ESCOLA NAVAL

talant de bien faire

Pedro Emanuel Queirós da Silva Marques

Monitorização e gestão de energia no veleiro autónomo FASt

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval Ramo de Armas e Eletrónica

Alfeite
2015

ESCOLA NAVAL

talanteve bien faire

ASPOF EN-AEL Pedro Emanuel Queirós da Silva Marques

Monitorização e gestão de energia no veleiro autónomo FASt

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval – Ramo Armas e Eletrónica

Orientação de: Prof. Dr. José Carlos Alves
Coorientação de: Prof. Dr. Vitor Sousa Lobo

O Aluno Mestrando — O Orientador

[Pedro Marques] — [Prof. Dr. José Alves]

Alfeite

2015

Resumo

Os veículos autónomos fazem cada vez mais parte do nosso quotidiano, seja em lazer ou para outro tipo tarefas que sejam de difícil realização por parte do ser humano. Apesar de serem eficazes nas tarefas que realizam, estes estão limitados devido à autonomia da sua bateria.

Este trabalho tem como objetivo o aumento da autonomia de um veleiro autónomo. Para tal foi desenvolvido um sistema auxiliar capaz de monitorizar e gerir a energia elétrica disponível no mesmo.

Nesta dissertação, começou-se por definir quais as especificações pretendidas para os veleiros autónomos e, posteriormente os requisitos que o sistema auxiliar deveria apresentar. Depois de discriminadas todas as especificações do sistema de gestão de energia, elaborou-se um protótipo de uma placa que fizesse a gestão das baterias do veleiro autónomo FASt.

Para a construção do protótipo efetuou-se um estudo dos vários microcontroladores existentes no mercado, bem como dos componentes necessários para o projeto. A escolha recaiu sobre o microcontrolador ATMega328P, sendo esse o centro de todo o sistema. Depois de escolhido o controlador, o mesmo foi programado para cumprir com os requisitos previamente identificados, fazendo uso do software e da linguagem utilizada na plataforma Arduino. A construção de placa do protótipo foi realizada já na fase final do projeto e posteriormente foram incorporados todos os componentes e realizados vários testes, recorrendo a uma simulação com recurso a um computador.

Dos resultados obtidos conclui-se que este sistema auxiliar é uma mais valia para o veleiro, uma vez que é um sistema robusto, com uma elevada autonomia e capaz de assumir o comando do veleiro em caso de necessidade. Contudo, a integração do sistema a bordo do veleiro FASt está dependente da realização de mais testes.

Abstract

Nowadays, autonomous vehicles are a part of our daily lives. They are used not only for leisure, but also for other tasks that are difficult to perform. Despite these advantages, the autonomy of the battery is a limitation.

This project aims to increase the autonomy of an autonomous sailboat. Therefore, a system capable of monitoring and manage the energy available was developed.

In this dissertation, initially were defined the specifications for autonomous sailboats and, subsequently, the requirements that the system should provide. After all specifications of the power management system were discriminated, a prototype of a board capable of manage the autonomous sailboat FASt's batteries was constructed.

To build the prototype, a market study of the various microcontrollers and components required for the project was made. In the end, the ATMEGA328P microcontroller was chosen. Subsequently, the controller was programmed to accomplish the requirements previously identified, using the software and language from Arduino platform. After prototype board construction and incorporation of all components, several simulation tests were preformed, using a computer.

The results obtained make possible to conclude that the system developed is an asset to the sailboat, since it is a robust system, with a high autonomy and capable to take the sailboat command, if necessary. However, more tests should be done in order to allow the integration of the system on sailboat FASt.

Agradecimentos

Quero agradecer a todos os que contribuíram, de alguma maneira, para a concretização deste objetivo, em especial ao Professor José Carlos Alves, por todo o apoio dado, ao Professor Vítor Sousa Lobo, ao CFR Ribeiro Correia, pela paciência nas reuniões semanais e a todos os camaradas da classe de Engenheiros Navais ramo de Armas e Eletrónica, pelo apoio dado.
Um agradecimento muito especial à Mariana Leitão, pela ajuda na revisão da tese, e também à Maria João Marques, Maria Aurora Marques e João Marques por toda a força que me deram durante este ano.
A todos vós, o meu muito obrigado.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AIS - *Automatic Information System*
AOSN - *Autonomous Oceanographic Sampling Network*
ASV - *Autonomous Surface Vehicle*
AUV - *Autonomous Underwater Vehicle*
BBB - *Beagle Bone Black*
DEEC - Departamento de Engenharia Eletrónica e de Computadores
ENSTA Bretagne - *École nationale supérieure de techniques avancées Bretagne*
FASt - FEUP Autonomous Sailboat
FEUP - Faculdade de Engenharia da Universidade do Porto
FPGA - *Field Programmable Gate Array*
GPS - *Global Position System*
GTOPP - *Global Tagging of Pelagic Predators*
I2C - *Inter-Integrated Circuit*
INNOC - *Austrian Society for Innovative Computer Sciences*
IRSC - *International Robotic Sailing Conference*
MISO - *Master In Slave Out*
MOOP - *Miniature Ocean Observation Platforms*
MOSI - *Master Out Slave In*
NM - *Nautical Milees*
NOAA - *National Oceanic and Atmospheric Administration*
PMEL - *Pacific Marine Environmental Laboratory*
PMU - *Power Management Unit*
PWM - *Pulse Width Modulation*
SCL - *Serial Clock*
SDA - *Serial Data*
SPI - *Serial Peripheral Interface*
SS - *Slave Select*
SWOT - *Strengths, Weaknesses, Opportunities, Threats*
UAV - *Unmanned Aerial Vehicles*
USART - *Universal Synchronous Asynchronous Receiver Transmitter*
USV - *Unmanned Surface Vehicles*
VAIMOS - *Voilier Autonome Instrumenté pour Mesures Océanographiques de Surface*
WRSC - *World Robotic Sailing Championships*

Capítulo 1

Introdução

O planeta terra é conhecido por todos como "Planeta Azul"e isso deve-se à grande área de oceanos existente na nossa superfície. Segundo a NOAA (*National Oceanic and Atmospheric Administration*), os oceanos são "*grandes porções de água salgada, que cobrem, aproximadamente, 71% da superfície da terra*". Devido a grande parte dessa área estar a mais de 3000 m de profundidade, os oceanos ainda estão pouco ou nada explorados, ou seja, existe ainda uma grande área do nosso planeta que nós não conhecemos devidamente e com recursos inexplorados e, provavelmente, com muito valor económico. A exploração dos oceanos é de interesse de várias áreas, como a ecologia, o ambiente, recursos energéticos ou até mesmo por questões de segurança. [1, 2]

Existe portanto, a necessidade de ter dispositivos capazes de explorar os nossos oceanos por longos períodos de tempo, com baixos custos de operação e manutenção e que permitam fazer uma monitorização contínua com equipamentos embarcados capazes de recolher e analisar dados sobre os oceanos. Contudo, estes dispositivos não podem andar "ao sabor da maré", temos que ser capazes de os controlar remotamente, de os fazer seguir um determinado trajeto ou seguir automaticamente animais marinhos, objetos à deriva ou manchas de crude, importantes para o maior conhecimento dos oceanos.

Com esta necessidade identificada, surgiram propostas de dispositivos que cumprem estes requisitos e que são capazes de desempenhar estas funções, contudo os veleiros autónomos são, de todas as propostas, os que provam ter uma capacidade de navegação, manobrabilidade e velocidade ideal para este tipo de missões, assim como são bastante eficazes a nível energético. Embora a energia para a propulsão seja limpa, gratuita e inesgotável, necessita-se de energia elétrica para alimentar atuadores que controlam as

velas, o leme, a eletrónica responsável pela navegação e os sensores para a recolha de dados.

É fácil a bordo de um veleiro autónomo obter a energia elétrica necessária para alimentar todos este equipamentos. O recurso a um painel fotovoltaico é a forma mais eficaz e menos intrusiva de obter energia elétrica. Contudo, estamos sempre sujeitos à inconstância do Sol devido às condições atmosféricas que podem variar de um dia para o outro, o que significa que a energia produzida no decorrer da missão é muito variável.

Para assegurar uma navegação à vela eficaz, com uma dinâmica lenta, requisitos computacionais baixos e sem elevada complexidade de navegação, não necessitamos mais do que um simples piloto automático que mantenha um rumo pré-determinado. No entanto, para as missões de exploração oceanográfica, como varrimento de um área onde é crucial manter um rumo fixo, ou seguir um rumo através de informações de sensores, já é necessário algo mais complexo, com maior capacidade de processamento e com maior velocidade e precisão no movimento dos atuadores, que garanta uma navegação mais precisa.

Com estas condições, é necessário ter um sistema que seja capaz de gerir os vários sistemas de bordo de forma a prioritizar a alimentação de energia aos vários componentes do veleiro, em função da energia disponível a bordo, da previsão de energia que se poderá vir a obter no curto prazo e do tipo de ação que está em curso no decorrer da missão.

1.1 Objetivo do trabalho

Este trabalho surge da necessidade de dotar o veleiro da Faculdade de Engenharia da Universidade do Porto, o FASt (FEUP Autonomous Sailboat), de uma maior autonomia energética, para assim ser capaz de realizar missões com um alcance elevado. O objetivo é desenvolver uma unidade de gestão de energia (daqui em diante designada por PMU) para monitorizar e gerir a energia elétrica disponível no veleiro e definir estratégias de navegação, tendo em conta o estado de energia do mesmo.

A PMU (*Power Management Unit*) corresponde a alguns requisitos para potencializar ao máximo o veleiro, como operar de forma independente do computador principal do FASt; possuir um fonte de alimentação diferente do veleiro; analisar periodicamente o estado da carga das baterias e definir modos de operação adequados à reserva de energia disponível.

Este sistema implementa ainda a função de *Watchdog* do computador principal, uma função onde implementa estratégias de navegação concebidas tendo em vista a sua integração no sistema de navegação e controlo do FASt e por fim, uma função de repouso total de todo o sistema eletrónico do veleiro.

1.2 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais cinco capítulos.

No capítulo 2 é descrito o estado da arte na matéria da navegação à vela e da automação de embarcações, começando pela sua origem até aos trabalhos que tem vindo a ser desenvolvidos.

No capítulo 3 define-se os parâmetros do sistema a implementar para resolver o problema da gestão energética, desde a definição do conceito de PMU, identifica-se os requisitos mínimos do sistema e idealiza-se a solução a construir.

O capítulo 4 aborda os estudos de consumo e autonomia realizados, que permitiram determinar qual a melhor solução a adotar para a construção da PMU. Posteriormente discrimina-se a interação da PMU com os equipamentos do veleiro e projeta-se o consumo teórico da mesma.

No capítulo 5 estão pormenorizados os teste efetuados e os resultados obtidos, que levaram à construção do protótipo que também é analisado neste capítulo. Por fim, o capítulo 6 são apresentadas as conclusões do projeto, feita com recurso a uma análise SWOT (*Strengths, Weaknesses, Opportunities, Threats*), bem como as recomendações e ideias para trabalhos futuros.

Capítulo 2

Estado da Arte

Neste capítulo é feita uma resenha histórica dos barcos à vela, desde as suas origens até à atualidade; posteriormente apresenta-se uma breve introdução teórica sobre os veículos aquáticos autónomos, com especial ênfase nos veleiros autónomos, referindo a sua importância para a sociedade atual, apresentando alguns exemplos, as suas vantagens e as atuais limitações em termos de utilização.

2.1 História e arte da navegação à vela

A navegação à vela é conhecida desde a antiguidade e o povo que dominasse a arte de controlar o vento a navegar estava em vantagem estratégica face aos seus vizinhos. Quer fosse para deslocar bens para fazer comércio, transportar pessoas para colonizar ou conquistar novas cidades através de ataques vindo do mar, domar o vento no alto mar era essencial.

Os povos do mediterrâneo foram os primeiros a aventurar-se a navegar à vela para realizar trocas comerciais de pequena escala e mais tarde, para combater e projetar forças em território inimigo.

Posteriormente, na epopeia dos descobrimentos, a navegação à vela evolui muito graças aos portugueses. Um grande contributo dos portugueses para a navegação à vela foi a invenção das velas latinas. As velas latinas, como podemos ver na Caravela *Vera Cruz*, ilustrada na figura 2.1, permitiam navegar à bolina, ou seja, navegar contra o vento. Com navios de grandes dimensões com uma grande capacidade de carga, as grandes travessias oceânicas que interligaram continentes e estreitaram os povos, criaram rotas comerciais,

como por exemplo a rota das especiarias da Índia que tornou Portugal numa potência mundial.

Figura 2.1: Caravela Vera Cruz, réplica das caravelas usadas nos descobrimentos.

A época seguinte, entre os séculos XVI e XVIII, fica marcada pelo domínio dos mares por parte das grandes potências europeias e pelos combates travados contra navios piratas ou contra outras potências. A arte de navegar ao sabor do vento tornou-se muito importante do ponto de vista bélico e da estratégia militar.

Até ao início do século XIX, a navegação à vela foi o meio mais importante de propulsão das embarcações, altura em que começaram a aparecer os navios a vapor, cujo o primeiro foi o navio apresentado na figura 2.2, o *Clermount*.

Apesar disso, a navegação à vela não foi posta de parte e Portugal foi e continua a ser um exemplo disso. Os portugueses desde cedo aprenderam a dominar o vento e no início do século XX continuaram a utilizaram navios à vela para a pesca do bacalhau. A frota bacalhoeira nacional contava com vários veleiros para realizar viagens marítimas entre o nosso país e os bancos da Terra Nova e Gronelândia. São exemplos desses os navios gémeos Santa Maria Manuela e Creoula, este último representado na figura 2.3, que continuam a navegar nas nossas águas, estando o primeiro a operar ao serviço de uma empresa turística portuguesa e o segundo encontra-se ao serviço da Marinha Portuguesa como Navio de Treino de Mar.

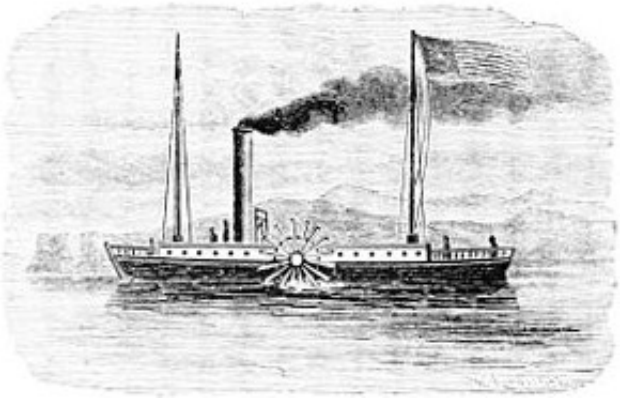


Figura 2.2: *Clermount* - Primeira embarcação a vapor.

Figura 2.3: Creoula no rio Tejo.

Atualmente, a navegação à vela não tem o mesmo peso de outrora, sendo mais virada para atividades lúdicas ou de competição. Os veleiros atuais são bastante sofisticados e cada vez mais aerodinâmicos, recorrendo à computação, à matemática avançada e à engenharia dos materiais criam-se velas e cascos adaptados ao tipo de embarcações. Os veleiros da regata *Volvo Ocean Race*, são o exemplo disso, com velas de grandes dimensões e cascos de materiais leves, desenhados para atingir grandes velocidades.

2.2 O vento como combustível nos transportes marítimos

Navegar à vela é o meio de propulsão que garante uma autonomia de operação virtualmente ilimitada, uma vez que a energia necessária para a sua propulsão provém de um recurso inesgotável, se bem que inconstante, que é o vento. Assim sendo, a navegação à vela apenas é limitada em termos da energia necessária para alimentar os equipamentos de navegação, por exemplo, GPS, VHF, AIS, que são imprescindíveis para a segurança da navegação.

Seguindo a tendência natural da tecnologia, em simplificar as tarefas e tornar mais confortável as mesmas para o homem, a evolução eliminou quase por completo as embarcações à vela no transporte de pessoas e mercadorias. Contudo a consciencialização para a necessidade da redução da pegada ecológica do transporte marítimo tem conduzido ao nascimento de soluções inovadoras, ou algumas já exploradas no passado que voltaram a ser utilizadas no presente.

2.2.1 Barco de Fletter

No início dos anos 20 do século passado, o engenheiro alemão Flettner criou o navio *Buckau*, representado na figura 2.4, o qual aproveitava o Efeito Magnus[1] como força de propulsão, usando como velas rotores verticais atuados por motores elétricos.

Apesar das suas velas não terem o formato ao qual estamos habituados, elas utilizam o vento de uma forma particular de modo a economizar combustível. As velas eram estruturas mecânicas laminares que estavam associadas a motores do navio, e estas captavam a energia do vento para assim auxiliar na propulsão do mesmo.

Na altura, a solução não era competitiva. Com os baixos custos dos combustíveis fósseis, o navio não teve sucesso comercial pretendido, tendo desaparecido dez anos mais tarde num temporal. O mesmo princípio foi aplicado em vários outros navios e em 1983 o investigador Jacques-Yves Cousteau desenvolveu um navio oceanográfico com base nesta tecnologia.

Mais recentemente no ano de 2010, a empresa alemã Enercon construiu um navio de transporte oceânico, representado na figura 2.5, o *E-Ship 1*, com quatro rotores que usam

[1] O Efeito Magnus é o fenómeno pelo qual a rotação de um objeto altera sua trajetória num fluido. Este efeito deriva do Princípio de Bernoulli que diz que em diferentes pontos de uma corrente uniforme, se o fluido se movimenta com velocidades diferentes, nos pontos de maior velocidade observa-se a menor pressão e vice-versa. [3]

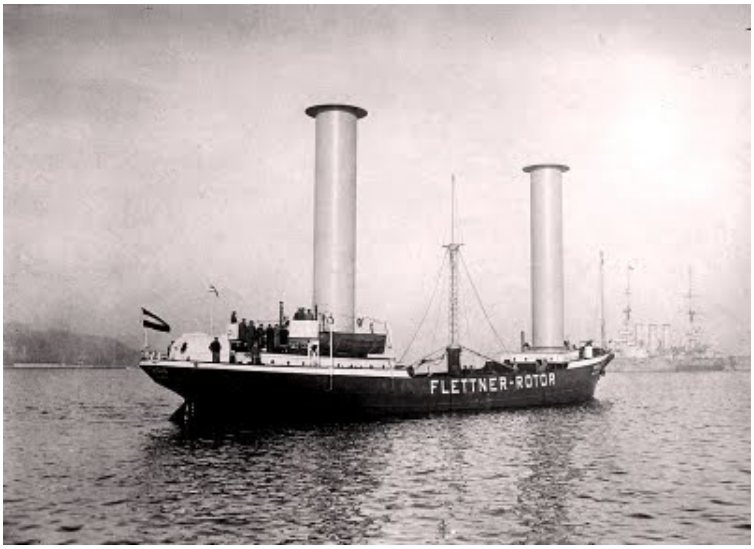


Figura 2.4: *Buckau*, o barco criado por *Flettner*.

o já referido Efeito Magnus, resultando numa poupança de combustível anunciada de 25% face a navios de propulsão convencional, traduzindo-se em menos 1700 t de combustível e menos 5100 t de emissões de CO_2 por ano. [4]

2.2.2 *SkySails*

Motivada pelas mesmas questões ambientais e económicas, surgiu no início do século XXI uma empresa com a proposta de usar "papagaios"gigantes para auxiliar a propulsão de navios de carga convencionais. A solução consiste em usar como vela uma asa semelhante às usadas no *kitesurf*[2], mas com maiores dimensões e com um sistema automático de lançamento e recolha que pode ser instalado em navios já existentes.

O primeiro cargueiro comercial a utilizar esta tecnologia foi o *MS Beluga* figura 2.6, que foi auxiliado, na sua deslocação por uma dessas velas, pela primeira vez, em Janeiro de 2008, entre a Alemanha e a Venezuela. A poupança de combustível foi de 35 %. [5]

2.2.3 Navios Moinho

Além do Efeito de Magnus dos barcos de Flettner e da vela em asa do *SkySails*, outra ideia interessante para aproveitar o vento como meio de propulsão são os navios moinho.

[2] Modalidade aquática que usa uma asa e uma prancha para se deslocar e fazer acrobacias.

Figura 2.5: *E-Ship 1*.

Figura 2.6: O *MS Beluga* a usar o *SkySails*.

A ideia consiste em ligar uma turbina eólica a um motor que faz girar a hélice do navio e produz energia elétrica para ser armazenada em baterias, ou ligar mecanicamente a turbina ao hélice.

Em meados do século XX, o arquiteto Michel Bigoin constrói a embarcação *Xargo II*, que se pode ver na figura 2.7. O modelo tinha 2,3 m de comprimento, 0,51 m de boca e com um peso de 37,5 Kg. Para a sua propulsão tinha um mastro de 1,05 m de altura e uma hélice eólica de 1,64 metros de diâmetro.

Graças ao mastro estar situado no centro da deriva, a embarcação é capaz de navegar a qualquer rumo com o vento a entrar em qualquer direção. Os dados apresentados provam essa faceta com a embarcação a fazer 2 nós com vento de proa com força quatro, 3,6 nós com o mesmo vento de través e 4,5 nós com vento de popa. [6]

Mais recentemente foi construido o catamarã *Apocalipse II*, apresentado na figura 2.8, que se encontra equipado com uma turbina eólica de propulsão.

