;
  servo_rudder.write(98);     // initilise the rudder to the center position (which we estimate
will be 98 degrees)

  if (!SD.begin(8)) {
    Serial2.println("Módulo cartão SD não detetado");
    return;
  }
  else {
    Serial2.println("Creating LOG File.");
    myFile = SD.open("LOG.txt", FILE_WRITE);
  }

  if (!bno.begin()) {
    Serial2.println("Bussula I2C não detetada! ");
    return;
  }
  else {
    Serial2.println("Bussula I2C OK");
  }
    /* Use external crystal for better accuracy */
  bno.setExtCrystalUse(true); //Compass

  /* Display some basic information on this sensor */
  displaySensorDetails(); //Compass

}
```

```
//****************************************

void loop() {

  ReadComand();

  head_I2C_Update();

  wind.Update(pulseIn(WIND_DIR_PIN, HIGH), analogRead(2), micros()); //

  gps.Update();

  if (gps.GPS_LOCK) {

    DistCalc = gps.CalcDist(gps.lat, gps.lon, (wp.wplat[wp_number]), (wp.wplon[wp_number]));
// Calcula a distancia entre os Wp's e a posição atual

    AziCalc = gps.CalcAzi(gps.lat, gps.lon, (wp.wplat[wp_number]), (wp.wplon[wp_number]));  //
Calcula o Azimute entre os Wp's e a posição atual

    wp_test();     // passa ao proximo waypoint se estiver dentro do raio wp_min_dist

  }


  Update_GCS(350); //250

  sail_update(3000);

  if (autonomous == true) {

    servo_rudder.write(rudder.Update(head, AziCalc));

    }

  else {

    servo_rudder.write(rudder.Update(head, 130));

  }


  // Send to function the update rate in millis wanted

}
```

```
void ReadComand() {
  if (Serial.available()) {
    Comand = Serial.readString();

    if (Comand == "sail_stop") {
      sail_stop == true;
    }
    if (Comand == "sail_go") {
      sail_stop == false;
    }
    if (Comand == "w") {
      wp_number++;
    }

    if (Comand == "sail_upwind") {
      sail_upwind == false;
    }
    if (Comand == "go_upwind") {
      sail_upwind == true;
    }

    if (Comand == "TestGPS") {
      Serial.println("Testing GPS");
      gps.Test();
    }
    if (Comand == "Manual") {
      autonomous = false;
      Serial.println("Manual mode engaged");
    }
    if (Comand == "Autonomous") {
      autonomous = true;
```

```
Serial.println("Autonomous mode engaged");
    }
  }
  if (( Comand.substring(0, 1) == "s") && (autonomous == false)) {
    sailManual = Comand.substring(1, 4).toInt();
  }
  if (( Comand.substring(0, 1) == "h") && (autonomous == false)) {
    AziManual = Comand.substring(1, 4).toInt();
  }
  if (Comand.substring(0, 3) == "stv")
    sail_teste_value = Comand.substring(2, 6).toInt();
}

void sail_update(int update_time){
   if (millis() - millis_sail > 3000) {
    millis_sail = millis();
    sail.Update(wind.Direction,1);
   }
}

void Update_GCS(int update_time) {
  if (millis() - millisBS > update_time) {
    millisBS = millis();
```

```
//_____
____//

//COMPASS NMEA

    char NMEA_compass [23];

    byte csm; //checksum compass

    PString strm(NMEA_compass, sizeof(NMEA_compass));

    strm.print("$HDM,");

    strm.print(lround(head));  // lround simply rounds out the decimal, since a single degree is
fine enough of a resolution

    strm.print(",M*");

    csm = checksum(NMEA_compass);

    if (csm < 0x10) strm.print('0');

    strm.print(csm, HEX);

    Serial2.println(NMEA_compass);

//_____
____//

//GPS NMEA

    char NMEA_gps_GPRMC [100];

    byte cs_GPRMC; //checksum compass

    PString strm1(NMEA_gps_GPRMC, sizeof(NMEA_gps_GPRMC));

    strm1.print("$GPRMC,");

    strm1.print((String(gps.time_GPS)).substring((0,5)));

    strm1.print(",A,");

    strm1.print(gps.LAT_NMEA,6);

    strm1.print(",N,");

    strm1.print(gps.LON_NMEA,6);

    strm1.print(",W,");

    strm1.print(gps.Speed_kts);

    strm1.print(",");

    strm1.print(gps.GPS_head);

    strm1.print(",");