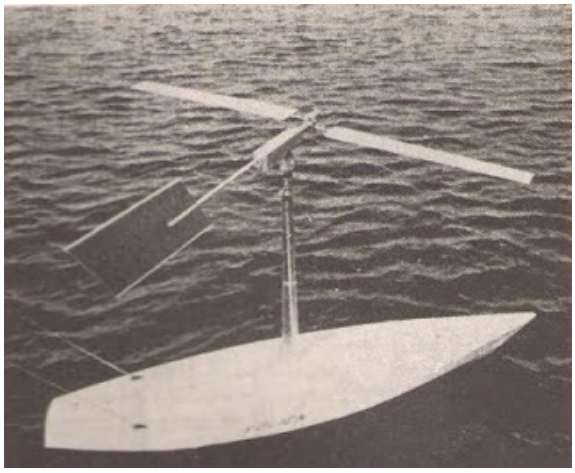


Figura 2.7: A embarcação *Xargo II*.

Estes e outros exemplos mostram que a navegação marítima com recurso à utilização do vento como fonte de energia está lentamente a ser reconsiderada como uma preparação antecipada para o fim anunciado dos combustíveis fósseis.

2.3 Veleiros autónomos

Devido à experiência de muitos anos de estória da navegação à vela e ao *know how* adquirido dos veleiros telecomandados, adaptar um veleiro de forma a torná-lo totalmente autónomo não foi difícil de idealizar.

Figura 2.8: O catamarã *Apoclipse II*.

A navegação à vela, de entre os tipos de veículos autónomos que existem, é ainda o modo de propulsão de embarcações que permite, em termos energéticos, um desempenho francamente superior, em termos de velocidade, navegabilidade e manobrabilidade, tendo em conta outros veículos de superfície de longo *endurance*.

Apesar da dependência do estado do vento e do mar e da incerteza natural associada ao estado das condições atmosféricas, tem capacidade de contornar ventos e correntes desfavoráveis.

Utilizando o vento como meio de propulsão e as técnicas de navegação à vela já há muito conhecidas, conseguiu-se idealizar um veleiro que seja totalmente independente de outras fontes energéticas para a sua propulsão, necessitando apenas de energia para alimentar a eletrónica que compõe os sistemas integrados a bordo. Assim, segundo Cruz N. e Alves J., podemos definir um veleiro autónomo como *"barcos robóticos que usam*

o vento como fonte de energia para a propulsão e que tem a capacidade de controlar as velas e os lemes sem a intervenção humana". [7]

Os veleiros autónomos por serem auto sustentáveis e por terem espaço livre no seu interior para transportar equipamento/sensores, ficam dotados com a capacidade de transportar cargas com grandes dimensões e pesos elevados, algo que outros meios ainda estão um pouco limitados. Assim os veleiros autónomos são capazes de [7]:

- Realizar missões de longa duração;
- Monitorizar grande áreas de uma forma contínua;
- Transmitir dados em tempo real;
- Rebocar sensores, como um sonar;
- Gerar pouco ruído, devido à ausência de motores;
- Reduzir os custos operacionais.

De seguida, são apresentados alguns dos projetos, na área da vela autónoma, que tem contribuído para o desenvolvimento e a modernização da navegação à vela. O principal objetivo de todos eles é o mesmo, equipar um veleiro de forma a que ele se torne capaz de navegar sem intervenção humana.

2.3.1 *SailDrone*

O *SailDrone* (ver figura 2.9) é um veleiro criado pela empresa *Saildrone* em colaboração com a *Marine Science and Technology Foundation*. Esta embarcação tem 5,79 m tanto de altura como de comprimento e é capaz de operar nas mais exigentes condições atmosféricas. Toda a energia elétrica necessária é assegurada pelos painéis solares de bordo no convés e para se deslocar usa uma vela rígida, que lhe confere uma velocidade média entre os 3 e os 5 nós, sendo a velocidade máxima registada de 14 nós. Com um peso aproximado de 100 Kg, este veleiro pode transportar um vasto número de sensores para monitorizar os oceanos, através de missões pré-programas.

No seu currículo tem já provas dadas da sua capacidade operacional com duas travessias entre São Francisco e o Hawai, de 34 dias cada, sendo também importante referir que

já navegou mais de 5000 NM (*Nautical Miles)* e completou 100 dias de navegação em completa autonomia.

Este veleiro vai ser implementado principalmente para estudos de caráter ambiental, tendo já acordos com várias empresas, como a GTOPP (*Global Tagging of Pelagic Predators*), para acompanhamento e monitorização de tubarões, e a NOAA, onde dois *SailDrones* equipados com sensores vão fazer recolha de dados para estudos meteorológicos e aquáticos. [8, 9]

Figura 2.9: *SailDrone*.

2.3.2 VAIMOS

O Instituto Francês de Investigação para a Exploração do Mar, em parceria com a ENSTA Bretagne desenvolveu o veleiro VAIMOS (*Voilier Autonome Instrumenté pour Mesures Océanographiques de Surface*), como se pode ver na figura 2.10, para a recolha de dados oceanográficos. Este foi construído com o objetivo de substituir os meios usados atualmente para a recolha de dados (barcos oceanográficos, boias fixas e boias flutuantes). Os veleiro apresentam características que favorecem a sua utilização em detrimento dos meios usados atualmente, como por exemplo, custos de operação e manutenção inferiores e a sua posição é fácil de controlar e de monitorizar.

Este veleiro tem 3,75 m de comprimento, duas velas convencionais e utiliza o vento e a água para gerar a energia necessária para a eletrónica de bordo. Devido à forma do seu casco, pode transportar uma grande variedade de sensores, recolhendo os mais diversos tipos de dados. [10, 11]

Figura 2.10: VAIMOS.

2.3.3 *Roboat*

Motivados por catástrofes naturais e por desastres ambientais em meio marinho, uma equipa de investigação da INNOC (*Austrian Society for Innovative Computer Sciences*) desenvolveu, no ano de 2006, o *ASV Roboat*, ver figura 2.11, um veleiro robótico completamente autónomo. Programado para realizar manobras de velas complexas, de uma forma autónoma e sem assistência humana, este veleiro é capaz de realizar missões de monitorização e de recolha de amostras em áreas de interesse com um custo reduzido.

Com 3,75 m de comprimento, o casco da embarcação foi originalmente concebido para a iniciação de crianças à navegação à vela, sendo depois adaptado para navegar autonomamente. Com 300 Kg de peso, este veleiro pode ainda levar uma carga de 50 Kg sem alterar o seu comportamento dinâmico no mar. Este veleiro está equipado com um painel solar capaz de produzir até 285 W de potência nas condições ideais, que alimenta

toda a eletrónica de bordo e recarrega as baterias. Tem ainda um célula de metanol que produz 65 W, que atua como *backup* da fonte de energia, em caso de emergência. [10, 12]

Figura 2.11: *RoBoat.*

Para além das aplicações já abordadas por outros veleiros do mesmo tipo, como recolha de dados oceanográficos e missões de reconhecimento, os construtores desta embarcação avançam com outras propostas como, instrutor de vela ou usar a embarcação para levar mantimentos a áreas inacessíveis. Outra proposta importante, e que merece destaque face às anteriores, consiste em aplicar a programação do veleiro como medida de segurança para velejadores solitários, atuando automaticamente no comando e controlo do veleiro caso aconteça algum acidente com o velejador. [10, 12]

2.3.4 *BeagleB*

Aproveitando o casco de um veleiro utilizado por pessoas com deficiências físicas para a prática de vela, a empresa *Robosoft* criou, no início de 2007, o *Beagle-B*, representado na figura 2.12. Com 3,5 m de comprimento e uma vela rígida de fibra de carbono com 2 m de altura, este veleiro foi construido para resistir às forças da natureza que se fazem sentir em alto mar. Equipado com dois painéis solares e um conjunto de quatro baterias de 60 Ah, faz com consiga operar quase indefinidamente desde que os painéis solares mantenham a bateria em carga.

Para além dos sensores que ajudam à navegação, o veleiro está equipado com um sensor oceanográfico multi-parâmetros que permite medir a temperatura da água, salinidade, turbidez, condutividade e a concentração de dióxido de carbono na água.

Um ano após ser criado, o *Beagle B* foi usado pela *Aberystwyth University's Institute of Biological Sciences* com o sensor multi-parâmetro e com um hidrofone, para monitorizar uma comunidade de golfinhos e tentar interpretar o seu comportamento face ao estado da água.[10, 13]

Figura 2.12: *BeagleB.*

2.3.5 MOOP

Os MOOP's, *Miniature Ocean Observation Platforms*, (figura 2.13) são pequenos veleiros autónomos de baixo custo, desenvolvidos com o objetivo de desenvolver os conceitos de navegação autónoma e de vela robótica, bem como com o objetivo de participar na competição *Microtransat* (ver próxima secção).

Ao contrário dos outros veleiros já aqui abordados, os MOOP's tem apenas 0,72 cm de comprimento e um peso aproximado de 4 Kg. Devido ao seu baixo preço de produção estes veleiros podem ser utilizados em missões onde existe um risco elevado e cuja a probabilidade de recuperação dos mesmos seja muito reduzida.

A sua pequena dimensão obrigou a repensar a tecnologia que equipa os comuns veleiros, sendo desenvolvidos vários modelos de equipamento vitais para a navegação, para encontrar os que melhor se adaptavam ao espaço reduzido deste veleiro. [10, 14]

Figura 2.13: MOOP.

2.3.6 Competições de veleiros

Para promover o desenvolvimento deste tipo de embarcações existem duas competições para averiguar qual dos veleiros é o melhor. Uma delas, a *Microtransat Challenge*, é uma competição com o objetivo de ter um veleiro totalmente autónomo a atravessar o Oceano Atlântico, a outra, a *SailBot*, é uma prova que averigua a precisão e velocidade de cada veleiro. Em seguida, são apresentados mais pormenores sobre cada competição.

2.3.6.1 *Microtransat Challenge*

O *Microtransat Challenge* é uma competição internacional criada em 2006, para desenvolver veleiros autónomos de pequena escala. O objetivo principal desta competição é conseguir com que um veleiro totalmente autónomo consiga fazer uma travessia transatlântica.

Associada a esta competição existe a WRSC (*World Robotic Sailing Championships*), que ocorre anualmente desde 2008. A WRSC é um encontro de várias equipas, que normalmente participam na *Microtranst*, para realizar competições de pequena dimensão

e para avaliar o desenvolvimento das várias embarcações em competição. A WRSC está integrada na IRSC (*International Robotic Sailing Conference*), que permite às equipas participantes trocar experiências e conhecimentos, apresentar artigos e novos trabalhos desenvolvidos e discutir sobre o futuro da navegação autónoma. [15]

2.3.6.2 *SailBot*

A *SailBot* é uma competição que tem lugar no Norte da América desde 2006. Esta é uma competição entre alunos nas universidades americanas e canadianas e tem três tipos de classes de embarcações, divididas de acordo com o seu tamanho (1 m, 2 m e até 4 m).

Esta competição baseia-se em cinco pequenas provas, que testam os veleiros em diferentes aspetos como o design, a construção ou a programação do sistema de controlo. [16, 17]

2.4 Outros veículos autónomos

O interesse verificado nas últimas décadas pela monitorização do mar e dos seus recursos naturais conduziu à necessidade de criar veículos robotizados capazes de permanecer no meio marinho durante longos períodos de tempo, mantendo a capacidade de recolher dados do meio marinho e aéreo.

Contudo, com a crise energética atual, somos obrigados a procurar alternativas viáveis ao petróleo para produzir energia. No que diz respeito à navegação marítima, dispomos de um vasto leque de alternativas que podem ser utilizadas para gerar a energia necessária para a locomoção e para os diversos sensores embarcados nos veículos autónomos.

Várias soluções foram propostas nos últimos anos para veículos autónomos (não tripulados) de longo *endurance*, sem motorização ativa, as quais podem ser divididas em dois grupos principais: veículos de superfície ou subaquáticos.

De seguida são apresentados dois exemplos de embarcações autónomas, um ASV (*Autonomous Surface Vehicle*) e um AUV (*Autonomous Underwater Vehicle*), utilizadas para fazer estas missões.

2.4.1 *Wave Glider*

O *Wave Glider* é uma plataforma ASV que tem um sistema híbrido de propulsão e de produção de energia, que aproveita tanto a energia solar como a energia gerada pelas ondas nos oceanos.

Com uma estrutura submersa que se pode ver na figura 2.14, o *Wave Glider* aproveita a energia das ondas e o painel solar no corpo principal da embarcação para gerar a energia necessária para os vários equipamentos e sensores de bordo e também pela propulsão da embarcação.

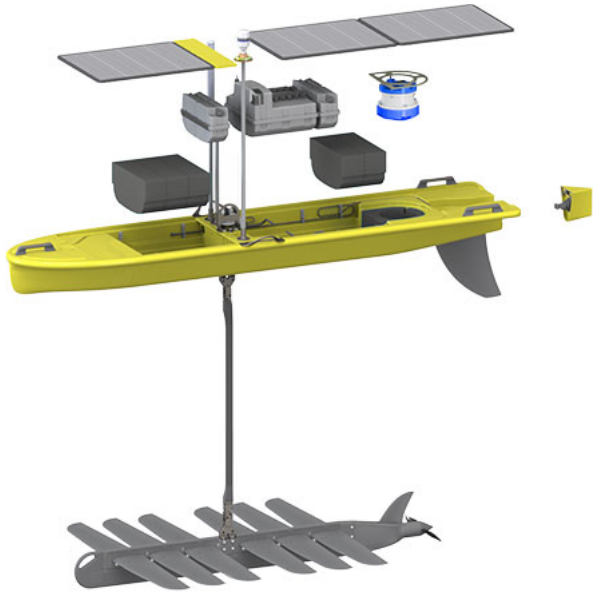


Figura 2.14: *Wave Glider SV3.*

Podendo ser operado individualmente ou em frota, o *Wave Glider* pode ser operado com qualquer tipo de condições ambientais durante todos os dias do ano. A empresa Liquid Robotics anuncia na sua página de *internet*, que o *Wave Glider SV3* tem uma poupança energética de 90 % tendo em conta as outras alternativas para a recolha de dados no mercado. [18]

Consegue transportar vários sensores e rebocar equipamentos capazes de recolher e armazenar vários tipos de dados que ajudam a melhor compreender os oceanos e o seu funcionamento. A sua elevada capacidade de processamento permite-lhe tratar uma grande quantidade de dados e consegue transmiti-los para terra através de uma rede *Wi-fi*, o que permiti ter respostas dos sensores em tempo real, uma mais valia na análise dos dados.

2.4.2 *Autonomous Underwater Gliders*

Um *Autonomous Underwater Glider* é um tipo de AUV que usa propulsores por flutuabilidade como meio de propulsão, isto é, através de alterações de flutuabilidade permite que o veículo tenha um movimento vertical, ou seja, mergulhar e voltar à superfície ou até a uma profundidade definida. Para converter o movimento vertical em horizontal os *gliders* estão equipados com alhetas na sua popa, idênticas às usadas pelos submarinos para manobrar. Vemos na figura 2.15, o *SeaExplore* a mergulhar e as alhetas que o permitem controlar a sua direção na parte final do seu corpo.

Atualmente os *gliders* submergem recorrendo a um de quatro métodos para alterar a sua flutuabilidade, podendo ser com recurso a meios elétricos ou mecânicos para alterar a flutuabilidade, usar a termoclina[3] que se faz sentir na área, a energia gerada pelas ondas é outro meio utilizado para submergir, ou caso seja um meio híbrido, usar os meios convencionais para submergir e regressar a superfície. [19]

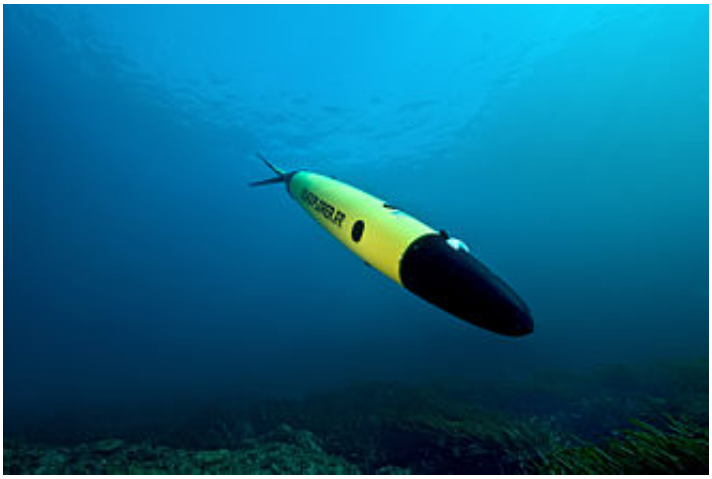


Figura 2.15: *SeaExplore* a mergulhar.

Apesar de ter uma velocidade reduzida, estes veículos representam um aumento significativo de autonomia se comparados com veículos movidos por motores elétricos, permitindo missões de recolha de dados de longa duração, passando de horas para semanas ou meses. A travessia do Oceano Atlântico do *Glider Scarlet Knight*, representada na figura 2.16, é um exemplo disso, precisando de 221 dias para percorrer a distância de 7409,60 Km entre New Jersey, nos Estados Unidos da América, e Baiona, em Espanha.[20, 21]

[3] Termoclina é a variação da temperatura da água em profundidade.

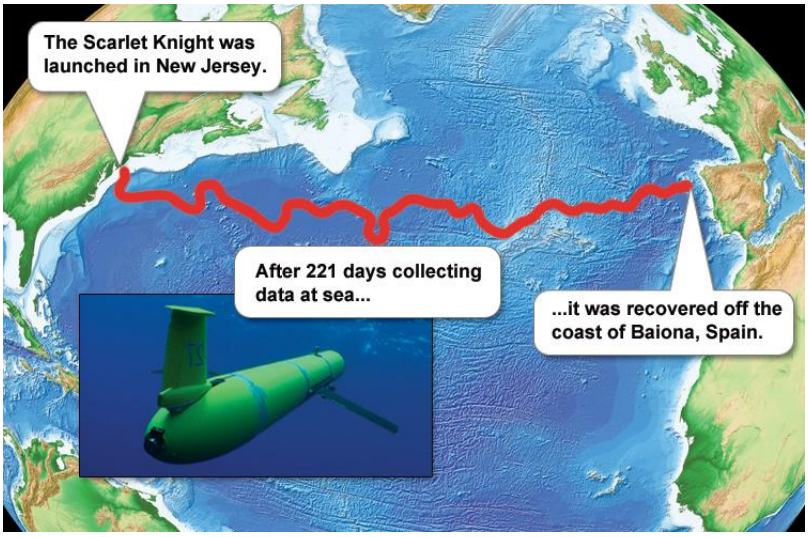


Figura 2.16: Travessia atlântica do *Scarlet Knight*.

Uma vantagem dos *gliders* face aos AUVs mais antigos é que podem seguir um programa de subidas e descidas predefinido, permitindo a aquisição de dados com variação de tempo e espaço de forma muito mais económica e eficaz.

2.5 Aplicação dos veleiros autónomos

Definido o conceito de veleiro autónomo, com exemplares capazes de fazer patrulhas e com provas dadas das suas capacidades, a utilização de veleiros autónomos para a recolha de dados oceanográficos é o próximo passo a dar, uma vez que o uso desses equipamentos trás algumas vantagens. Assim, segundo Cruz, N. e Alves, J., [7] os veleiros autónomos são importantes para a recolha de dados oceânico sobre:

- Circulação oceânica;
- Concentração de clorofila;
- Poluição por hidrocarbonetos, não só à superfície mas também em profundidade;
- Acústica submarina;
- Dinâmica à superfície, mar-atmosfera.

Contudo, a utilização dos veleiros autónomos não se limita à observação oceânica, podendo ser aplicados a outras áreas, por exemplo, vigilância costeira ou missões do âmbito militar. Um exemplo disso é o protótipo desenvolvido pela *Harbor Wing* em parceria com a *US Navy* (ver figura 2.17).

Este veleiro autónomo pode ser equipado com vários equipamentos para vigilância e patrulhamento de costa, tais como câmaras e radares, para assim se obter uma imagem do panorama de superfície de uma determinada zona. [22]

Figura 2.17: Protótipo desenvolvido pela *Harbor Wing* em parceria com a *US Navy*.

Na área da vigilância costeira, os veleiros poderiam ser utilizados para patrulhamento de costa em zonas onde o narcotráfico e a emigração ilegal são problemas constantes, por exemplo no Mar Mediterrâneo; numa missão de busca e salvamento, mantendo assim uma patrulha constante mesmo quando os meios tradicionais de busca não são possíveis de utilizar; ou utilizado para *Harbor Protection*. [7, 23]

No âmbito militar estes veleiros trazem grandes vantagens devido às suas características, sendo indicados para missões de reconhecimento, uma vez que podem transportar vários sensores e iludir o inimigo devido à sua natureza lúdica. Por outro lado, as missões *Mine Survey* e *Mine Countermeasures* poderiam ser feitas por estes veleiros, desde que equipassem os mesmos com o material apropriado. [23]

2.6 FASt - FEUP Autonomous Sailboat

O veleiro FASt é um projeto que está a ser desenvolvido por uma equipa de alunos e professores da Faculdade de Engenharia da Universidade do Porto, em resposta ao *Microtransat Challenge*. No início de 2007, iniciou-se a construção do FASt, tendo sido concluída em 2008, contudo o veleiro vem sofrendo algumas alterações ao modelo original com vista à melhoraria da sua performance. Ao iniciar este projeto, a ideia era construir um veleiro de pequenas dimensões, completamente autónomo, capaz de realizar tarefas de observação oceanográfica e de vigilância, mas que estivesse dentro dos parâmetros definidos pela *Microtransat*, para poder entrar na competição.[7]

Figura 2.18: Veleiro autónomo FASt.

2.6.1 Características físicas do FASt

O veleiro FASt, representado na figura 2.18, foi projetado à escala de grandes veleiros oceânicos, aproveitando assim as provas dadas de resistência às intempéries, característica necessária para a construção de um navio autónomo oceânico. Construído em fibra de vidro e resinas de *polyester*, o FASt tem 2,5 m de comprimento e um peso aproximado de

Tabela 2.1: Principais parâmetros do veleiro autónomo FASt

Comprimento (m)	2,5
Calado (m)	1,5
Boca (m)	0,67
Deslocamento (Kg)	50
Altura Mastro (m)	2,60

50 Kg, cumpre com todos os requisitos exigidos pela *Microtransat* para participar nesta competição. Na tabela 2.1 podem-se observar as principais dimensões do veleiro.

À semelhança dos veleiros convencionais, o FASt possui um patilhão com um lastro de 20 Kg, que proporciona uma maior estabilidade na navegação em mar alto. Equipado com duas velas, ambas controladas por um único atuador, o FASt consegue atingir velocidades médias de 5 nós. Para operar o FASt em segurança, a velocidade do vento não pode ser superior a 20 nós e o estado do mar é outro fator a ter em conta, sendo que não é aconselhável operar com muita vaga. O FASt, devido às suas dimensões e aos dois lemes, é um veleiro com muita manobrabilidade, o que permite fazer uma navegação precisa, sem muitos desvios ao planeamento estabelecido.

2.6.2 Eletrónica do FASt

O FASt está equipado com GPS, agulha magnética, inclinómetro, anemómetro, indicador de direção do vento e sensor de posição da retranca. Além disso tem ainda um sensor de luminosidade, sensores de temperatura e um módulo de monitorização da bateria. Todos estes sensores são essenciais para a navegação, que pode ser controlada remotamente por um utilizador, ou fazer uma navegação autónoma, cumprindo com um planeamento previamente carregado. Para garantir a navegação por controlo remoto, bem como a troca de dados recolhidos com recurso aos seus sensores, o FAST está equipado com uma antena de receção de rádio comando, uma antena *Wi-fi* e uma antena IRIDIUM. A energia elétrica é gerada a bordo através de um painel fotovoltaico de 45 W e é armazenada num conjunto de baterias de 190 W de Li-ion. [24, 25]

2.7 O futuro dos veículos autónomos oceânicos

Como já foi referido anteriormente, existem vários tipos de veículos autónomos, com diferentes características, que se adaptam ao meio aquático, mas não existe um veículo perfeito que se possa definir como o exemplo a adotar. Já foram abordados os pontos fortes dos veleiros e a sua possível utilização, contudo na ausência de vento ele não se vai deslocar.

Para colmatar essas falhas e fazer uma correta utilização de todos os meios disponíveis, o futuro do patrulhamento oceânico pode passar por se criar uma rede onde todos os meios autónomos disponíveis contribuam, enviando dados para um sistema principal, para assim se poder fazer uma patrulha eficaz de um área de interesse oceanográfico.

Essa ideia já foi abordada em 1993, por Thomas B. Curtin, que desenvolveu e criou o conceito de AOSN (*Autonomous Oceanographic Sampling Network*), ou seja, uma rede de sensores que realiza uma constante monitorização de uma área de interesse. [26] Na ideia original de Curtin, a rede seria constituída por AUV's e por sensores acústicos. Esses sensores transmitiriam os dados recolhidos para um satélite ou diretamente para uma estação costeira, enquanto que os AUV's acoplariam a base retransmissora onde descarregavam os seus dados, para depois serem retransmitidos do mesmo modo que os sensores acústicos, como se pode ver na figura 2.19.

Este conceito pode ter sido um pouco ambicioso para a data, contudo baseada nesta ideia e com os avanços da tecnologia, surgiram versões mais recentes, como a de *Monterey Bay* (ver figura 2.20) que recorre a outras plataformas para a recolha de dados, por exemplo, *gliders*, UAVs (*Unmanned Aerial Vehicles*), USVs(*Unmanned Surface Vehicles*) e embarcações convencionais.

Como no caso dos veleiros, esta rede não se destina exclusivamente à recolha de dados oceanográficos, podendo ser ampliado e aplicado às diferentes áreas do interesse civil ou militar, depois do conceito criado.

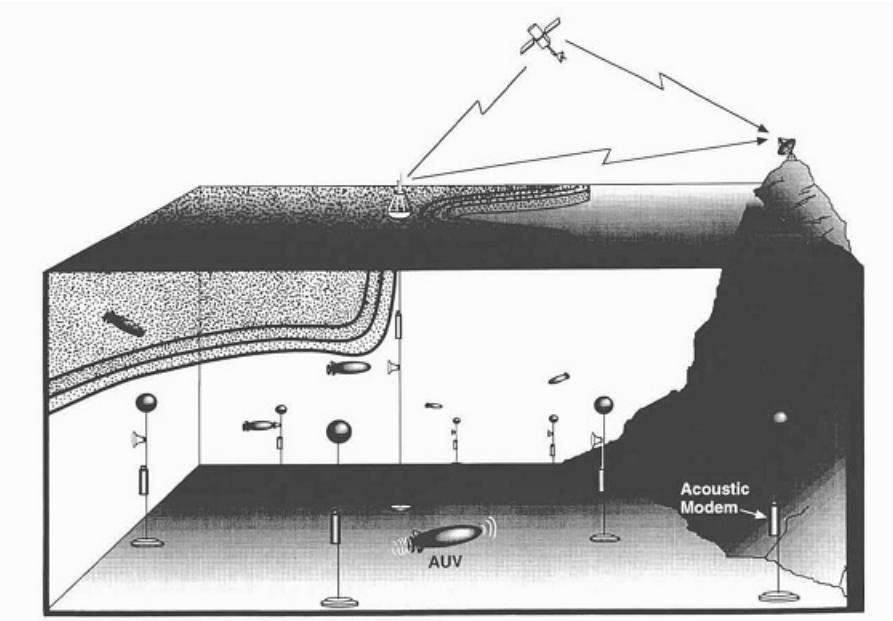


Figura 2.19: Conceito de AOSN por Thomas Curtin.

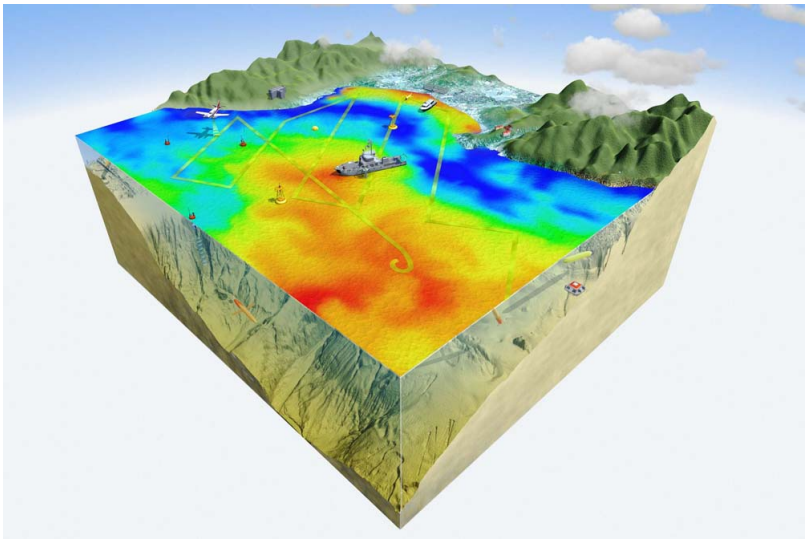


Figura 2.20: AOSN usada em *Monterey Bay*.

Capítulo 3

Arquitetura da solução proposta

Neste capítulo vai ser estudado os aspetos principais da solução proposta para o problema do controlo energético de bordo. Começando por definir o conceito de PMU, identificar os seus requisitos funcionais e operacionais e os seus modos de funcionamento. Em seguida é estudado qual a melhor plataforma para desenvolver esta solução. Por fim, analisa-se o *software* desenvolvido para programar a PMU, onde se explica o funcionamento de cada uma das rotinas principais e também se explica a função de cada rotina secundária.

3.1 PMU

As PMUs já são conhecidas há muito tempo estando já implementadas em muitos equipamentos de uso quotidiano. As PMUs são constituídas por um microcontrolador responsável pelo controlo da energia de qualquer plataforma digital. [27]

Segundo o livro "*Desktop and Portable Systems: Second Edition*", de Owen W. Linzmayer [28], as PMUs são responsáveis por funções como:

1. Monitorizar os níveis de bateria;
2. Carregar as baterias quando necessário;
3. Controlar a energia disponibilizada para os outros circuitos;
4. Desligar sistemas desnecessários;

5. Controlar funções de *power saving* e de *sleep*.

Estas e outras funções são aplicadas, por exemplo, nos computadores pessoais, o que permite que exista uma monitorização constante das operações de consumo de energia, comparando-as com as opções de poupança de bateria selecionadas, permitindo ao utilizador otimizar a sua utilização.

O conceito de PMU está já definido para vários sistemas, inclusive para veleiros autónomos. O *BeagleB*, abordado no capítulo anterior, para além do seu sistema de processamento de dados e de controlo dos atuadores responsáveis pela navegação, possui um sistema separado que monitoriza a corrente consumida e gerada abordo. [13] O sistema do *BeagleB* é composto por quatro transdutores de corrente, modelo LEM CAS 6-NP, ligados respetivamente ao painel solar, ao módulo das baterias, ao atuador do leme e ao atuador da vela, que produzem um nível de voltagem que corresponde à corrente gerada ou consumida pelos equipamentos a que estão associados. Essa voltagem é depois processada por um *Arduino*[1] *Uno* (ATMega328) que envia uma mensagem via protocolo Ethernet ao computador principal, possibilitando um contínuo controlo da produção e dos gastos de energia. [29]

A ideia original do sistema é comutar entre os diversos estados de funcionamento do veleiro, ligando e desligando sistemas e equipamentos, tendo em conta a reserva energética das baterias. A solução para este projeto passa por criar um sistema idêntico ao implementado pelo *BeagleB*, adaptado ao veleiro FASt e incluir no sistema outras funções importantes para a poupança energética do veleiro. Para tal, foram identificados requisitos operacionais e funcionais que vão ser a base para a construção deste sistema.

Os requisitos funcionais

Os requisitos funcionais estão inerentes ao tipo de funções que a PMU vai desempenhar no veleiro e a implicação dessas mesmas funções na navegação e nos dispositivos de bordo. Assim sendo, os requisitos operacionais da PMU são:

Controlar o nível das baterias e da energia produzida pelo painel sonar; Esta função controla o estado de carga das baterias e determina se o painel solar está a carregar as baterias ou não, através da medição da tensão aos terminais do painel solar.

[1] *Arduino* é um projeto *open-source*, constituído por placa de desenvolvimento por uma linguagem de programação, que possibilita a criação de pequenos projetos de pessoais até projetos de grande dimensão.

Monitorizar o bom funcionamento do computador principal; Para evitar que o computador principal entre em algum *loop* que o impeça de navegar em segurança, a PMU vai também funcionar como *Watchdog* ao computador principal, fazendo *reset* caso ocorra algum problema.

Assumir o controlo da navegação com uma função de piloto automático; Quando os níveis de bateria não permitirem uma navegação no máximo das suas capacidades, sem comprometer a missão, a PMU assume o controlo da navegação com uma rotina de *Auto-Pilot*, que apesar de ter menor precisão de rumo é mais poupada a nível energético.

Controlar o ligar e desligar dos equipamentos de bordo; Para evitar picos no consumo de corrente quando se liga o veleiro, a PMU vai controlar e ligar os equipamentos de bordo, com intervalos de tempo reguláveis.

3.1.1 Os requisitos operacionais

Definidos os requisitos funcionais para a PMU, é necessário identificar os requisitos operacionais, ou seja, os aspetos técnicos que a PMU deve ter para conseguir realizar as funções descritas acima. Os requisitos operacionais da PMU são os seguintes:

- Ter uma alimentação independente da energia produzida a bordo, recorrendo a uma pilha alcalina ou a uma bateria;
- Apresentar um baixo consumo energético para maior rendimento da bateria;
- Possuir I/O analógicas para receber dados (três no mínimo, para as baterias e para o sensor de tensão do painel solar);
- Ter I/O digitais para poder ligar e desligar os vários equipamentos disponíveis (router Wi-fi, rádio série, rádio controlo, GPS, CPU,...);
- Possuir I/O PWM para controlar a posição dos lemes (dois no mínimo, um para cada leme);
- Estabelecer comunicações com o CPU, a bússola e as baterias;
- Apresentar modos de *power saving* e de *sleep*, para fazer um uso equilibrado da bateria;
- Ter *Interrupts* internos e externos.

3.1.2 O funcionamento da PMU

O modo de funcionamento da PMU está intrinsecamente associado ao estado das baterias do veleiro. Esta tem na sua programação três modos de funcionamento principais que estão associados ao nível da bateria e um modo secundário responsável pelo arranque do sistema. Na secção do *Software* está descriminado pormenorizadamente o funcionamento de cada um destes quatro modos da PMU. A figura 3.1 apresenta o modo de funcionamento da PMU associado ao estado das baterias.

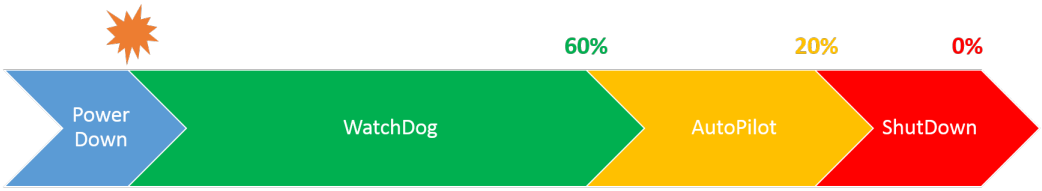


Figura 3.1: Estados de funcionamento da PMU.

A transição do modo *PowerDown* para o modo *WatchDog* deve-se a uma interrupção externa, tal como já havia sido referido. Posteriormente a PMU volta a trocar de estado quando as baterias atingem o nível de 60 %, entrando no modo de *AutoPilot*. Por fim, o modo *ShutDown*, só é acionado quando as baterias estão num nível crítico, ou seja, a baixo dos 20 %. Este é o modo *standard* de funcionamento da PMU, podendo o mesmo ser alterado, alterando os nível de bateria pretendidos.

3.2 Hardware

3.2.1 A plataforma de desenvolvimento e o microcontrolador

Para ir de encontro aos requisitos analisados foi preciso adquirir um microcontrolador que estivesse de acordo com os requisitos operacionais, e que, para além disso, fosse economicamente viável (barato e de fácil acesso no mercado), com dimensões reduzidas, para ocupar o mínimo espaço possível, com uma programação de fácil implementação e que fosse possível de replicar no futuro numa placa construída de raiz.

Antes de escolher um microcontrolador da vasta gama de controladores disponível no mercado, foi necessário escolher a plataforma de desenvolvimento do projeto. A família de plataformas escolhida foi o *Arduino*, por três razões específicas: primeiro porque existe um grande número de utilizadores da plataforma *Arduino*, que formam uma comunidade de ajuda e de partilha de experiências, tornando o trabalho mais fácil de idealizar; a interoperabilidade do código entre as diferentes plataformas existentes na família *Arduino* é outra das razões, pois assim não fica comprometido o aumento de capacidades da PMU, não obrigando à alteração do tipo de plataforma em questão; por fim, o conhecimento *a priori* da linguagem de desenvolvimento desta plataforma, o que facilitou o desenvolvimento do código.

Identificada a família da plataforma de desenvolvimento, analisaram-se as opções que existem para escolher o que melhor se adequa aos requisitos operacionais. A escolha do microcontrolador consiste na escolha da plataforma com a qual vamos trabalhar, uma vez que, cada controlador está associado, por norma, a uma plataforma.

Estão apresentados na tabela 3.1 as diferentes plataformas da marca *Arduino*, assim como o microcontrolador associado a cada plataforma. Das várias plataformas disponíveis, a que melhor se adequa a este projeto é o *Arduino Uno*, figura 3.2.

O *Arduino Uno* é o modelo de referência da marca *Arduino* e existem muitos projetos desenvolvidos com este modelo, o que foi uma vantagem na realização deste trabalho, a nível do apoio da comunidade de utilizadores. Este modelo foi o escolhido, em detrimento dos outros, porque na fase embrionária do projeto é conveniente ter-se uma plataforma que corresponda apenas com o necessário e que não seja equipada com capacidades que não sejam aproveitadas e que iriam consumir energia desnecessariamente.

O modelo do *Arduino Uno* utilizado foi o *Arduino Uno R3*, que tem uma diferença que

Tabela 3.1: Características dos modelos *Arduino*

Plataforma	Processador	Velocidade Processamento (Mhz)	I/O Anl. - Digi. (PWM)	Alimentação (V)	Preço (€)
Uno	ATMega328	16	6 - 14 (6)	7 - 12	20
Due	AT91SAM3X8E	84	12 - 54 (12)	7 - 12	37
Leonardo	ATMega32u4	16	12 - 20(7)	7 - 12	18
Mega 2560	ATMega2560	16	16 - 54 (15)	7 - 12	38
Mega ADK	ATMega2560	16	16 - 54 (15)	7 - 12	45
Micro	ATMega32u4	16	12 - 20(7)	7 - 12	19
Mini	ATMega328	16	8 - 14 (6)	7 - 12	15
Nano	ATMega328	16	8 - 14 (6)	7 - 19	33
Fio	ATMega328P	8	8 - 14 (6)	3.7 - 7	20
Pro	ATMega328	16	6 - 14 (6)	5 - 12	13
Pro Mini	ATMega168	8 16	6 - 14	3.35 - 12 5 - 12	9

Figura 3.2: *Arduino Uno R3*, modelo usado neste projeto.

se evidência face ao modelo original e que favorece a realização deste projeto, o microcontrolador utilizado neste modelo é o ATMega328P que é a versão de baixo consumo do modelo ATMega328. Apesar de não avançar com dados concretos, a Atmel, empresa que desenvolve os microcontroladores, refere que a poupança anda na ordem do miliamperes de corrente. [30]

Para além das características apresentadas na tabela 3.2, o ATMega328P está dotado de outras funcionalidades como uma *Serial Port* USART, *Serial Interfaces* com os protocolos I2C e SPI e *interrupts* internos ou externos. Outro aspeto deste controlador que é importante na construção deste protótipo, são os modos de poupança de energia, vulgarmente conhecidos por modos *sleep*, que corretamente empregues na programação do

Tabela 3.2: Características do ATMega328P-PU

Parâmetros	**Características**
Memória Flash	32 Kbytes
Memória RAM	2 Kbytes
Numero Pinos	28
Freq. Max. Operação	20Mhz
CPU	8-bit AVR
I/O Digitais (PWM)	13 (6)
I/O Analógicas	6

controlador conseguem baixar o consumo do mesmo.

Na figura 3.3, retirada da *datasheet* do microprocessador ATMega328P, podem-se observar os diversos modos de *sleep* e as implicações de cada um deles no funcionamento do *Arduino*.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Figura 3.3: Os modos *sleep* do *Arduino*.

Dos seis modos de *sleep* disponíveis, o mais "profundo"é o modo *Power-Down* e o mais "superficial"o modo *Idle*. Como é visível na figura, os modos de *sleep* podem ser interrompidos de formas diferentes, com recurso a interrupções externas ou através de um tempos pré-definidos.

3.3 Software

Em seguida são explicadas as rotinas base da programação de um *Arduino*, as rotinas principais da PMU e de uma forma mais breve, as rotinas secundárias, também elas importantes para o correto funcionamento da PMU.

3.3.1 Rotinas Base

Na programação utilizada nos microcontroladores ATMega existem duas rotinas base responsáveis pelo comando de todas as ações a tomar pelo controlador, sendo essas a *Setup* e a *Loop*.

A rotina *Setup* é uma rotina de preparação do controlador, ou seja, é nesta rotina que são iniciadas e definidas as variáveis, introduzidos os parâmetros iniciais de todo o programa, onde se define o *pinout* do microcontrolador e se iniciam as bibliotecas de controlo e de comunicação com outros dispositivos. Esta rotina é normalmente executada uma vez no início de cada utilização do controlador, repetindo-se a sua execução quando se faz *reset* ao controlador. [31]

A rotina *Loop* é onde se desenvolve o programa, ou seja, onde se definem os comandos para o microcontrolador que faz executar as funções definidas, como por exemplo, leitura de sensores, atuar em motores ou servomecanismos, trocar informação com outros dispositivos. Ao contrário da rotina anterior, a rotina *Loop* está inserida num ciclo, que tal como o nome indica, volta ao início sempre que conclui todas as ações definidas.[31]

Estas duas rotinas base são obrigatórias mesmo que não sejam necessárias para o programa que pretendemos desenvolver, porém e neste caso em particular, esta estrutura foi muito importante para uma escrita mais eficaz do programa, como se pode constatar em seguida.