    strm1.print(gps.date);
```

```
strm1.print(",002.1,W*");
    cs_GPRMC = checksum(NMEA_gps_GPRMC);
    if (cs_GPRMC < 0x10) strm1.print('0');
    strm1.print(cs_GPRMC, HEX);
    Serial2.println(NMEA_gps_GPRMC);
//_____________________________________________________________________________
__//
//WIND NMEA
    char NMEA_wind [23]; //ex: $NRMWV,179,R,55,N,A*1e
    byte cs_wind; //checksum compass
    PString strm2(NMEA_wind, sizeof(NMEA_wind));
    strm2.print("$NRMWV,");
    strm2.print(wind.Direction);
    strm2.print(",R,");
    strm2.print(wind.Speed);
    strm2.print(",N,A*");
    cs_wind = checksum(NMEA_wind);
    if (cs_wind < 0x10) strm2.print('0');
    strm2.print(cs_wind, HEX);
    Serial2.println(NMEA_wind);
//_____________________________________________________________________________
__//
//RUDDER NMEA
    char NMEA_rudder [23];
    byte cs_rudder; //checksum compass
    PString strm3(NMEA_rudder, sizeof(NMEA_rudder));
    strm3.print("$RSA,");
    strm3.print(rudder.NMEA_rudder);
    strm3.print(",A,,*");
    cs_rudder = checksum(NMEA_rudder);
    if (cs_rudder < 0x10) strm3.print('0');
    strm3.print(cs_rudder, HEX);
```

```
Serial2.println(NMEA_rudder);
//_____
__//
//PITCH ROLL NMEA
      char shrSentence [50]; //ex:$IIXDR,A,5.0,,PTCH,A,12.0,,ROLL,*hh<CR><LF>
      byte csp;
      PString strp(shrSentence, sizeof(shrSentence));
      strp.print("$$IIXDR,A,");
      strp.print(pitch,1);
      strp.print(",,PTCH,A,");
      strp.print(roll,1);
      strp.print(",,ROLL,*");
      csp = checksum(shrSentence);
      if (csp < 0x10) strp.print('0');
      strp.print(csp, HEX);
      Serial2.println(shrSentence);



      //update_debug (); //Print serial debug
  }


  //myFile = SD.open("LOG.txt", FILE_WRITE);
  //myFile.print(gps.lat); myFile.print(","); myFile.print(gps.lon); myFile.print(",");
myFile.print(gps.Speed_kts); myFile.print(",");
  //myFile.print(wind.Direction); myFile.print(","); myFile.println(sail.angle);
  //myFile.close();
}


void update_debug (){


    Serial2.print("Autonomous: ");     Serial2.println(autonomous);
```

```
if (autonomous == false) {
  Serial2.print("Azi_Manual: ");    Serial2.println(AziManual);
  Serial2.print("Sail_Manual: ");   Serial2.println(sailManual);
 }
 if (autonomous == true) {
  Serial2.print("sailAuto: ");      Serial2.println(sail.angle);
 }
 Serial2.print("c.head: ");        Serial2.println(head);
 Serial2.print("c.roll: ");        Serial2.println(roll);
 Serial2.print("c.pitch: ");       Serial2.println(pitch);

 Serial2.print("w.dir: ");         Serial2.println(wind.Direction);
 Serial2.print("w.speed: ");       Serial2.println(wind.Speed);
 Serial2.print("w.RPM: ");         Serial2.println(wind.RPM);
 Serial2.print("rudder: ");        Serial2.println(rudder.angle);

 Serial2.print("GPS LOCK = ");     Serial2.println(gps.GPS_LOCK);
 if (gps.GPS_LOCK) {
  Serial2.print("gps.fix_age: ");   Serial2.println(gps.fix_age);
  Serial2.print("gps.lat: ");       Serial2.println(gps.lat);
  Serial2.print("gps.lon: ");       Serial2.println(gps.lon);
  Serial2.print("gps.Speed_kts: "); Serial2.println(gps.Speed_kts);
  Serial2.print("gps.GPS_head: ");  Serial2.println(gps.GPS_head);
  Serial2.print("distancia: ");     Serial2.println(DistCalc);
  Serial2.print("AziGPS: ");        Serial2.println(AziCalc);
  Serial2.print("Time: ");          Serial2.println(gps.time);
  Serial2.print("Date: ");          Serial2.println(gps.date);
 }
 Serial.print("wpnumber: ");      Serial.println(wp_number);
 Serial.print("wplat: ");         Serial.println(wp.wplat[wp_number]);
 Serial.print("wplon: ");         Serial.println(wp.wplon[wp_number]);
```