3.3.2 Rotinas Principais

O programa possui quatro rotinas principais, cada uma delas equivalente a um estado de funcionamento do controlador. A cada uma das rotinas, estão atribuídas funções específicas e, para as realizar, recorrem a uma ou mais rotinas secundárias para processar dados ou para auxiliar na tomada de decisão. As rotinas principais são todas chamadas na rotina *Loop* e consoante certas condições o programa vai executar uma delas obrigando o microcontrolador a estar num dos quatro estados de poupança possíveis.

A primeira rotina a ser chamada é a *PowerDown*, é o estado de repouso da PMU à espera de receber uma ordem para desencadear todo o processo de arranque do sistema elétrico do veleiro. Depois de ligar todos os equipamentos, a PMU entra na rotina *WatchDog* e faz o controlo do funcionamento do computador principal. Com intervalos de tempo predefinidos, a PMU avalia o estado das baterias e decide para qual das rotinas/estado vai a seguir: mantém o *WatchDog*, ou passa para o *AutoPilot*, efetuando uma navegação de baixo consumo, mas pouco precisa, ou em último recurso desliga todos os equipamentos e entra na rotina de *PowerOff*.

Após uma generalização do funcionamento do programa, cada rotina vai ser explicada mais detalhadamente, dando a conhecer também qual a influência na navegação da rotina em causa.

- *PowerDown*;

  A PMU entra num modo de baixo consumo onde aguarda por uma interrupção externa, proveniente de um relé já existente no veleiro, para desencadear todo o processo de arranque controlado do veleiro.

  Esta rotina está definida para substituir o atual método de ligar e desligar o sistema, que exige uma sobrecarga de corrente por ligar tudo em simultâneo, tornando o arranque e o desligar de todo o sistema mais seguro, mais eficiente e menos exigente para os equipamentos.

  Esta rotina não influência a navegação, uma vez que ela é chamada quando se liga ou desliga o sistema, ou seja, no início ou no fim de uma navegação.

- *Watchdog*;

  Após iniciar todo o sistema, a PMU entra no estado de *Watchdog*, que consiste num estado constante de hibernação, com o objetivo de reduzir o consumo, que só é interrompido momentaneamente para verificar a voltagem das baterias e para receber informação relativa à navegação (rumo e velocidade média do veleiro) provenientes do computador principal.

  Numa navegação normal, esta será a rotina que mais tempo estará a ser executada pela PMU, daí ser das mais importantes na navegação apesar, de não contribuir ou alterar nenhum dado relativo à mesma.

- *AutoPilot*;

Quando as baterias estão num nível baixo, mas que não é crítico ao ponto de comprometer a missão que está a realizar e caso o painel solar não esteja a carregar as baterias), esta rotina é executada para assegurar a navegação de um modo pouco preciso, mas ao mesmo tempo mais poupado.

Com a informação de rumo e velocidade média que estavam a ser praticados pelo veleiro, a rotina de *AutoPilot* compara a informação recebida da bússola com o rumo que era suposto fazer e atua nos lemes para corrigir esse rumo.

Nesta rotina optou-se por não mexer nas velas porque assim era necessário atuar nos servos das velas, que exigem um elevado consumo de corrente perdendo a vantagem da poupança energética e porque, durante navegações oceânicas não vai existir uma mudança repentina do vento, sendo este constante em intensidade e direção durante largos períodos de tempo.

Esta rotina é também chamada quando por algum motivo a comunicação entre o computador principal e a PMU não é estabelecia ou existe algum erro na informação recebida. Caso isso aconteça, a PMU assume o controlo momentâneo da navegação (recorrendo aos últimos dados da navegação em memória) enquanto se reinicia o computador principal.

A rotina de *AutoPilot* é importante para a navegação, pois funciona como um segundo método de controlo do veleiro, com erros ao nível da precisão dos movimentos e com uma velocidade de processamento inferior, mas é uma redundância ao controlo principal do veleiro sempre pronta a assumir o comando, quer seja por pouca bateria ou por erro do computador principal.

- *PowerOff*;

  Quando as baterias estão num estado crítico, que pode comprometer a realização da missão, o controlador entra na rotina de *PowerOff* e força ao encerramento total do veleiro.

  Antes de desligar guarda os dados relativos à navegação que estava a efetuar, para a retomar assim que o estado das baterias o permita. Neste estado, o controlador acorda periodicamente e analisa o estado das baterias e retoma a navegação assim que as mesmas estejam suficientemente carregadas.

No anexo A, apresentam-se os fluxogramas do programa, começando pelo fluxograma geral de todo o programa e de cada rotina principal, onde se percebe a arquitetura geral da solução e das diferentes rotinas já abordadas, assim como a interação entre elas, a sequência lógica da programação e os critérios de decisão, para a transmissão entre cada rotina. O código desenvolvido para esta solução pode ser consultado no Anexo B.

3.3.3 Rotinas Secundárias

Como auxílio ao processamento das rotinas principais foram implementadas algumas rotinas secundárias responsáveis por pequenas funções como por exemplo, cálculo de rumos ou troca de informação.

As rotinas secundárias são então as seguintes:

- *WakeUp* - Rotina responsável por ligar os periféricos associados à PMU (módulo de baterias, GPS, *Beagle Bone Black*, *router WiFi* o rádio controlo e o rádio série). A ordem para ligar os equipamentos e os intervalos de espera entre equipamentos são parametrizáveis e estão definidos na rotina base *Setup*.

- *ShutDown* - Contrariamente à rotina *WakeUp*, a função desta rotina é desligar os equipamentos, uma vez mais com ordem e intervalos parametrizáveis.

- *GetStatus* - Esta rotina é responsável por obter os dados necessários à PMU para cumprir o seu objetivo. Para obter esses dados recorre a duas outras rotinas que lhe estão associadas e que são explicadas em seguida.

- *GetStatusBat* - Esta rotina é parte integrante da *GetStatus* e lê e interpreta o estado das baterias, medindo a tensão aos seus terminais. Para reduzir o erro, efetua a média de três medições, sendo esse o valor que é enviado para ser processado.

- *GetStatusBBB* - Como a anterior, esta rotina é parte integrante da *GetStatus* e estabelece a comunicação e a troca de informação entre a PMU e o *Beagle Bone Black*.

- *Navigation* - Esta rotina é responsável pelos cálculos de navegação necessários para conduzir o veleiro, quando este está em modo de *AutoPilot*.

- *Button* - Sempre que se transita de estado, esta rotina verifica se o relé de *reset* do veleiro foi acionado, de forma a iniciar a rotina de *PowerDown* para encerrar de uma forma segura o veleiro.

Capítulo 4

Solução proposta

Identificadas no capítulo anterior as funções e os requisitos inerentes à PMU e após escolher a plataforma a utilizar no desenvolvimento da PMU, este capítulo começa por apresentar um modelo alternativo ao *Arduino*, que oferece uma poupança significativa na quantidade de energia consumida. Em seguida, explica-se como a PMU vai interagir com os sistemas e equipamentos do veleiro, tendo em conta o código desenvolvido para esta solução e por fim, apresenta-se a autonomia teórica da PMU, com base nos seus modos de funcionamento.

4.1 Standalone Arduino

O *Arduino Uno* é uma plataforma com muitas potencialidades, contudo neste projeto não vamos necessitar de todas as que ele possui. Assim, para cumprir com o segundo requisito operacional de ter um baixo consumo energético, pretende-se depois de ter todos os pormenores definidos, eliminar todos os extras que consumem energia desnecessariamente e construir um *Arduino* apenas com o essencial para o seu funcionamento.

A página oficial do *Arduino* apresenta uma proposta para a construção de um sistema a que chamaremos daqui em diante "Standalone Arduino", que com algum material básico e um microprocessador, consegue recriar uma versão básica de um *Arduino*, que desempenha as mesmas funções, mas com consumos de energia consideravelmente mais baixos. Para a construção do *Standalone Arduino*, sugere-se fazer a montagem da figura 4.1, usando o seguinte material:

- 1 microcontrolador, neste caso o ATMega328P;

- 1 cristal de 16 MHz;
- 1 resistência de 10 kΩ;
- 2 condensadores de 22 pF;
- 1 condensador de 10 μF.

Da montagem que é sugerida na proposta do *Standalone Arduino*, foram feitas duas alterações: foi removido o LED e adicionado um botão de *reset*. Estas alterações foram efetuadas com o objetivo de obter uma maior poupança energética e uma interação mais fácil com o *Standalone Arduino*, ficando a montagem igual à apresentada na figura 4.2.

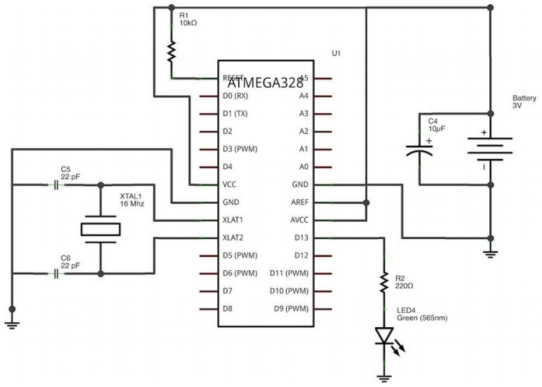


Figura 4.1: Esquema sugerido da montagem do *Standalone Arduino*.

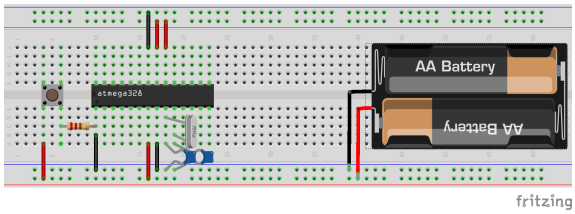


Figura 4.2: Montagem realizada do *Standalone Arduino*, com as referidas alterações.

4.1.1 Testes de consumo

Para se poder quantificar a poupança energética do *Standalone Arduino* face ao *Arduino* original realizou-se um teste de análise de consumos. Durante este teste, o *Arduino*

e o *Standalone Arduino* processaram duas rotinas, enquanto que o consumo de corrente associado ao seu processamento era medido.

As rotinas utilizadas foram a *BareMinimun* e a *Blink*. A rotina *BareMinimun* corresponde ao *bootloader* que vem pré-carregado nos microprocessadores e na qual está inserido as rotinas base de *Setup* e de *Loop*. Já a rotina *Blink* é uma função básica do *Arduino*, que acende e desliga o LED da placa com um intervalo de tempo fixo.

Outro aspeto que foi alterado na realização dos testes, foi a tensão de alimentação. De acordo com os dados da tabela 3.1 do capítulo 3, o *Arduino UNO* pode ser alimentado com uma tensão compreendida entre os 7 V e os 12 V. Este pode no entanto ser alimentado com tensões inferiores, não garantindo tensões de 5 V nas saídas digitais. Como os testes realizados pretendem analisar o consumo apenas ao nível do processamento de dados, os valores de tensão utilizados foram os 3 V e os 5 V. Estes valores foram escolhidos de acordo com o intervalo de alimentações do microprocessador ATMega328P, onde foram realizados os mesmos teste, garantindo assim que não existe incoerência nos valores da alimentação.

Os resultados obtidos nos testes de consumo estão representados na figura 4.3.

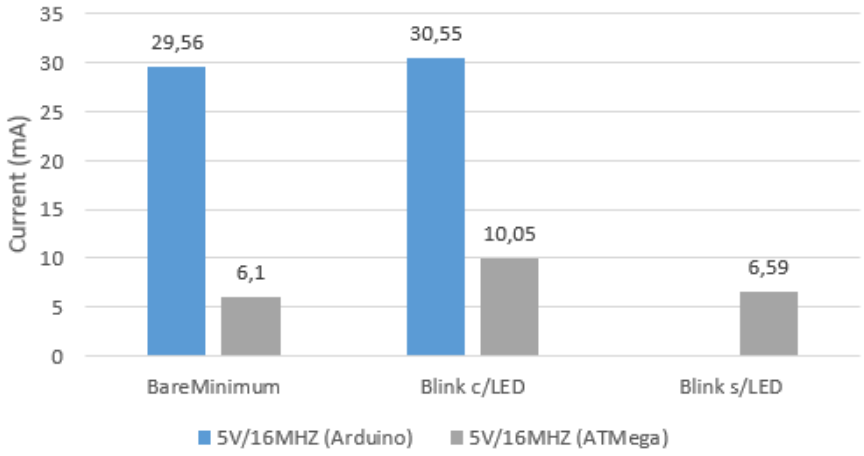


Figura 4.3: Correntes consumidas (mA) pelo *Arduino* e pelo *Standalone Arduino*.

Pela análise dos resultados obtidos concluímos que a poupança energética é bastante eficaz, conseguindo obter-se uma redução na ordem dos 80 %. Pode-se também concluir que a poupança se deve sobretudo à não existência de periféricos no *Standalone Arduino*,

pois no teste *BareMinimun* usando o *Arduino* verificou-se valores de consumo de corrente muito elevados, devido aos periféricos que são alimentados e não estão a ser utilizados.

Identificado o *Standalone Arduino* como a solução que apresentava maior eficiência energética e aliados aos modos de *sleep* implementados na programação da PMU, o requisito de ter uma solução com um baixo consumo energético pode ser alcançada. Para tal projetou-se um protótipo, com base no esquema do *Standalone Arduino*, para a construção da PMU.

4.2 Interação PMU-FASt

Para construir o protótipo da PMU, de forma a que existisse uma correta interação com sistema de controlo do veleiro, foi necessário recorrer a protocolos de comunicação, para facilitar a troca de dados, bem como o uso de integrados para poder interagir diretamente com outros equipamentos de bordo. Na figura 4.4 está representado o esquema geral da interação PMU-FASt, que em seguida é explicada com mais pormenor abordando as áreas mais importantes.

4.2.1 Ligar e desligar equipamentos

Para a PMU controlar o arranque em segurança do veleiro, este tem que ser capaz de ligar, no mínimo seis equipamentos principais de bordo, sendo eles: o GPS, o rádio série, o radio controlo, o *router wifi*, o computador principal e o módulo de energia. A ideia original foi ter cada equipamento associado a um relé, com a PMU a ligar e a desligar cada equipamento enviando um sinal para esse relé.

4.2.1.1 O relé

Ao definir os contornos da PMU, um dos principais aspetos a ter em conta foi o tipo de relés a utilizar para controlar os equipamentos, uma vez que o funcionamento normal dos relés obriga a corrente a percorrer a bobine do relé para que este mantenha o seu estado, isso leva a que o consumo de energia seja muito elevado.

Para evitar esse consumo excessivo usou-se um relé *latch* com dois enrolamentos, modelo G6EK-134P-US, da OMRON Corporation. Este relé permite poupar a energia do sistema, pois só é necessário um impulso para comutar o estado do relé e quando o

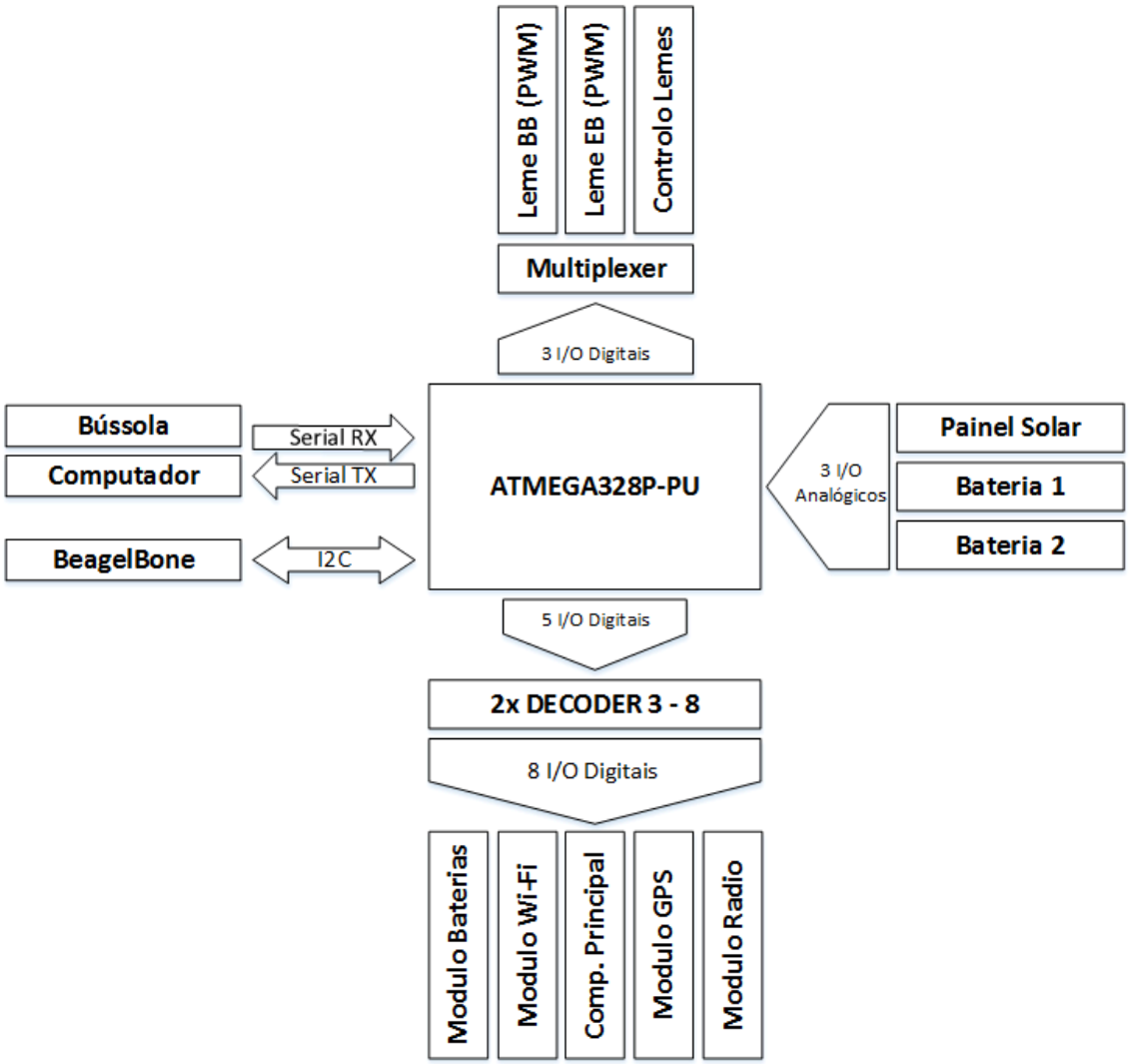


Figura 4.4: Esquema geral das ligações da PMU.

impulso é interrompido, o estado mantém-se recorrendo a meios mecânicos ou eletromagnéticos.

A única desvantagem deste relé é o facto de ser necessário induzir um impulso noutro enrolamento para fazer com que o relé volte ao seu estado original, o que leva à necessidade de usar dois sinais de controlo para cada relé.

Segundo os dados da figura 4.5, retirada da *datasheet* do relé G6EK-134P-US, verifica-se que para uma alimentação de 5 V, um relé com sensibilidade *standard*, consome cerca de 40 mA, para alimentar as bobines. Estes dados são importantes para o cálculo da

Classification	Rated voltage	Rated current (mA)		Coil resistance (Ω)		Must set voltage (V)	Must reset voltage (V)	Max. voltage (V)	Power consumption	
		Set coil	Reset coil	Set coil	Reset coil	% of rated voltage			Set coil (mW)	Reset coil (mW)
Standard	5 VDC	40.0	40.0	125	125	70% max.	70% max.	190% (at 23°C)	Approx. 200	Approx. 200
	6 VDC	33.3	33.3	180	180					
	9 VDC	22.2	22.2	405	405					
	12 VDC	16.7	16.7	720	720					
	24 VDC	8.3	8.3	2,880	2,880					
Low-sensitivity	5 VDC	79.4	79.4	63	63	70% max.	70% max.	170% (at 23°C)	Approx. 400	Approx. 400
	6 VDC	66.6	66.6	90	90					
	9 VDC	44.3	44.3	203	203					
	12 VDC	33.3	33.3	360	360					
	24 VDC	16.7	16.7	1,440	1,440					

Note 1. The rated current and coil resistance are measured at a coil temperature of 23°C with a tolerance of ±10%.
2. Operating characteristics are measured at a coil temperature of 23°C.
3. The maximum voltage is the highest voltage that can be imposed on the relay coil.
4. Refer to the engineering data for relations between the ambient temperature and maximum coil voltage.

Figura 4.5: Dados do relé G6EK-134P-US.

autonomia da PMU.

4.2.1.2 O *decoder*

A utilização de um relé com dois enrolamentos obriga a que sejam utilizados dois sinais para controlar o funcionamento do equipamento, ou seja, são necessários dois pinos por equipamento, um para ligar e outro par desligar. Devido ao número limitado de pinos disponíveis no controlador, foi necessário usar dois *decoders*, um utilizado para ligar os relés e o outro para os desligar. Esta solução adotada a PMU pode controlar no máximo oito equipamentos, recorrendo apenas de cinco saídas digitais do controlador.

O *decoder* utilizado foi o 74HCT138, da NXP Semiconductors, que é um descodificador binário 3-8.

Assim sendo, tem-se três saídas digitais do controlador ligadas às entradas A0, A1 e A2, dos dois *decoders* e duas saídas digitais do controlador ligadas ao *enable* E3 de cada *decoder*. O sinal dos *enable* é que vai determinar se é para ligar ou desligar os equipamentos. As saídas dos *decoder's*, do Y0 ao Y7, estão ligadas a transístores BC547 que atacam os terminais dos enrolamentos dos relés. Na figura 4.6 podem observar-se um esquema das ligações anteriormente referidas.

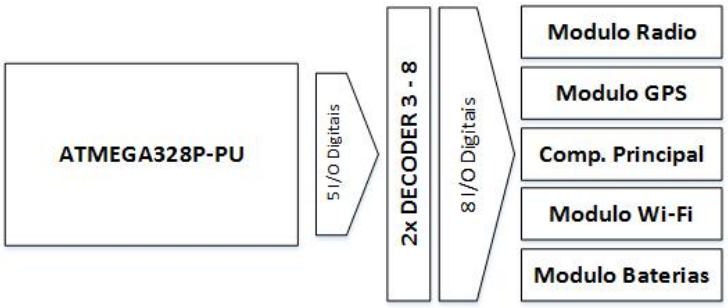


Figura 4.6: Esquema das ligações entre o controlador e os equipamentos.

4.2.2 Comunicação com o *BeagleBone Black*

Na ideia original, a comunicação entre a PMU e o computador principal (ver figura 4.7) seria feita via protocolo SPI. SPI é a sigla de *Serial Peripheral Interface*, isto é um protocolo que permite a comunicação do microcontrolador com diversos componentes, formando uma rede. Neste protocolo existe um elemento *master*, que controla toda a comunicação e os restantes elementos da rede são *slave*. Para se estabelecer esta ligação são precisos quatro canais, um de sincronização através de um sinal de *clock*, um de seleção do *slave* com o qual o *master* quer comunicar e outros dois para a troca de dados entre eles, sendo que nestes dois os dados só passam num sentido. Esses canais são, respetivamente, o SCL (*Serial Clock*), SS (*Slave Select*), MISO (*Master In Slave Out*) e o MOSI (*Master Out Slave In*).

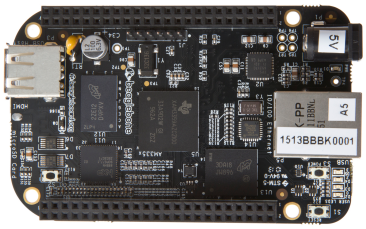


Figura 4.7: *BeagleBone Black.*

Apesar do controlador já vir preparado para fazer este tipo de comunicação, o elevado número de saídas necessário e a incompatibilidade de usar a comunicação via SPI com outras bibliotecas usadas na programação da PMU, obrigou a procurar uma alternativa para comunicar entre entre a PMU e o computador principal.

O protocolo de comunicação utilizado foi o I2C (*Inter-Integrated Circuit*). Este protocolo foi criado pela Philips, e funciona baseado numa relação *master-slave*. No caso da PMU, a PMU atua como *master* da ligação e o computador principal como *slave*, enviando dados relativos à navegação quando solicitado pela PMU. Para este protocolo são apenas necessários dois canais de comunicação, o SDA (*Serial Data*) e SCL (*Serial Clock*).

4.2.3 Comunicação com a Bússola

Como o controlador só possui uma *Serial Port* e esta estava a ser utilizada para a PMU transmitir ao utilizador dados sobre o estado do seu funcionamento, a comunicação com a bússola seria realizada através de uma *Serial Port* virtual, com recurso a uma biblioteca que cria em quaisquer dois pinos do controlador, um canal de comunicação série. Contudo, as bibliotecas utilizadas apresentavam erros de compatibilidade com a *Serial Port* que comunica com o computador.

A solução adotada foi usar só a *Serial Port* do controlador, sendo o canal de receção usado para a PMU receber os dados da bússola. No canal de transmissão a PMU disponibiliza para o utilizador informações sobre o seu estado de funcionamento. Esta solução só foi possível, uma vez que a bússola (ver figura 4.8) transmite os dados com um fluxo contínuo, não sendo necessário à PMU requisitar nenhum dado à bússola.

A solução adotada para as comunicações da PMU encontra-se representada na figura 4.9.

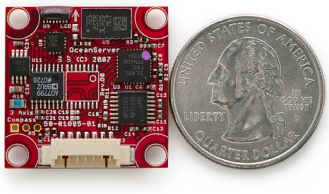


Figura 4.8: Bússola digital OS5000.

4.2.4 Nível das baterias

Como já foi referido nos requisitos do sistema, o controlador deveria possuir para além de pinos digitais, pinos analógicos.

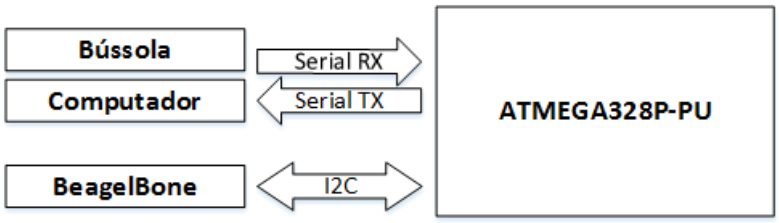


Figura 4.9: Esquema das ligações entre o controlador e o computador principal e a bússola.

As entradas analógicas tem associadas um conversor analógico-digital de 10 bits, admitindo uma gama de tensões de entrada entre 0 V e 5 V, que transforma a voltagem que lhe chega num valor digital entre 0 bits e 1023 bits.

Os pinos analógicos são necessários para fazer a medição da tensão das baterias e do painel solar e com esses valores calcular a percentagem de bateria restante e determinar se o painel solar está a carregar as baterias. Essa interação pode ser observada na figura 4.10.

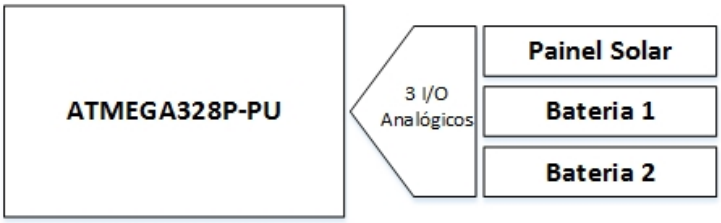


Figura 4.10: Esquema das ligações entre o controlador e as baterias e o painel solar.

Apesar do módulo de potência, onde estão inseridas as baterias, enviar via *Serial Port* o nível de carga das mesmas, era necessário saber o nível de carga por outra forma, uma vez que a *Serial Port* já estava a ser utilizada.

Sabendo que os pinos analógicos só podem receber até 5 V, dimensionou-se um divisor de tensão para as entradas das duas baterias no controlador e, de acordo com os dados da figura 4.11, retirada da *datasheet* das baterias, foi possível definir um método mais expedito para determinar a carga da bateria. Com esta informação a PMU define em que modo de poupança de bateria se deve utilizar e, por conseguinte, a rotina que deve ser executada.

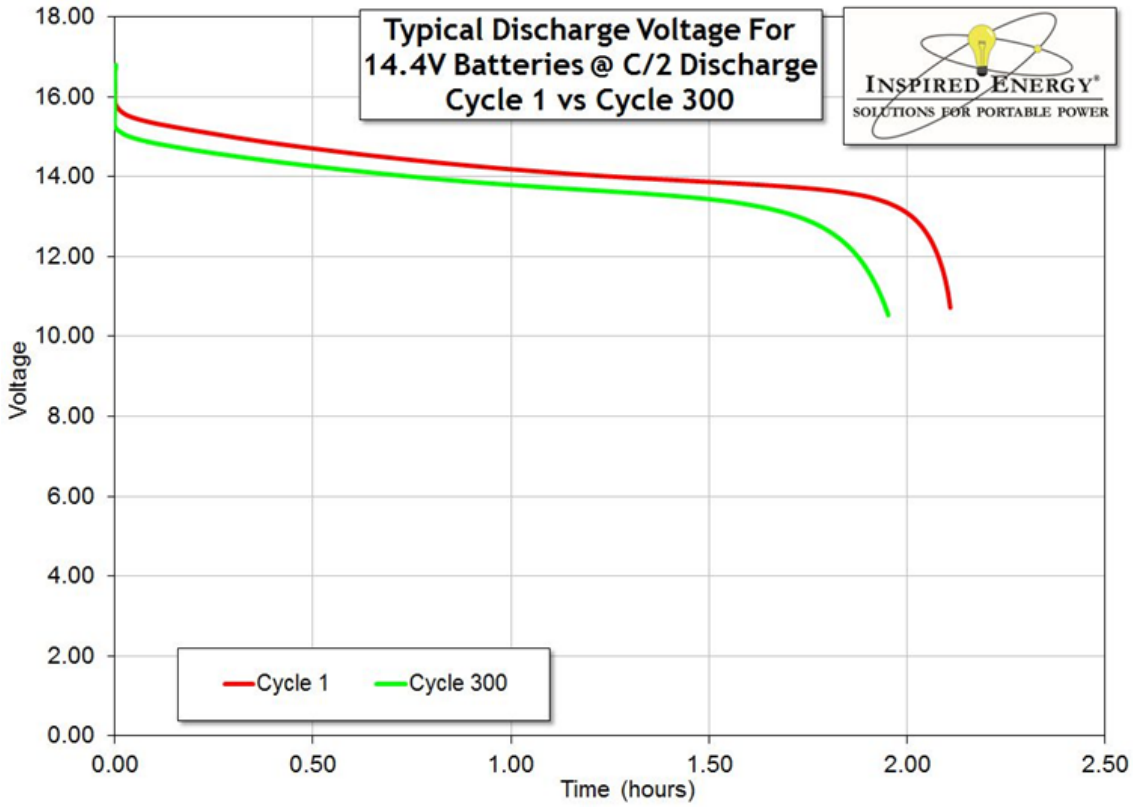


Figura 4.11: Variação da voltagem das baterias em função do tempo, consoante os ciclos de descarga.

4.2.5 Controlo dos lemes

Assim como os pinos analógicos, foi requisito operacional que o controlador tivesse pinos do tipo PWM. Usaram-se as saídas PWM para gerar os sinais de comando dos servo-motores dos lemes do veleiro na rotina de *AutoPilot*.

Contudo, no seu modo normal de funcionamento, os lemes estão sobre o comando do computador principal do veleiro. Foi necessário arranjar uma forma de poder comutar o controlo dos lemes entre o computador principal e a PMU, quando esta passa para modo *AutoPilot*. Para se fazer isso usou-se o *multiplexer* DM74HCT157, (figura 4.12), que com um sinal da PMU, o controlo dos lemes transita do computador principal para a mesma e vice-versa.

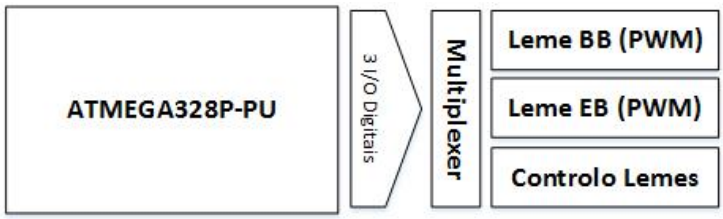


Figura 4.12: Esquema das ligações entre o controlador e os lemes.

4.3 Autonomia da PMU

Antes de avançar dados sobre a autonomia expectável da PMU, é importante perceber que o modo de funcionamento desta depende quase exclusivamente do estado das baterias e por isso avançar com uma estimativa da autonomia do protótipo é um processo pouco preciso e sujeito a muitos erros.

Apesar da solução adotada, o *Standalone Arduino* representar por si só uma considerável poupança de energia. Para atingirmos valores que permitam uma grande autonomia, é necessário recorrer aos modos de *sleep* na programação. Segundo um artigo publicado no página do EngBlaze [32], o modo de *sleep Ilde Mode* consegue reduzir os consumos do *Arduino* para um terço, passando de 15 mA para os 5 mA. Este artigo refere ainda que esta poupança pode atingir valores na ordem dos microamperes, caso se utilize os modos de *sleep* mais profundos. Todavia, estes dados são relativos à utilização destas rotinas num *Arduino*, enquanto que nesta situação os modos de *sleep* são usados no *Standalone Arduino*, onde não existe nenhum estudo de consumo.

Assim sendo, com a informação disponível é expectável obter valores na ordem dos microamperes, quando estivermos nas rotinas de *ShutDown* e de *PowerDown*, uma vez que ambas utilizam os modos mais profundos de *sleep* que o microcontrolador dispõe. Na rotina de *WatchDog* também são expectáveis estes valores, porém quando a PMU acordar para trocar dados com o computador principal e para verificar o estado das baterias, o seu consumo vai aumentar, sendo de esperar que os valores rondem os miliamperes. Algo de que se pode ter a certeza é que a rotina de *AutoPilot* é a que vai apresentar maiores valores de consumo, uma vez que esta rotina está sempre a processar dados da navegação, não havendo um correto aproveitamento das rotinas de *sleep*.

Considerando um cenário hipotético em que os consumos nos diferentes modos estão de acordo com o que foi descrito anteriormente, e cujo os valores considerados são os

apresentados na primeira coluna da tabela 4.1, e sabendo que o protótipo será alimentado por pilhas alcalinas, com uma tensão de 1,5 V e uma capacidade de 2,1 Wh para uma corrente de 1400 mA, a autonomia do protótipo para cada modo pode ser observada na segunda coluna da já referida tabela.

Tabela 4.1: Autonomia prevista para o protótipo

Modo de Funcionamento	Consumo (mA)	Autonomia - Pilhas AA (dias)
PowerDown	0,3	195
WatchDog	0,6	97
AutoPilot	2	30
ShutDown	0,1	195

Neste cenário hipotético, tem-se uma autonomia de meio ano em modo *PowerDown*, três meses em modo *WatchDog*, cerca de trinta dias em modo *AutoPilot* e uma autonomia superior a um ano em modo *ShutDown*. Estes dados, apesar de meramente hipotéticos, são um bom indício para a autonomia do protótipo da PMU, contudo é necessário efetuar testes para quantificar os consumos e calcular com maior precisão a autonomia da solução criada.

Capítulo 5

Testes realizados

Neste capítulo estão descritos os testes realizados ao protótipo criado. No início do projeto foi definida a realização de dois tipos de testes ao protótipo, sendo os primeiros realizados num ambiente simulado, onde fosse possível controlar todas as variáveis e os segundos seriam realizados em ambiente real com a placa já integrada no sistema de controlo do veleiro. Contudo, por motivos logísticos, não foi possível a realização dos testes em ambiente real, como havia sido definido.

Os testes realizados em ambiente simulado foram agrupados em dois grupos. O primeiro grupo de testes foi realizado com recurso ao *Arduino Uno R3* e com os componentes isolados, ou seja, foi testado simplesmente a parte de código específica da interação entre o controlador e cada equipamento.

Com as conclusões dos testes isolados desenhou-se e construiu-se o protótipo da PMU, que foi submetido aos mesmos testes e a um teste global. Com estes testes concluídos, correu-se uma simulação, onde foi registado o consumo de corrente nas diversas fases da simulação.

5.1 Testes às rotinas

5.1.1 Testes das baterias e painel solar

O primeiro teste realizado foi à rotina de medir as tensões da baterias e do painel solar. Para simular a descarga das baterias e a diferença da tensão do painel solar usaram-se três

potenciómetros que permitiam controlar a tensão aos terminais entre os 0 V e os 5 V. A figura 5.1 representa a montagem efetuada para realizar este teste.

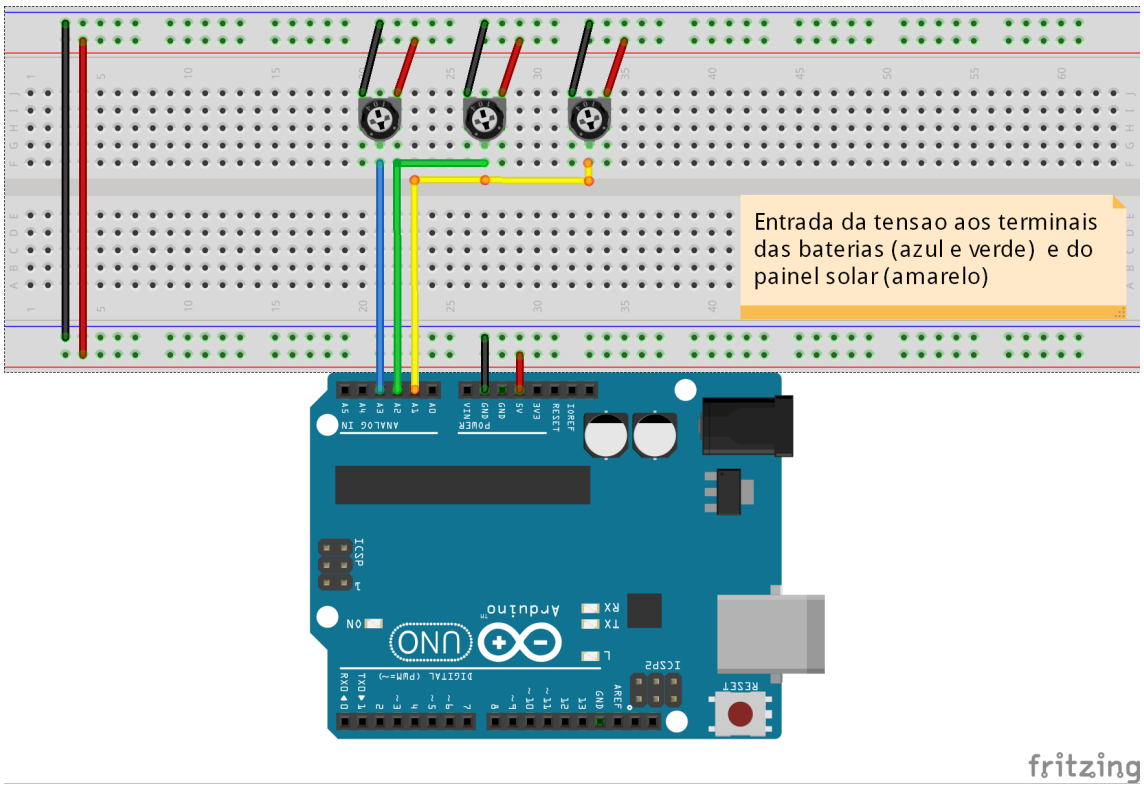


Figura 5.1: Esquemas do teste da rotina das baterias e do painel solar.

Tabela 5.1: Valores da voltagem das baterias apresentados pelo *Arduino*

%	Valores Reais (V)	Valores Arduino (V)
100	5	5
75	3.75	3,72
60	3	2,98
50	2,5	2,5
35	1.75	1,72
20	1	0,99
0	0	0

Este teste foi bastante útil, pois só assim foi possível detetar um erro na rotina que calculava a percentagem de bateria. Depois do erro corrigido, os valores apresentados pelo

Arduino e os valores reais da tensão aos terminais dos potenciómetros eram aceitáveis, sendo o erro máximo de 0,03 V, como se pode verificar na tabela 5.1.

5.1.2 Testes dos relés

O segundo teste efetuado foi com os relés *latch* e os *decoders*. Para simular os equipamentos a ligar e a desligar foram colocados dois LED's nos terminais dos relés, como se verifica na montagem da figura 5.2. Entre o *Arduino* e os relés foram usados dois *decoders* 74HCT138 para codificar os sinais, com recurso a apenas cinco sinais digitais vindos do *Arduino*. É ainda de salientar que entre o *decoder* e o relé existia um transístor, cuja função era amplificar os sinais provenientes do *decoder* para o relé.

O teste realizado foi bastante conclusivo, uma vez que permitiu verificar o correto funcionamento das rotinas *PowerOn* e *PowerOff*. Para além de cumprir com o objetivo principal, este teste permitiu ainda identificar a necessidade de se colocar um transístor a amplificar o sinal antes relé. Foi alterado apenas um pormenor na atribuição do índice correspondente a cada equipamento, usando o "0"como primeiro índice a codificar e não o "1", evitando assim uma alteração do estado dos *bits* codificadores. A codificação final de cada equipamento corresponde à apresentada na tabela 5.2.