```
}

void wp_test() {
  if (DistCalc < wp_min_dist)
    wp_number++;
  if (wp_number > 9)
    wp_number = 0;
  if ((wp.wplat[wp_number] == 0) || (wp.wplon[wp_number] == 0))
    wp_number = 0;
}

float head_to_go(float AziCalc, float windDirection) {

  float head;

  if (AziCalc - windDirection < 45 && AziCalc - windDirection > - 45) {
    if (AziCalc >= windDirection) {
      head += 45 - (AziCalc - windDirection);
    }
    if (AziCalc <= windDirection) {
      head -= 45 - (windDirection - AziCalc);
    }
  }
  if (AziCalc - windDirection < - 315 && AziCalc - windDirection > 315) {
    if (AziCalc >= windDirection) {
      head -= 45 - (360 - AziCalc + windDirection);
    }
    if (AziCalc <= windDirection) {
      head += 45 - (360 - windDirection + AziCalc);
    }
  }
```

```
if (head > 360) {
   head -= 360;
  }
  if (head < 0) {
   head += 360;
  }
  return (head);
}


int head_I2C_Update() {

  sensors_event_t event;
  bno.getEvent(&event);
  head = ((float)event.orientation.x);
  roll = ((float)event.orientation.y);
  pitch = ((float)event.orientation.z);
  uint8_t sys, gyro, accel, mag = 0;
  bno.getCalibration(&sys, &gyro, &accel, &mag);
}


void displaySensorDetails(void)
{
  sensor_t sensor;
  bno.getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" xxx");
```

```
Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" xxx");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(250);
}

byte checksum(char* str) //Checksum function
{
   byte cs = 0;
   for (unsigned int n = 1; n < strlen(str) - 1; n++)
   {
     cs ^= str[n];
   }
   return cs;
 }
```

Apêndice B

```
#include "Arduino.h"
#include <TinyGPS.h>

class GPS{
TinyGPS gps_nr1;
public:
 //*****************************************
 long lat,lon;
 int GPS_head;
 float Speed_kts;
 unsigned long fix_age;
 unsigned long time, date;
 unsigned long time_GPS;
 bool GPS_LOCK = false;
 float NMEA_lat;
 float NMEA_lon;
 float LAT_NMEA;
 float LON_NMEA;
 //*****************************************
  void Update(){
```

```
while(Serial1.available()){
      if(gps_nr1.encode(Serial1.read())){
         gps_nr1.get_datetime(&date, &time, &fix_age);
         time_GPS =time;
         gps_nr1.get_position(&lat,&lon,&fix_age);
         Speed_kts = gps_nr1.f_speed_knots();
         GPS_head = gps_nr1.f_course();
         NMEA_lat = lat/1000000.0;
         NMEA_lon = abs(lon)/1000000.0;
         }

         if (fix_age == TinyGPS::GPS_INVALID_AGE){
          GPS_LOCK = false;
         }
          else if (lat < 1){
                GPS_LOCK = false;
          }
                else{
                GPS_LOCK = true;
                }
         }

         LAT_NMEA = Convert_GGmm_dddddd(NMEA_lat);
         LON_NMEA = Convert_GGmm_dddddd(NMEA_lon);
       }


   float Convert_GGmm_dddddd(float coordenada){
   int x =  coordenada;
   float y = (coordenada - x)*60;
   String z = String(x);
```

```
if (y < 10){
   z = z + "0"+ String(y,6);
   }
   else{
   z = z + String(y,6);
   }
   float w = z.toFloat();
   //w = w*100;
   return w;
         }



  float CalcDist(float flat1,float flon1,float x2lat,float x2lon){
// flat1 = our current latitude. flat is from the gps data.
// flon1 = our current longitude. flon is from the fps data.

   flat1=flat1/1000000;
   flon1=flon1/1000000;
   float dist_calc=0;
   float dist_calc2=0;
   float diflat=0;
   float diflon=0;

   diflat=radians(x2lat-flat1);
   flat1=radians(flat1);
   x2lat=radians(x2lat);
   diflon=radians((x2lon)-(flon1));
   dist_calc = (sin(diflat/2.0)*sin(diflat/2.0));
   dist_calc2= cos(flat1);
   dist_calc2*=cos(x2lat);
```

```
dist_calc2*=sin(diflon/2.0);
   dist_calc2*=sin(diflon/2.0);
   dist_calc +=dist_calc2;
   dist_calc = (2*atan2(sqrt(dist_calc),sqrt(1.0-dist_calc)));
   dist_calc*=6371000.0;
   return(dist_calc);
   }

   float CalcAzi(float flat1,float flon1,float x2lat,float x2lon){

   flat1=flat1/100000;
   flon1=flon1/100000;
   flat1 = radians(flat1);

   flon1 = radians(flon1);
   x2lat = radians(x2lat);
   x2lon = radians(x2lon);

   float heading;
   heading = atan2(sin(x2lon-flon1)*cos(x2lat),cos(flat1)*sin(x2lat)-
sin(flat1)*cos(x2lat)*cos(x2lon-flon1)),2*3.1415926535;
   heading = heading*180/3.1415926535;
   float head = heading;

      if(head<0){
            head+=360;
            }
      if(head>360){
            head-=360;
            }
      return(head);
```