Tabela 5.2: Codificação dos equipamentos com recurso ao *decoder* 74HCT138

Equipamento	**Saída Decoder**	**Entradas Decoder (A2 A1 A0)**
Módulo Baterias	Y0	0 0 0
Módulo Wi-Fi	Y1	0 0 1
Computador Principal	Y2	0 1 0
GPS	Y3	0 1 1
Rádio Série Rádio Controlo	Y4	1 0 0

5.1.3 Testes das comunicações

Os protocolos de comunicação foram testados com recurso a equipamentos que não correspondem aos equipamentos com os quais a PMU vai comunicar quando estiver integrada no veleiro, isto porque em simultâneo com este projeto, o veleiro sofreu um *upgrade*

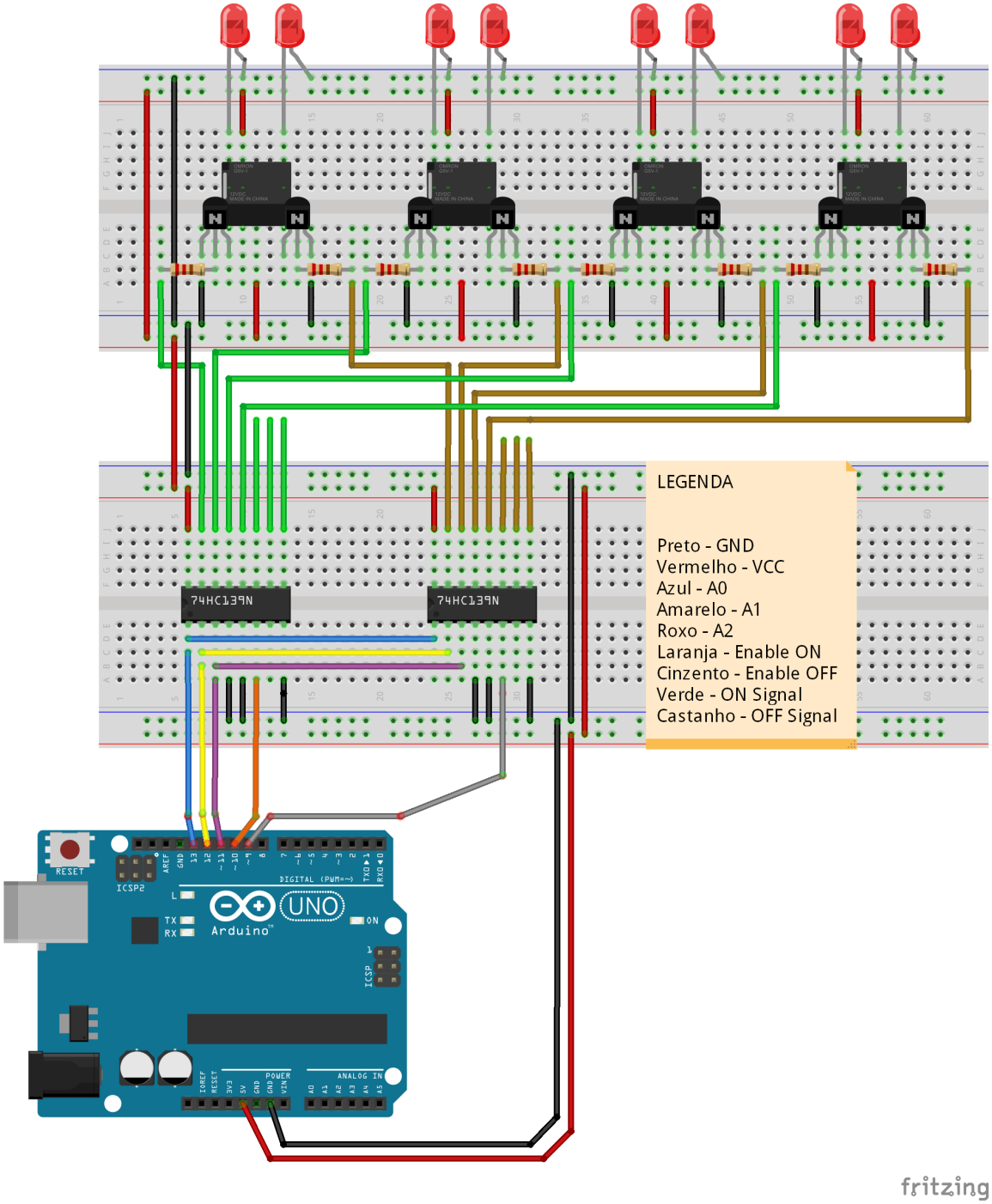


Figura 5.2: Testes dos relés *latch*.

no computador principal, substituindo o anterior pelo *Beagle Bone Black*. Isso impossibilitou a realização dos testes com os equipamentos do veleiro, sendo a bússola digital OS5000, que equipa o veleiro, substituída por outra bússola digital, modelo OS4000 e o computador principal foi substituído por um *Arduino UNO R3*. Apesar dos equipamentos não serem os equipamentos originais, não houve qualquer limitação ao teste realizado.

A bússola OS4000 corresponde ao modelo anterior da bússola que equipa o veleiro, contudo o protocolo de comunicação, o formato dos dados e a taxa de transferência de dados é a mesma. A diferença entre a OS4000 e a OS5000 são aspetos de precisão e de aquisição de outros dados não essenciais para a navegação e que em nada influenciam o resultado destes testes.

O computador principal foi substituído por outro *Arduino* programado para responder, segundo o mesmo protocolo de comunicação utilizado entre o computador principal e a PMU, fazendo a troca dos dados de navegação.

Só após a realização destes testes é que se verificou a incompatibilidade de bibliotecas utilizadas para o controlo dos lemes, com a *Serial Port* virtual criada para comunicar com a bússola. Como já foi referido no capítulo 3, a opção adotada foi usar a *Serial Port* que o controlador disponibiliza para fazer a troca de dados com a bússola, como se verifica na figura 5.3. Quanto ao teste da comunicação entre a PMU e o *Arduino*, os resultados obtidos foram os esperados, havendo o envio do rumo e da velocidade média do veleiro, do computador principal para a PMU, obedecendo ao seguinte formato "vv.v,rrr.r", onde o "v"corresponde ao valor da velocidade média, em nós e o "r"ao valor do rumo, em graus.

5.1.4 Testes dos lemes

O teste do controlo dos lemes foi realizado usando o *multiplexer* 74HCT157 e dois LEDs, com se pode observar na figura 5.4. Neste teste, os pinos 10 e 11 do *Arduino*, ambos codificados internamente para PWM, enviavam dados para o integrado que tem como função variar o brilho do respetivo LED (pino 10 controla o brilho do LED da esquerda e o pino 11 o brilho do LED da direita). Caso o pino 12 bloqueasse a saída, tal correspondia à passagem do comando para o computador principal e os LEDs permaneciam desligados.

Os resultados deste teste foram os esperados, com o brilho do LED a corresponder com a informação enviada pelo *Arduino*. Contudo, para se certificar que o integrado estava a funcionar em condições, utilizou-se um segundo *Arduino* para enviar informação

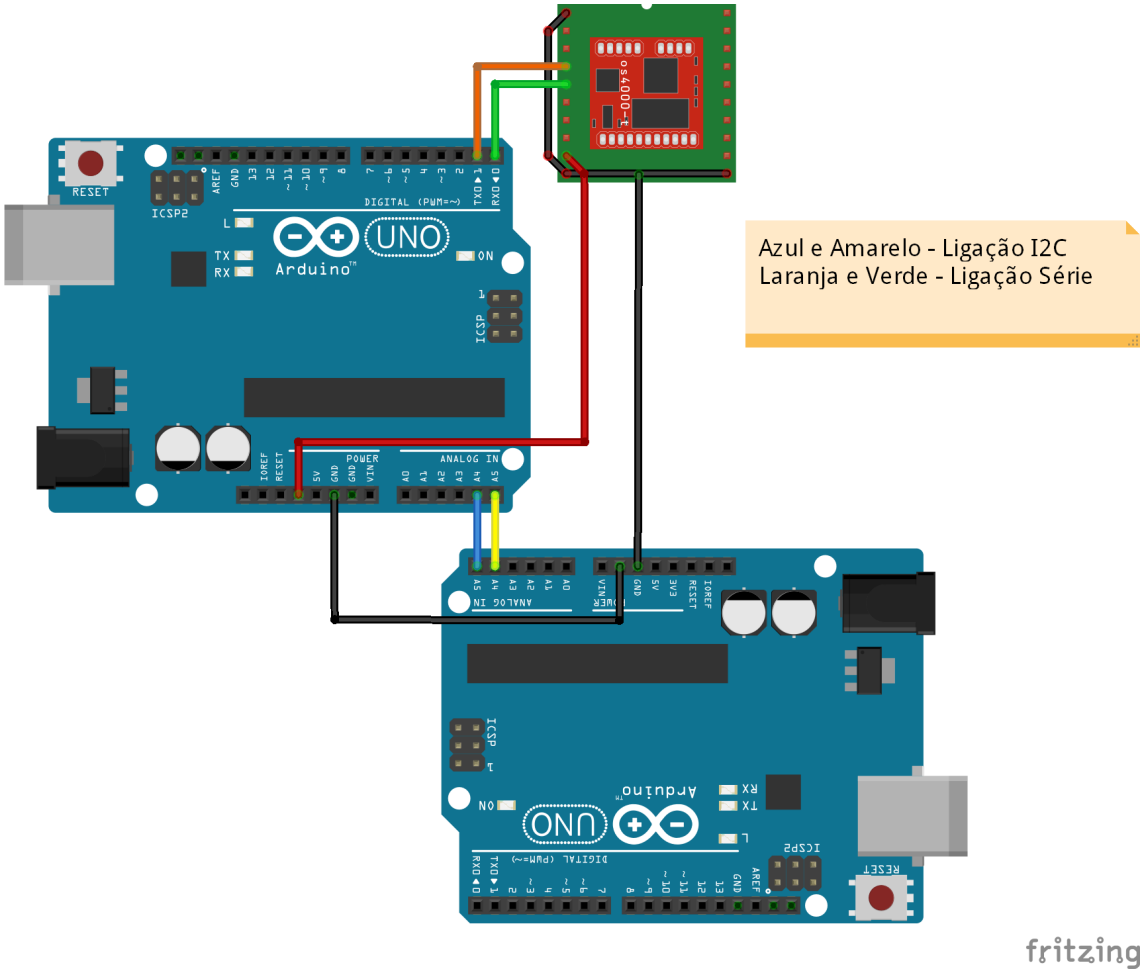


Figura 5.3: Testes das comunicações com a bússola e o computador principal.

diferente, garantindo assim a correta comutação do controlo dos lemes entre a PMU e o computador principal.

5.2 Teste global

Com os resultados obtidos e as lições aprendidas dos testes isolados de cada equipamento, criou-se uma placa que deu origem ao protótipo de PMU. Com recurso ao Proteus[1] fez-se o desenho da placa de modo a otimizar o espaço das ligações entre os integrados, de modo a reduzir o tamanho da mesma. Na figura 5.5 temos a virtualização em três

[1] *Software* de desenho de placas de circuito impresso.

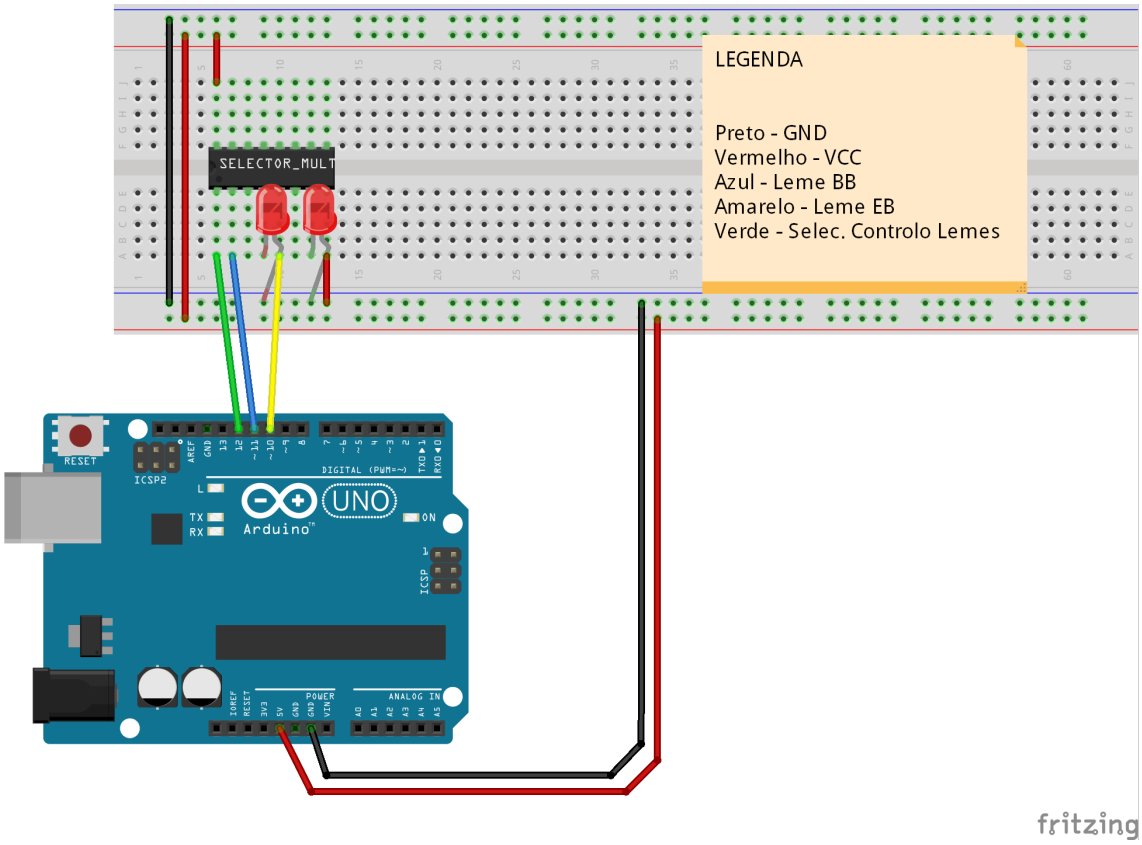


Figura 5.4: Testes do controlo dos lemes.

dimensões da placa criada e na figura 5.6 temos representado o resultado final da placa, o protótipo da PMU.

Depois de concluído o protótipo da PMU, este foi submetido a um teste final (ver figura 5.7) que consistiu em fazer uma simulação onde os parâmetros eram passíveis de ser alterados para fazer uma análise dos consumos de corrente. Devido às limitações de equipamentos já aqui expostas, a simulação foi realizada com um *Arduino UNO R3*, a substituir o computador principal do veleiro, a bússola OS4000 substituiu a bússola do veleiro, a OS5000, e os servomotores dos lemes do veleiro foram substituídos por outros de menor dimensão. É importante referir que todas estas substituições não condicionaram o resultado dos testes, uma vez que o objetivo foi quantificar o consumo do protótipo.

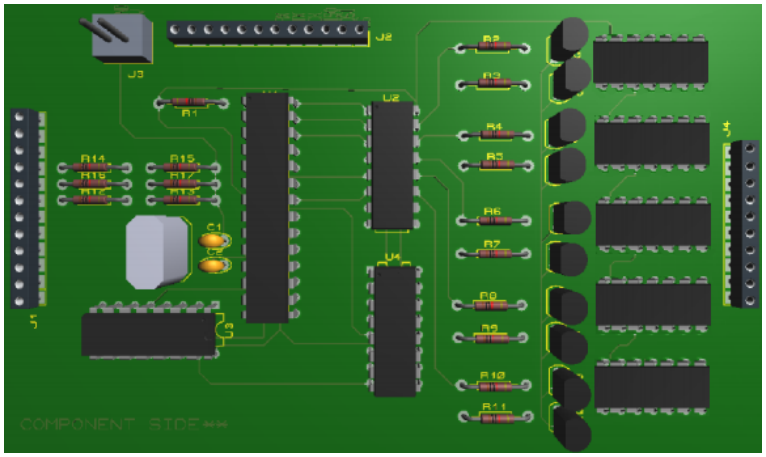


Figura 5.5: Placa de circuito impresso da PMU.

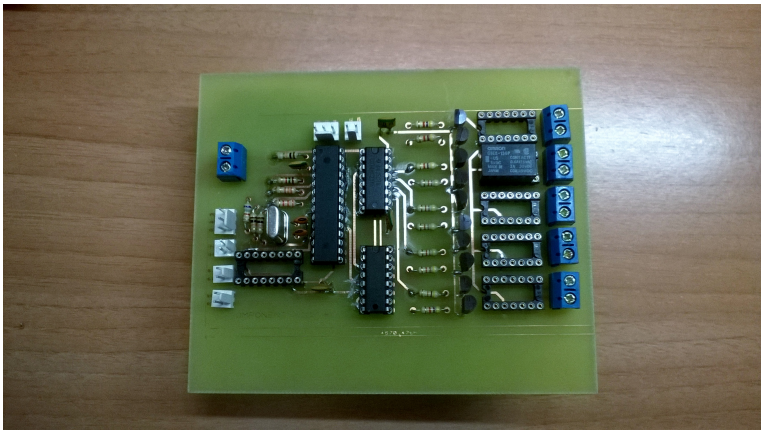


Figura 5.6: Protótipo da PMU.

5.2.1 Resultados obtidos

Os dados do consumo de corrente retirados da simulação foram muitos bons, superando até as expectativas de consumo. Para cada um dos quatro estados de funcionamento da PMU existe um regime de consumo diferente associado, sendo que o consumo depende de vários fatores, mas são condicionados principalmente pelas rotinas de *sleep* aplicadas no decorrer de cada rotina principal. Os consumos médios de cada rotina são apresentados na tabela 5.3.

Como seria de esperar, a rotina com um consumo menor é a *ShutDown*, pois nesta rotina é aplicado o modo de *sleep* mais profundo, o modo *Power Down*, o que permite ter

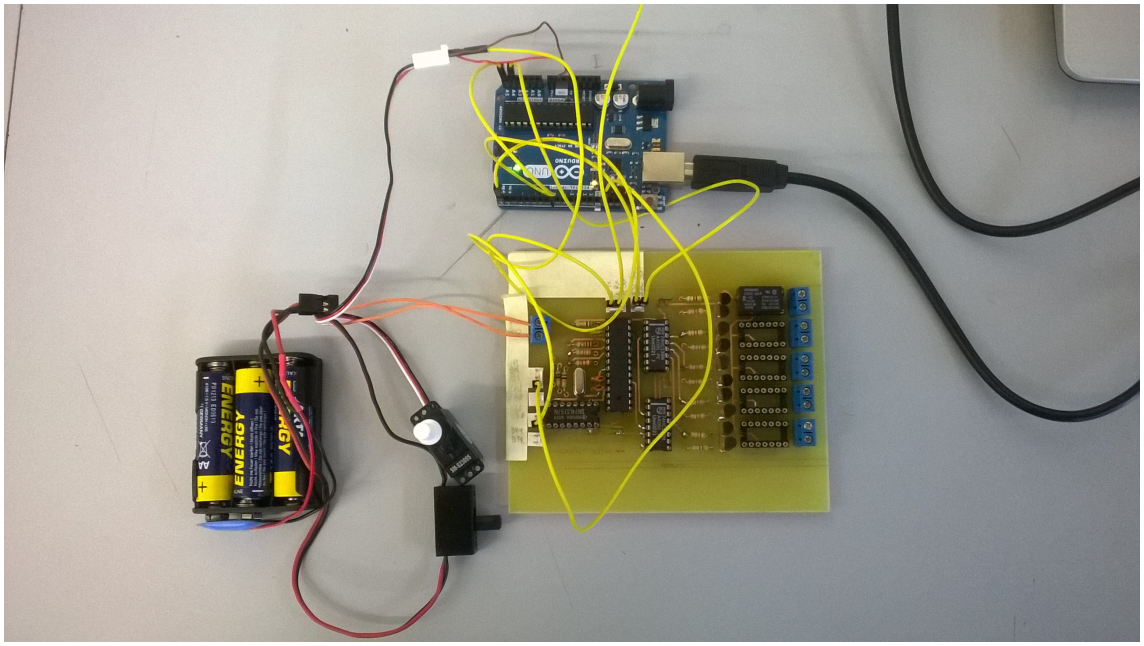


Figura 5.7: A PMU a correr uma simulação.

Tabela 5.3: Consumos de corrente da PMU

Rotina	Consumo Médio (mA)	Consumo Máximo (mA)
PowerDown	0,270	0,270
WatchDog	0,024	1,6
AutoPilot	1,6	1,8
ShutDown	0,024	0,024

consumos tão baixos.

O mesmo consumo apresenta a rotina de *WatchDog*, mas esporadicamente o consumo aumenta para 1,6 mA quando existe troca de dados com computador principal. Contudo, esse aumento não influencia o valor médio do consumo da PMU quando está no modo *sleep*, e tal deve-se à baixa frequência como que a PMU efetua a troca de dados com o computador principal.

A rotina de *AutoPilot* regista o maior consumo de corrente, pois neste modo a PMU está sempre ativa e a processar dados do controlo da navegação. O modo *PowerDown* tem um consumo de 0,270 mA, porque neste modo não pode ser aplicado o modo *sleep* que permite uma maior poupança, pois a PMU tem que detetar uma interrupção externa para arrancar o veleiro.

De registar ainda que os maiores picos de corrente ocorrem quando a PMU liga ou desliga um equipamento e isto deve-se aos 50 mA que têm que ser induzidos nas bobines dos relés para os fazer atracar ou desatracar.

Como se constata, os modos *sleep* são bastantes eficazes e a aproximação efetuada no caso hipotético, não era totalmente descabida. Os valores estimados estavam enquadrados com a realidade à exceção dos 0,1 mA no modo de *ShutDown*, em que o valor medido foi mais baixo e a principal alteração foi no modo *WatchDog*, o qual na aproximação efetuada considerou-se um valor bastante elevado tendo em conta a troca de dados que a PMU vai fazer com o computador, quando na verdade essa troca de dados não altera significativamente a média do consumo.

5.3 Autonomia da PMU e do veleiro

5.3.1 PMU

Após medidos os consumos de corrente das diferentes fases de funcionamento da PMU, era importante ter dados de autonomia das diferentes rotinas principais e posteriormente num caso real.

Como era requisito do sistema, a alimentação da PMU é diferente da alimentação do computador principal, recorrendo para tal a uma fonte externa que foram as pilhas alcalinas. Para este caso foram realizados dois cálculos de autonomia com diferentes pilhas, sendo que num primeiro cálculo foram utilizadas as pilhas alcalinas tipo AA e num segundo cálculo, as pilhas alcalinas tipo D. As *datasheets* de ambas as pilhas podem ser consultadas no anexo C.

Com os dados da capacidade de cada pilha alcalina, e sabendo que a PMU é alimentada com uma tensão de 4.5 V, calculou-se a autonomia para cada rotina principal. Os dados da tabela 5.4 representam os resultados alcançados.

Como seria de esperar, dado os valores do consumo de corrente medidos, a autonomia da PMU é muito elevada. Aplicar estes resultados a um cenário de navegação de longo curso para estimar o consumo diário da PMU não é algo linear, pois o modo de funcionamento da PMU está intrinsecamente relacionado com o estado das baterias do FASt. Porém, considerando o cenário mais pessimista, que seria navegar com apenas 8 h de sol durante o dia, com a PMU em modo *WatchDog*, e durante noite em modo Auto-Pilot, a PMU consumia aproximadamente 118 mW, ou seja, podia estar a navegar nestas

Tabela 5.4: Autonomia do protótipo da PMU

Modo de Funcionamento	Consumo (mA)	Autonomia - Pilhas AA (dias)	Autonomia - Pilhas D (dias)
PowerDown	0,27	386	3086
WatchDog	0,024	4340	34722
AutoPilot	1,6	65	521
ShutDown	0,024	4340	34722

condições durante 3 meses (pilhas AA) ou 2 anos (pilhas D). É de salientar que nestes cálculos não foram contabilizados os consumos dos servomotores dos lemes e da bússola, essenciais para a rotina de *AutoPilot*, uma vez que estes são alimentados pelas baterias do veleiro.

5.3.2 FASt

Como esta provado em teoria, a PMU é uma solução viável a utilizar no FASt, uma vez que aumenta a sua robustez e a sua capacidade de navegação de longo curso em navegação autónoma. Para explorar melhor as potencialidades desta solução era importante ter dados da autonomia e do consumo energético do FASt, contudo esses dados nunca foram medidos devido a falta de oportunidade. Todavia, podem-se avançar alguns pressupostos para a autonomia das baterias do FASt:

- O sistema eletrónico do mesmo, incluindo GPS e bússola, tem um consumo inferior a 2 W; [33]
- A maior percentagem do consumo deve-se à atuação dos servomecanismos da vela e dos lemes;
- Considerando que numa navegação de longo curso a intensidade e direção do vento será constante durante longos períodos de tempo, a posição das velas vai ser corrigida esporadicamente, o que reduz o consumo de energia para valores insignificantes;
- A atuação dos lemes é o que consome mais energia em todo o sistema do veleiro, uma vez que a frequência com que os lemes são atuados está associada à precisão da navegação que se pretende. Numa navegação de longo curso não é necessária

tanta precisão, o que permite reduzir a frequência de atuação dos lemes e com isso reduzir o consumo de energia;

- Em navegação noturna, por questões de segurança da navegação, todas as embarcações tem que ter as luzes de navegação acesas. O FASt está equipado com três LEDs de alto brilho, que representa um considerável aumento no consumo de energia;

- O painel solar que equipa o veleiro tem um potência de pico de 45 W. Contudo, segundo recomendações dos fabricantes, deve-se considerar que a média diária da potência produzida corresponde a 10 % da potência de pico. Assim, podemos contar com cerca de 4,5 W de potência média ao longo de 24 h.

Posto isto, pode-se concluir que a autonomia das baterias do veleiro é algo muito volátil, pois existem muitos fatores que condicionam o consumo médio diário. Segundo Alves, J., Ramos, T. e Cruz, N. [33] o sistema eletrónico inicial gasta 1,65 W e que isso representa, segundo as estimativas iniciais, mais de 50 % do consumo total, o que quer dizer que o consumo total médio será inferior a 3,3 W. Como as baterias têm uma capacidade total de 190 Wh, admitindo que se pode gastar até 80 % dessa energia teríamos autonomia para 46 h ou 1,9 dias. No entanto, essa medida depende fortemente das condições de navegação, em particular do estado do mar e do vento, o que implica uma maior ou menor frequência de atuação nas velas e nos lemes, que são os componentes responsáveis pelo maior consumo de energia. É de salientar que o *upgrade* para o *Beagle Bone Black* permite reduzir ainda mais o consumo energético do sistema computacional, permitindo aumentar a autonomia estimada.

Como tal seria importante realizar uma navegação de longa duração para se poder monitorizar os consumos do FASt. Essa é alias uma das propostas para trabalho futuro, que está descrito no próximo capítulo.

Capítulo 6

Conclusões e Trabalho Futuro

Após todo o trabalho desenvolvido, neste capítulo são apresentadas as conclusões que podem ser retiradas deste projeto. Começa-se com uma análise ao protótipo criado, recorrendo ao método SWOT, em seguida é realizado um resumo do trabalho desenvolvido ao longo do ano e por fim são apresentadas propostas para trabalho futuro.

6.1 Análise SWOT

O termo SWOT é uma sigla inglesa e um acrónimo de *Strengths, Weaknesses, Opportunities* e *Threats*. Esta análise foi desenvolvida para avaliar o funcionamento de empresas multinacionais, mas devido à sua simplicidade de avaliação e às conclusões que se podem retirar, pode ser aplicada a qualquer tipo de análise de projetos. Deste modo são apresentados os pontos fortes e fracos da PMU desenvolvida, assim como as oportunidades a explorar e as fraquezas do projeto que devem ser melhoradas.

6.1.1 *Strength*

- O baixo consumo da PMU em todas as fases de funcionamento, o que lhe confere uma grande autonomia, usando pilhas convencionais;

- A integração da PMU no sistema de controlo do veleiro torna o mesmo mais robusto e potencia o aumento do raio de ação do FASt;

- O sistema de navegação da rotina *AutoPilot* oferece uma redundância no controlo do veleiro, caso ocorra algum problema com o computador principal.

6.1.2 *Weakness*

- Com todo o programa carregado, o controlador tem pouca memória disponível para processar novos dados;
- Os periféricos ao controlador estão a ser constantemente alimentados, aumentado o consumo de energia;
- A configuração atual da PMU não disponibiliza ao utilizador nenhum *feedback* sobre o seu estado de funcionamento.

6.1.3 *Oportunities*

- É possível melhorar a comunicação entre o computador principal e a PMU para se obter mais dados relativos à navegação;
- Pode fazer-se da PMU uma caixa negra do veleiro, onde serão guardados dados sobre o funcionamento do computador principal e de todos os equipamentos de bordo, numa memória não volátil de grande capacidade (cartão de memória).

6.1.4 *Threats*

- A falta de um *feedback* da PMU para o utilizador pode dificultar a operação do veleiro;
- A pouca memória disponível com todo o programa carregado impossibilita o processamento e armazenamento quer de novos dados da navegação, quer de dados recolhidos por outros sensores.

6.2 Conclusão final

Após concluir o trabalho diria que o objetivo geral desta dissertação foi alcançado, uma vez que a solução apresentada no final efetua tudo o que foi proposto.

Recorrendo a uma análise da tensão disponível aos terminais das baterias e com alguns cálculos baseados na curva de descarga das mesmas, a PMU é capaz monitorizar e gerir a energia do veleiro.

A solução proposta está também programada para aplicar um de três métodos de funcionamento, *WatchDog*, *AutoPilot* e *ShutDown*. A escolha do método que é utilizado na navegação é feita tendo em conta a análise da energia disponível, tal como proposto no objetivo inicial.

Para complementar o objetivo principal, a solução desenvolvida tem incluída uma rotina que faz com que o sistema ligue e desligue de um forma controlada, de modo a garantir o sequenciamento correto do arranque dos vários componentes e, acima de tudo, o desligar controlado do computador principal, invocando as tarefas de *shutdown* antes de desligar a alimentação elétrica.

Para a concretização desta solução, numa primeira fase foi estudado qual o melhor microprocessador que garantisse os requisitos mínimos do sistema e desenvolveu-se o código que permitiu implementar os métodos de navegação e as rotinas para arrancar e desligar o sistema, fazendo uso de uma programação o mais correta e eficaz possível, utilizando sempre que possível rotinas de *sleep* para diminuir os consumos.

Posteriormente, foi estudada e desenhada uma placa que permitisse integrar o controlador com todos os periféricos necessários para a solução, que viria a ser depois construida com o auxílio do Departamento de Armas e Eletrónica da Escola Naval e da Faculdade de Engenharia da Universidade do Porto. A placa foi submetida a vários testes recorrendo a uma simulação por computador para avaliar o cumprimento dos requisitos para a qual foi criada e para quantificar os consumos da mesma, para poder estimar-se a sua autonomia.

Os resultados obtidos, permitem concluir que esta solução está bem desenvolvida, cumprindo com tudo a que foi proposto e que oferece uma autonomia suficiente para garantir um sistema de *backup* para uma missão de longa duração. Contudo, e como já foi identificado na análise SWOT, existem alguns aspetos que podem ser melhorados, como o *feedback* ao utilizador e a pouca memória disponível.

Apesar de não ter sido realizado nenhum teste em ambiente real, como estava previsto inicialmente, devido aos resultados bastante positivos que se obteve, a integração da PMU no veleiro prevê-se que ocorra sem grandes problemas, correspondendo às expectativas e aos propósitos para qual foi criada.

6.3 Trabalho Futuro

Os resultados obtidos nos testes de laboratório são muito promissores para esta solução e por isso, é um projeto ao qual se deve dar continuidade. Para tal, existem duas metas a curto prazo e duas a médio/longo prazo, às quais se devia dedicar algum tempo e trabalho para ser possível ter o FASt a realizar missões de longo curso num futuro próximo.

Assim, um dos trabalhos a ser realizado a curto prazo seria o desenvolvimento de uma nova placa da PMU, aproveitando o esquema e os integrados do protótipo criado, melhorando alguns aspetos como o tamanho da placa e a qualidade do material utilizado na sua construção.

Com a nova placa construída e já integrada no veleiro, outro trabalho a curto prazo passa por submeter o veleiro a uma simulação de uma viagem de longa duração, mas sob condições atmosféricas reais. Para tal, iria ser usado o *software* de simulação de viagens desenvolvido para o FASt, o *METASail* [34], e o veleiro seria exposto às condições atmosféricas, num local vigiado, para que os valores dos sensores fossem reais. Para tornar a simulação mais real, uma sugestão passaria por se utilizar um sistema de elásticos para que os servomotores da velas e dos lemes fossem contrariados quando estivessem a efetuar algum movimento.

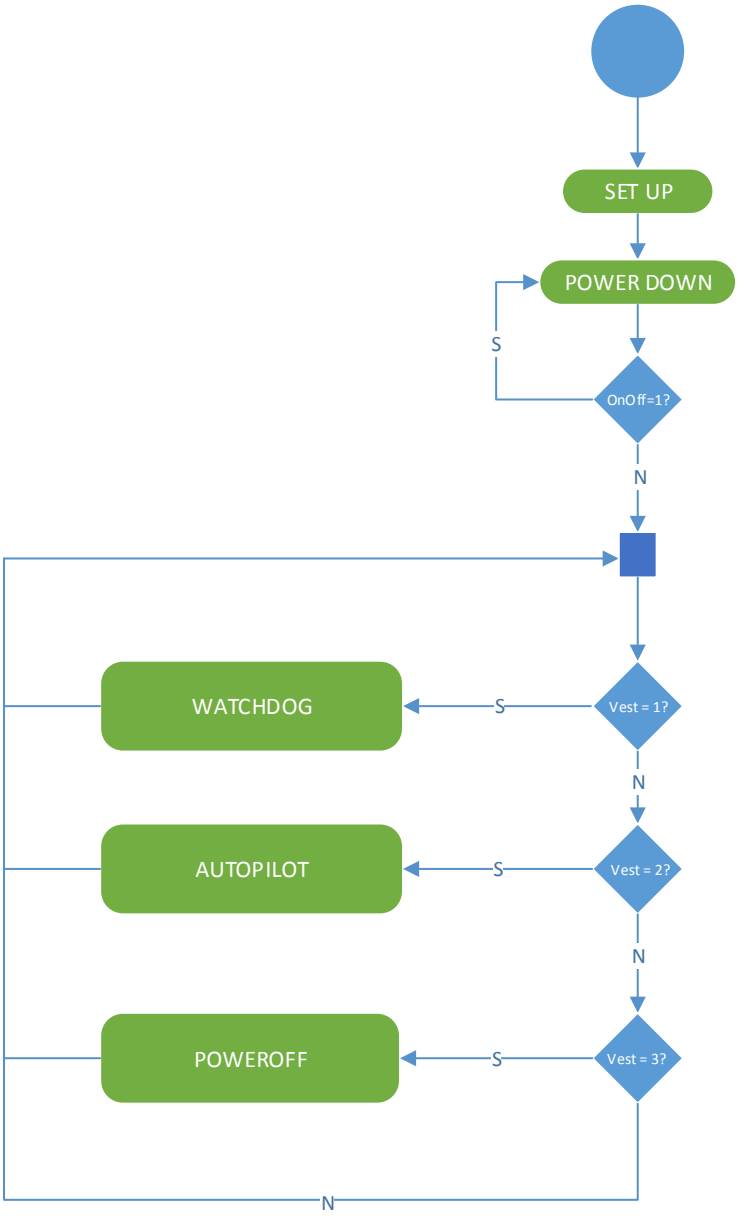
Para um trabalho a médio/longo prazo, como já foi referido no capítulo anterior, a realização de uma navegação num ambiente controlado era importante para se poder quantificar os gastos de energia do FASt. Uma sugestão interessante seria a realização da mesma navegação sem a PMU e com a PMU, de modo a poder identificar-se os ganhos de autonomia que a PMU trás ao veleiro.

Para atingir a meta final de ter o FASt a realizar missões de longa duração, sugere-se uma navegação de longo curso, de uma forma totalmente autónoma, com uma duração nunca inferior a 24 h. Essa navegação seria realizada após os resultados das simulações em ambiente controlado garantirem as condições de segurança para a sua realização. Um local recomendável para a realização desta navegação seria os Açores, com uma travessia entre o grupo Oriental e o grupo Central, por exemplo entre a cidade de Ponta Delgada, na ilha de S. Miguel e a cidade de Praia da Vitória, na ilha Terceira, completando uma distância de 90 NM.

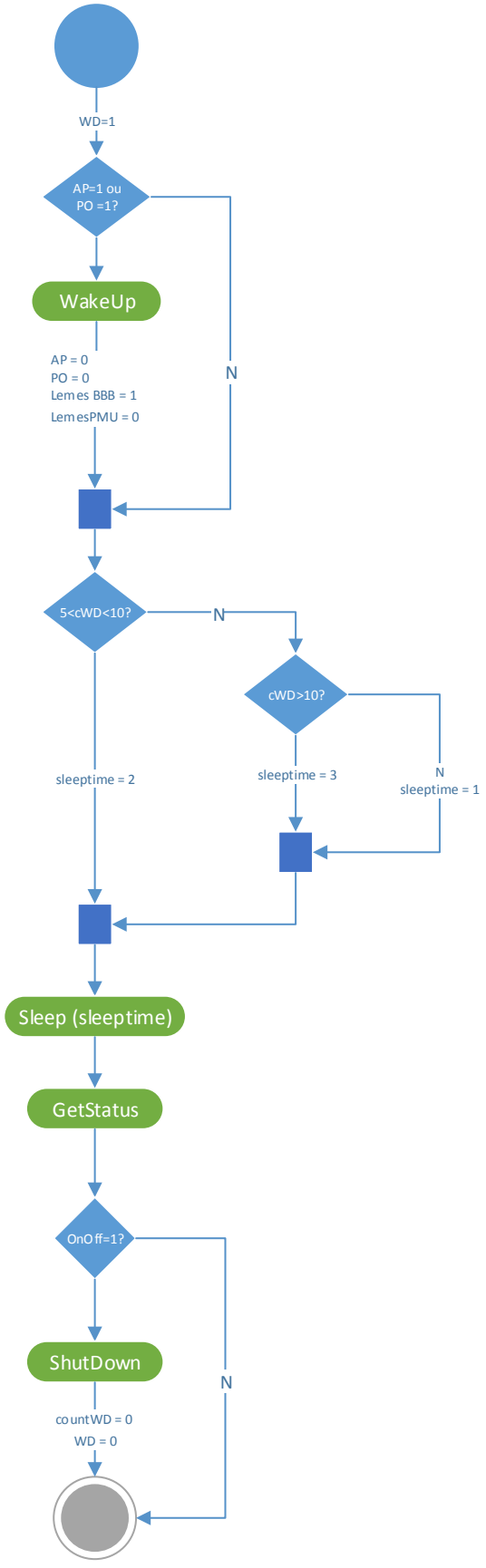
Anexo A

Fluxogramas das rotinas principais

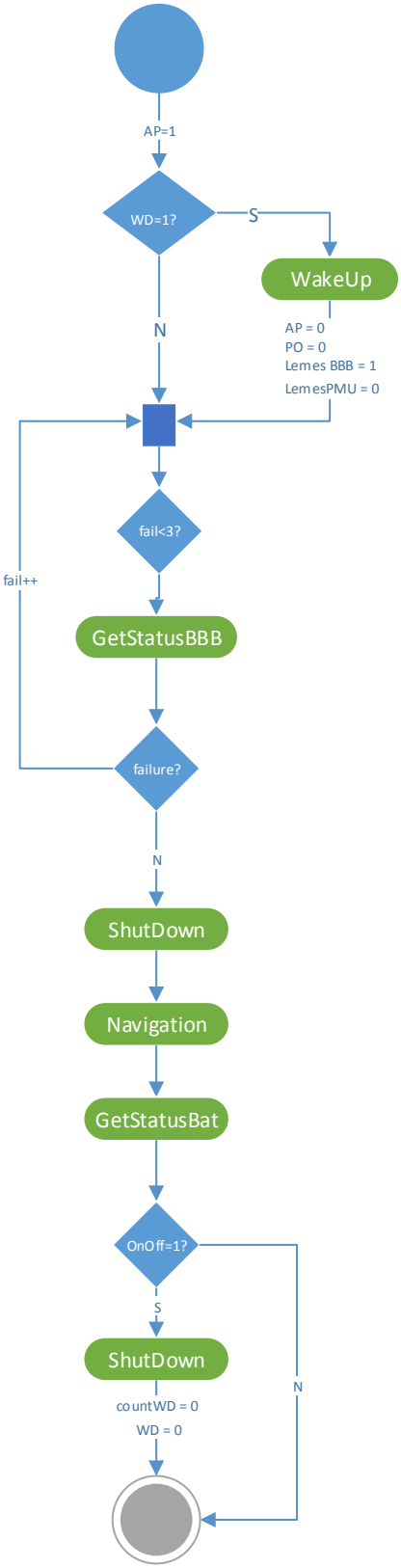
Main


SET UP
POWER DOWN
S
OnOff=1?
N
WATCHDOG
S
Vest = 1?
N
AUTOPILOT
S
Vest = 2?
N
POWEROFF
S
Vest = 3?
N

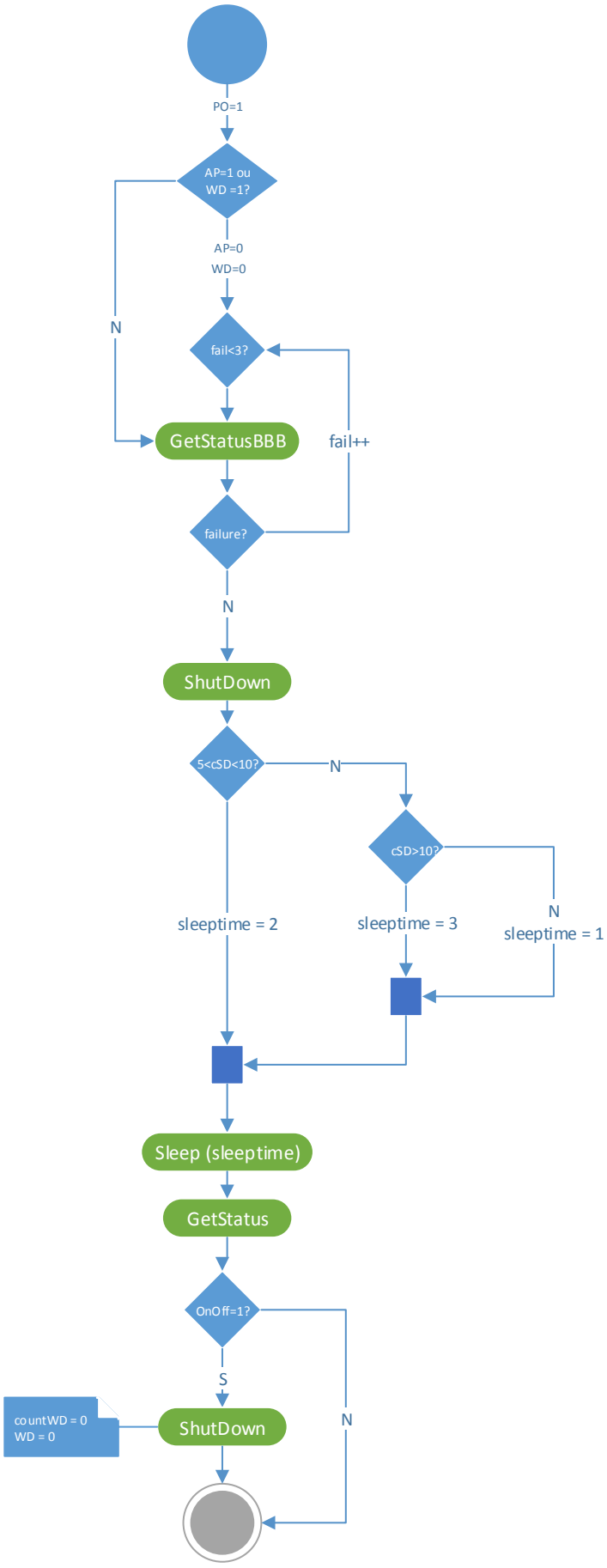
WatchDog


WD=1
AP=1 ou PO =1?
N
WakeUp
AP = 0
PO = 0
Lemes BBB = 1
LemesPMU = 0
5<cWD<10?
N
cWD>10?
sleeptime = 2
sleeptime = 3
N
sleeptime = 1
Sleep (sleeptime)
GetStatus
OnOff=1?
N
ShutDown
countWD = 0
WD = 0