```
}

   void Test(){
   bool a = true;
   String stringOne= "";
   while (a){
   while (Serial1.available()){
   char c = Serial1.read();
   if (c== '\n'){
     Serial2.println(stringOne);
     stringOne ="";
   }
   else{
   stringOne =  String(stringOne + c );
     }
    }
   if (Serial2.available()) {
   String Comand = Serial2.readString();
   if (Comand == "Normal"){ // End Cicle, if the user send the comand "Normal";
     Serial2.println("Normal mode");
     a = false;
     }
    }
   }
}

};
```

Apêndice C

```
#include "Arduino.h"
#include <math.h>

class Rudder{
public:
int angle = 68;
int middle_rudder = 98;      // EB < 98 < BB
int correction_speed = 1;
int NMEA_rudder;

int Update(int compass_head, int azi){

   int delta = compass_head - azi; // calcula a diferença
   if (delta> 180){
      delta = 360 -  delta;
   }
   if (delta < (-180)){
      delta = 360 + delta;
      delta = -delta;
   }
```

```
}
    angle = (middle_rudder+(correction_speed*delta));


   if(angle>128){
     angle=128;
            }
   if(angle<58){
     angle=58;
     }
   NMEA_rudder = middle_rudder-angle;
   return(angle);
  }
};
```

Apêndice D

```
#include "Arduino.h"



class Sail{
public:
int angle = 180;
long double time_update_sail = 0;
void Update(int WDirection, bool autonomous){

 if (autonomous == false){
    Serial3.println(WDirection);
  }

 if (autonomous == true){
  if (WDirection > 180 && WDirection < 225)
    angle = WDirection + 90;
  if (WDirection > 135 && WDirection < 180)
    angle = WDirection - 90;
  if (WDirection > 0 && WDirection < 135)
    angle = WDirection - 15;
  if (WDirection > 225 && WDirection < 360)
    angle = WDirection + 15;
 }
     Serial3.print(angle);
  }
};
```

Apêndice E

```
#include "Arduino.h"

class WP{
public:
 //*****************************************
   float wplat[9];
   float wplon[9];
 //*****************************************

   void Update(){
     wplat[0]= 38.67015;
     wplon[0]= -9.14484;

     wplat[1]= 38.66847;
     wplon[1]= -9.14506;

     wplat[2]= 38.66936;
     wplon[2]= -9.14279;

     wplat[3]= 38.66958;
     wplon[3]= -9.14453;

     wplat[4]= 38.67061;
     wplon[4]= -9.14309;
   }
};
```

Apêndice F

```
#include "Arduino.h"


class Wind {


  public:
   //****************************************
   int Direction;
   //int mast_angle = 180;
   int Speed;
   int RPM;
   int pre_val = 0;
   int hallState = 0;
   unsigned long  t; //time variables
   int norm_pwm_value_0_360;
   float readingsX[50];
   float readingsY[50];// the readings from the analog input
   int readIndex = 0;              // the index of the current reading
   float totalX = 0;
   float totalY = 0;// the running total
   float averageX = 0;
   float averageY = 0;// the average
```

```
//****************************************


  void Update(float pwm_value, int sig, unsigned long cur_t) {


   norm_pwm_value_0_360 = 360*((pwm_value - 3)/(811-3))-87;
 // subtract the last reading:
 totalX = totalX - readingsX[readIndex];
 totalY = totalY - readingsY[readIndex];
 // read from the sensor:
 readingsX[readIndex] = cos((norm_pwm_value_0_360*3.14159)/180);
 readingsY[readIndex] = sin((norm_pwm_value_0_360*3.14159)/180);
 // add the reading to the total:
 totalX = totalX + readingsX[readIndex];
 totalY = totalY + readingsY[readIndex];
 // advance to the next position in the array:
 readIndex = readIndex + 1;

 // if we're at the end of the array...
 if (readIndex >= 50) {
  // ...wrap around to the beginning:
  readIndex = 0;
 }

 // calculate the average:
 averageX = totalX / 50;
 averageY = totalY / 50;
 //convert to polar:
 Direction = (atan2(averageY,averageX)*180)/3.14159;
 // send it to the computer as ASCII digits
 if (Direction < 0){
```

```
Direction *= -1;
 }
 else {
  Direction -= 360;
  Direction = abs(Direction);
 }
 delay(1);
   // read the state of the hall effect sensor:
  hallState = digitalRead(2);

 if (hallState == LOW && pre_val == 1) {
  cur_t = micros();
  int RPM = (1000000 * 60 / (cur_t - t));
  Speed = (RPM + 246.68)/75.334; //y=(x + 246.68)/75.334
  t = micros();
  //Serial.println(Speed);
 }
 pre_val = hallState;
  }
};
```

Apêndice G

```
#include "DFRobotVeyronBrushed.h"

#define VALOR_MAX_POTENCIOMETRO_VELA 1000
#define VALOR_MIN_POTENCIOMETRO_VELA 100

DFRobotVeyronBrushed VeyronBrushed;

 int angle = 660;
 int sail_read1, sail_read2, sail_angle1, sail_angle2;
 int diference1, diference2;
 int aux = 180;
 long double millis_update = 0;

 int sail_error = 2;

 int id = 1;           //The ID of the Veyron
 int motorNumber = 1;    //The motor number: 1 for M1 and 2 for M2
 int speed;            //Store the speed of the motor in RPM (Revolutions Per Minute)

 bool Normal = true;
```

```
void setup()
{

  Serial3.begin(57600);
  Serial.begin(57600);
  VeyronBrushed.begin(Serial3);
  Serial.println("a iniciar");
}

void loop()
{
  sail_read1 = analogRead(A6);
  sail_read2 = analogRead(A5);
  if (sail_read1 > VALOR_MAX_POTENCIOMETRO_VELA || sail_read1 <
VALOR_MIN_POTENCIOMETRO_VELA || sail_read2 > VALOR_MAX_POTENCIOMETRO_VELA ||
sail_read2 < VALOR_MIN_POTENCIOMETRO_VELA)
  safe();
  Serial.print("sail 1: "); Serial.println(analogRead(A6));
  Serial.print("sail 2: "); Serial.println(analogRead(A5));




   while (Serial.available()) {
        aux = Serial.parseInt();
        Serial.print("novo angulo: "); Serial.println(aux);
        if(aux > 5 && aux < 355){
        angle = aux;
        Normal = true;
        }
          else{
            if(aux == 666){
```

```
Normal = false;
          }
          else{
         Serial.print("not a valid angle! Try again mate. Value: "); Serial.println(aux);
         delay(1000);
         }
    }
 }
 if (Normal == true){
       normal();
       }
       else{
       borboleta();
       }
 }



 void normal(){
   if(angle > 20 && angle < 340){
        speed = - (0.8633*angle+380 - sail_read1); //0.8633*angle+330.2 - sail_read1
        Serial.print("speeed :    "); Serial.println(speed);
   if ( speed > sail_error ||  speed < -sail_error){
       speed_normalization();
       speed = speed*1.1;
       VeyronBrushed.setSpeed(id, 1, speed);
          }
        else{
        VeyronBrushed.setSpeed(id, 1, 0);
        }
```

```
speed = - (0.8578*angle+520 - sail_read2);   //Update sail 2 //0.8578*angle+307 - sail_read2
      if ( speed > sail_error ||  speed < -sail_error){
        speed_normalization();
        speed = speed*1.0;
        VeyronBrushed.setSpeed(id, 2, speed);
        }
         else{
         VeyronBrushed.setSpeed(id, 2, 0);
         }

   }
}


void borboleta(){
         speed = sail_read1+30 - 820; // Update sail 1
     if ( speed > sail_error ||  speed < -sail_error){
        speed_normalization();
        VeyronBrushed.setSpeed(id, 1, speed);//Stop the motor 1
           }
         else{
         VeyronBrushed.setSpeed(id, 1, 0);
         }

          speed = sail_read2-20 - 460;   //Update sail 2
      if ( speed > sail_error ||  speed < -sail_error){
        speed_normalization();
        VeyronBrushed.setSpeed(id, 2, speed);//Stop the motor 1
           }
         else{
         VeyronBrushed.setSpeed(id, 2, 0);
```

```
}
    }

void speed_normalization(){
        if(speed  > 0)
        speed = 110;
        if (speed < 0)
        speed = -110;
}



void safe(){
  VeyronBrushed.setSpeed(id, 1, 0);
  VeyronBrushed.setSpeed(id, 2, 0);
       while(true) {
        Serial.println("erro a vela saiu do sitio");
        Serial.print("sail 1: "); Serial.println(analogRead(A6));
        Serial.print("sail 2: "); Serial.println(analogRead(A5));
        delay(500);
        }
       }
```