AutoPilot


AP=1
WD=1?
S
WakeUp
AP = 0
PO = 0
Lemes BBB = 1
LemesPMU = 0
N
fail<3?
fail++
GetStatusBBB
failure?
N
ShutDown
Navigation
GetStatusBat
OnOff=1?
S
ShutDown
countWD = 0
WD = 0
N

PowerOff


PO=1
AP=1 ou WD =1?
AP=0
WD=0
N
fail<3?
GetStatusBBB
fail++
failure?
N
ShutDown
5<cSD<10?
N
cSD>10?
sleeptime = 2
sleeptime = 3
N
sleeptime = 1
Sleep (sleeptime)
GetStatus
OnOff=1?
S
countWD = 0
WD = 0
ShutDown
N

Anexo B

Código da PMU

```
//Bibliotecas=================================================
#include <avr/sleep.h>
#include <Sleep_n0m1.h>
#include <Wire.h> //Arduino Uno A4 (SDA), A5 (SCL)
#include <StopWatch.h>
#include <Servo.h>
//============================================================

//AltSoftSerial compass;
Sleep sleep;
StopWatch sw_bbb;
StopWatch sw_autopilot;

//CONSTANTES==================================================

//PINOUT======================================================
const int P_MOO = 2;  // pin do botão MASTER_ONOFF
const int P_LEB = 5;  // pin do leme de EB (PWM)
const int P_LBB = 6;  // pin do leme de BB (PWM)
const int P_EON = 14;  // pin do enable do decoder ON
const int P_EOFF = 13;// pin do enable do decoder OFF
const int P_CL = 7;   // pin do selecionar o controlo dos lemes
const int P_C = 10;   // pin do A2 dos decoders
const int P_B = 11;   // pin do A1 dos decoders
const int P_A = 12;   // pin do A0 dos decoders
const int P_VP = 17;  // pin da tensao do painel solar (analogico)
const int P_Bat2 = 15;// pin da tensao da bateria 1 (analogico)
const int P_Bat1 = 15;// pin da tensao da bateria 2 (analogico)

//MATRIZ======================================================
const int cols = 5, rows = 3;

//START=======================================================
//tempo que decorre desde que inicia a rotina ate o euipamento ligar
int PowerON = 1000, WifiON = 3000, BBBON = 6000, GPSON = 9000, RadioON = 12000;
int OnArray [3][5] = {
  {1, 2, 3, 4, 5},
  {PowerON, WifiON, BBBON, GPSON, RadioON},
  {0, 0, 0, 0, 0}
};


//SHUTDOWN====================================================
int PowerOFF = 12000, WifiOFF = 3000, BBBOFF = 1000, GPSOFF = 6000, RadioOFF = 9000;
int OffArray [3][5] = {
  {1, 2, 3, 4, 5},
  {PowerOFF, WifiOFF, BBBOFF, GPSOFF, RadioOFF},
  {0, 0, 0, 0, 0}
};

//SLEEP TIME==================================================
unsigned long SleepArrayWD [3] = {3000, 6000, 9000};            //3seg,6seg,9seg
unsigned long SleepArraySD [3] = {9000, 12000, 15000};          //9seg,12seg,15seg
//unsigned long  SleepArrayWD [3] = {60000, 180000, 300000};    //1min, 3min, 5min
//unsigned long  SleepArraySD [3] = {60000, 120000, 180000};    //1min, 2min, 3min

//============================================================
```

```
//VARIAVEIS=============================================================
unsigned long sleepTime;         //how long you want the arduino to sleep (ms)

//CONTROLO==============================================================
int var_estado = 1; //estado da navegação (1)navegação normal / (2)shutdown / (3)autopilot
int WD = 0;         //navegação normal - info para o AUTOPILOT
int AP = 0;         //auto pilot - info para o SHUTDOWN
int PO = 0;
int reset = 0;      //variavel que controla o reset do BBB
volatile int MasterOnOff = 0;    //varivael de estado do botão MASTER_ONOFF
volatile boolean process_it = false; //controlo da receção de dados do BBB

//CONTADORES============================================================
int count_WD=0;       //quantos cilcos de watchdog fez
int count_SD=0;      //quantos cilcos de shutdown fez
int fail = 0;           //conta numero de fail

//GETSTATUSBBB==========================================================
char w = w;
char buf [14];
byte pos;
char c;
int bbbstat;

String OK, vm, r;
volatile float vel; //velocidade media
volatile float rumo;    //rumo
int j = 1;      //contador

//BUSSOLA===============================================================
String data, head, roll, pitch;
boolean done = false;
char d;

//NAVIGATION============================================================
Servo lemeBB;  // create servo object to control a servo
int val = 511;    // variable to read the value from the analog pin
int BB = 5;

//error
float error_heading; //diferença entre rumo e proa
float error_heading_d; //diferença entre error_heading e error_heading0
float error_heading0; //erro heading da ultima iteração
int error_heading_i = 0; //integral error

//receive data from BBB
float hdg_setting; // rumo definido pelo BBB;
float averaged_speed; // velocidade média fornecida pelo bbb

//receive data from compass
float averaged_hdg; // proa dada pela bussola

//PID constants
int heading_p_gain = 4000;
int heading_i_gain = 100;
int heading_d_gain = 0;
float max_hdg_diff_integral = 10; //maximo current heading error
float max_hdg_integral_error = 100;
```

```
//GETSTATUSBAT===============================================
#define NUM_SAMPLES 10
#define Vmin 9.6
#define Vmax 16.8
#define R1 9850
#define R2 3240
const float Vint = Vmax - Vmin;
const float VDV = (R2 + R1) / R2; //voltage divide value
float batstatus;
//============================================================

void setup () {
  //Interruptor principal
  pinMode(P_MOO, INPUT);

  //Pins decoders
  pinMode(P_A, OUTPUT);
  pinMode(P_B, OUTPUT);
  pinMode(P_C, OUTPUT);
  pinMode(P_EON, OUTPUT);
  pinMode(P_EOFF, OUTPUT);
  digitalWrite(P_EON, 0);
  digitalWrite(P_EOFF, 0);

  //Pins lemes
  pinMode(P_LEB, OUTPUT);
  pinMode(P_LBB, OUTPUT);
  pinMode(P_CL, OUTPUT);
  digitalWrite(P_CL, 0);

  //Pins baterias e painel solar
  pinMode(P_VP, INPUT);
  pinMode(P_Bat2, INPUT);
  pinMode(P_Bat1, INPUT);

  //order os vetores de delay
  Sort(OnArray);        //ordena o vetor dos delays de iniciação
  Sort(OffArray);       //ordena o vetro dos dealys de shutdown

  Wire.begin();        // join i2c bus (address optional for master)
  lemeBB.attach(BB);
  Serial.begin(19200);  // iniciar comunicacao serie com o pc
  Serial.flush();
  Serial.println("O micro em modo POWER_DOWN"); //Serial info
  delay(100);
  PowerDown();     // entra em modo PowerDown
}
//============================================================

void loop() {
  if (MasterOnOff == 1) {
    WakeUp();
  }
  MasterOnOff = 0;
  switch (var_estado) {     //var_estado (1)navegação normal  / (2)autopilot  / (3)shutdown
    case 1:          //(1)navegação normal
```

```
      WatchDog();
      count_WD++;       //incrementa a contagem dos ciclos de sleep
      break;
    case 2:                 //(2)autopilot
      Autopilot();
      break;
    case 3:         //(3)shutdown
      PowerOff();
      break;
    default:
      break;
  }
}

//WATCHDOG=====================================================
//WatchDog do MPU
void WatchDog() {
  Serial.println("\nwatchdog");
  WD = 1;

  if ((AP == 1)||(PO ==1)) {
    AP = 0;
    PO = 0;
    WakeUp();
    digitalWrite(P_CL, 0);
    Serial.println("lemes BBB");
    digitalWrite(P_LEB, 1); //APAGAR
  }

  if (count_WD > 5 && count_WD <= 10) { //paraemtrizar o sleeptime par ao watchdog
    sleepTime = SleepArrayWD [1];       // 1° sleeptime
    //Serial.println(sleepTime, DEC);
  }else {
    if (count_WD > 10) {
      sleepTime = SleepArrayWD [2];     //2°sleeptime
     // Serial.println(sleepTime, DEC);
    }else {
      sleepTime = SleepArrayWD [0];     //3°sleeptime
     // Serial.println(sleepTime, DEC);
  }}

  delay(1000);
  sleep.pwrDownMode();      //entra no modo debaixo comsumo(powerDownmode)
  sleep.sleepDelay(sleepTime);      //tempo definido automaticamente
  delay(100);

  //obter dados
  GetStatus();

  //detetar se masteronoff foi primido
  if (MasterOnOff == 1) {   //detetor do MasterOnOff para encerrar o sistema
    count_WD = 0;       //reset ao contador de ciclos sleep
    WD = 0;
    ShutDown();         //entra em modo ShutDown
  }
}
//=============================================================
```

```
//AUTOPILOT=============================================================
void Autopilot () {
  Serial.println("\nautopilot");
  PO=0;
  AP=1;
  if (WD == 1) {
    WD = 0;
    count_WD = 0;       //reset ao contador de ciclos sleep
  }
  else {
    WakeUp();
  }

  if (fail < 3) {
    GetStatusBBB();
  }

  ShutDown();

  Navigation();

  GetStatusBat();

  if (MasterOnOff == 1) {   //detetor do MasterOnOff para encerrar o sistema
    count_WD = 0;       //reset ao contador de ciclos sleep
    AP = 0;
    ShutDown();         //entra em modo ShutDown
  }
}
//=====================================================================

void Navigation (){
  Serial.println("\nnavigation");
  digitalWrite(P_CL, 1);
  Serial.println("lemes micro");
  digitalWrite(P_LEB, 0);//APAGAR
  digitalWrite(P_LBB, 0);//APAGAR
  sw_autopilot.start();

  while (sw_autopilot.elapsed() < 10000){
    float p_gain_correcting_factor;
    averaged_hdg = Bussola();
    if (averaged_hdg>360)
      averaged_hdg = Bussola();
    Serial.print("Heading(Degrees): ");Serial.println(hdg_setting);
    Serial.print("Averaged Speed: ");Serial.println(averaged_speed);
    Serial.print("Averaged Heading - ");Serial.println(averaged_hdg);
    Serial.println("");
    // heading error in integer degrees:
    error_heading = ( heading_diff( hdg_setting, averaged_hdg ));  // + 5 ) / 10; //- ver
    este caso com o professor
    Serial.print("error heading - ");Serial.println(error_heading);
    // integrate the error only if the current heading error is less than 10 degrees
    (max_hdg_diff_integral)
    if ( abs(error_heading) < max_hdg_diff_integral ) {
      // saturarte the integral to a maximum of 100 units (max_hdg_integral_error)
      if ( abs(error_heading_i) < max_hdg_integral_error )
        error_heading_i += error_heading;
```

```
      //Serial.print("error heading i - ");Serial.println(error_heading_i);
  }
  else { // if not integrating, decrease the integral dividing by 2 in each iteration:
    error_heading_i >>= 1 ;
    //Serial.print("error heading i - ");Serial.println(error_heading_i);
  }

  // calculate the difference to the previous error
  // if the current error is less than the previous error, this will be
  // negative and
  error_heading_d = heading_diff(error_heading, error_heading0);
  //Serial.print("error heading d - ");Serial.println(error_heading_d);
  error_heading0 = error_heading;
  //Serial.print("error heading 0 - ");Serial.println(error_heading0);

  // compute the actuation:
  // proportional gain:
  p_gain_correcting_factor = 1.0;

  // if speed is greater than 2.0 knots, reduce the gain:
  if ( averaged_speed > 2.0 ) {
    p_gain_correcting_factor = ((float)20) /  averaged_speed;
  }
  //Serial.print("p_gain_correcting_factor - ");Serial.println(p_gain_correcting_factor);

  // proportional gain: 9000
  int servo_control = ((p_gain_correcting_factor * heading_p_gain * error_heading))/1024;
  //Serial.print("servo control 1 - ");Serial.println(servo_control);
  // add the integral gain: integral gain: 100
  servo_control += ((int)(p_gain_correcting_factor * heading_i_gain * error_heading_i)) >>
  10;
  //Serial.print("servo control 2 - ");Serial.println(servo_control);
  // add the derivative gain: derivative gain: 0 (not used).
  servo_control += ((int)(p_gain_correcting_factor * heading_d_gain * error_heading_d)) >>
  10;
  //Serial.print("servo control 3 - ");Serial.println(servo_control);
  servo_control /= 12.31;
  //Serial.print("servo control 4 - ");Serial.println(servo_control);
  if (servo_control>50){
     servo_control = 50;
  }else{
    if (servo_control<-50){
      servo_control = -50;
    }
  }
  val = 90 + servo_control;
  lemeBB.write(val);    // sets the servo position according to the scaled value
  Serial.print("Leme: ");Serial.println(val);


  Serial.print("Rudder Position: ");Serial.print(abs(servo_control));
  if (servo_control>0){
    Serial.println(" EB");
  }else{
    Serial.println(" BB");
  }
  Serial.println("--------------------");
  delay(1500);
```

```
  }

  Serial.println("\nO FASt navegou durante 5min.");

  sw_autopilot.stop();
  sw_autopilot.reset();

  digitalWrite(P_CL, 0);
  Serial.println("lemes BBB");
}

//POWEROFF=================================================
void PowerOff() { //POWEROFF
  Serial.println("\npoweroff");
  if ((AP == 1) || (WD == 1)) {
    AP = 0;
    WD = 0;
    PO = 1;
    if (fail!=3)
      bbbstat = GetStatusBBB();
    ShutDown();
  }
  if (count_SD > 2 && count_SD <= 4) { //paraemtrizar o sleeptime par ao watchdog
    sleepTime = SleepArraySD [1];        // 1° sleeptime
  }else {
    if (count_SD > 4) {
      sleepTime = SleepArraySD [2];      //2°sleeptime
    }else {
      sleepTime = SleepArraySD [0];      //3°sleeptime
  }}
  digitalWrite(P_EOFF, 0);
  sleep.pwrDownMode();       //entra no modo debaixo comsumo(powerDownmode)
  sleep.sleepDelay(sleepTime);       //tempo definido automaticamente
  count_SD++;        //incrementa a contagem dos ciclos de shutdown
  delay (1000);
  //Serial.println(sleepTime, DEC); //Serial info
  GetStatusBat();
}
//=========================================================

//GETSTATUS================================================
//obter dados de navegação e de bateria
void GetStatus() {
  GetStatusBat();
  fail = 0;
  while (fail <= 2) {
    bbbstat = GetStatusBBB();
    if (bbbstat == 1) {
      ++fail;
    }
    else {
      fail = 0;
      break;
    }
  }
  if ((fail == 3)&&(var_estado!=3)) {
    var_estado = 2;
  }
```

```
}
//============================================================

//BBB=========================================================
//faz checkup ao BBB e obtem dados de navegação
int GetStatusBBB() {
  Serial.print("\ngetstatusbbb");
  Wire.requestFrom(2, 12);
  j = 1; pos = 0;
  OK = ""; vm = ""; r = "";
  process_it = false;
  sw_bbb.start();
  while ((sw_bbb.elapsed() < 2500)){
    if (Wire.available()){
      char c = Wire.read(); // receive a byte as character
      buf [pos++] = c;
      process_it = true;
    }
  }
  if (process_it) {
    for (int i = 0; i < pos; i++) {
      if (buf[i] != ',') {
        switch (j) {
          case 1 :
            vm += buf[i];
            break;
          case 2 :
            r += buf[i];
            break;
          default :
            break;
        }
      }
      else {
        j++;
      }
    }
    averaged_speed = vm.toFloat();
    hdg_setting = r.toFloat();
    Serial.print("   Vel media: ");Serial.print(averaged_speed);Serial.print("   Rumo: ");
    Serial.println(hdg_setting);
    sw_bbb.stop();
    sw_bbb.reset();
    return (0);
  }
  else {
    sw_bbb.stop();
    sw_bbb.reset();
    return (1);
  }
}
//============================================================

//BAT=========================================================
//obtem estado da bateria por analise de voltagem
void GetStatusBat() {
  Serial.print("\ngetstatusbat");
  float val = 0, percent = 0, sum1 = 0, sum2 = 0;
```

```
  unsigned char sample_count = 0; // current sample number
  float voltage = 0.0;            // calculated voltage

  while (sample_count < NUM_SAMPLES) {
    sum1 += analogRead(P_Bat1);
    sum2 += analogRead(P_Bat2);
    sample_count++;
    delay(10);
  }
  sum1 = (sum1 + sum2) / 2;
  float amostra = ((float)sum1 / (float)NUM_SAMPLES * 5.015);
  voltage = (amostra / 1024.0) * VDV;
  Serial.print("Voltagem:");Serial.print(voltage);
  if (voltage < Vmin) {
    val = 0;
  } else {
    if (voltage > Vmax) {
      val = 1;
    }
    else {
      val = (voltage - Vmin) / Vint;
    }
  }
  percent = (val) * 100;Serial.print("     Estado:");
  Serial.println(percent);
  if (percent < 25) {
    var_estado = 3; //var_estado (1)navegação normal  / (2)autopilot  / (3)shutdown
  } else {
    if ((percent > 35) && (var_estado == 3)) {
      WD = 1;
      var_estado = 1; //var_estado (1)navegação normal  / (2)autopilot  / (3)shutdown
    }else {
      if (percent < 65){
        AP = 1;
        var_estado = 2;
      }else {
        WD = 1;
        var_estado = 1;
       }
    }
  }
}
//================================================================

//POWERDOWN=============================================================
//o MPU enta em modo sleep
void PowerDown() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);   // sleep mode is set here
  sleep_enable();          // enables the sleep bit in the mcucr register so sleep is
  possible. just a safety pin
  noInterrupts();
  attachInterrupt(0, Button, RISING); // use interrupt 0 (pin 2) and run function wakeUpNow
  when pin 2 gets LOW
  interrupts();
  sleep_mode();            // here the device is actually put to sleep!! THE PROGRAM
  CONTINUES FROM HERE AFTER WAKING UP
}
//================================================================
```

```
//BUSSOLA=============================================================
float Bussola () {
  Serial.println("\nbussola");

  //compass.begin(19200);
  done=false;
  data=" ";head=" ";
  if (Serial.available()) {
    while (!done){
      d = Serial.read();
      data += d;
      if (d == '\n')
        done = true;
    }
  }
  int first = data.indexOf(',');
  head = data.substring(0, first);
  Serial.print("head: ");  Serial.println(head);
  return (head.toFloat());
}
//====================================================================

//SORT================================================================
//função para odenar vetores de inciação
void Sort (int myArray [3][5]) {
  int i0 = 0, i1 = 1,  i2 = 2;
  int cols = 5, rows = 3;
  int size = 5;
  for (int o = 0; o < (size - 1); o++) {         //ordena os equipamentos para definir que
  liga/desliga primeiro
    for (int j = 0; j < (size - (o + 1)); j++) {
      if (myArray [i1][j] > myArray[i1][j + 1]) {
        int eaux = myArray [i0][j];
        int taux = myArray [i1][j];
        myArray [i0][j] = myArray [i0][j + 1];
        myArray [i1][j] = myArray [i1][j + 1];
        myArray [i0][j + 1] = eaux;
        myArray [i1][j + 1] = taux;
      }
    }
  }
  for (int j = 0; j < size; j++) {       //faz os delays entre equipamentos
    if (j == 0) {
      myArray [i2][j] = myArray [i1][j];
    }
    else {
      myArray [i2][j] = myArray [i1][j] - myArray [i1][j - 1];
    }
  }
}
//====================================================================

//WAKEUP==============================================================
//função para acordar o MPU do modo ShutDown
void WakeUp() {
  Serial.println("\nwakeup");
  int j = 0;
```

```
if (AP == 1) {
  Serial.flush();
  Serial.println("AP_Reboot - O micro vai fazer reboot para obter dados");//Serial info
}
else {
  Serial.flush();
  Serial.println("PWRON_INIT - O sistema vai iniciar.");//Serial info
}

for (j; j < 5; j++) {     //liga os equipamentos
  switch (OnArray [0][j]) {
    case 1:       //liga as baterias
      if ((AP == 1)||(WD == 1)) { // reset
        break;
      }
      else {
        delay(OnArray [2][j]);
        digitalWrite(P_A, 0 % 2);
        digitalWrite(P_B, (0 >> 1) % 2);
        digitalWrite(P_C, (0 >> 2) % 2);
        digitalWrite(P_EON, 1);
        digitalWrite(P_EOFF, 0);
        Serial.println("Power Supply - ON");
        delay(1000);
        break;
      }
    case 2:       //liga o wifi
      delay(OnArray [2][j]);
      digitalWrite(P_A, 1 % 2);
      digitalWrite(P_B, (1 >> 1) % 2);
      digitalWrite(P_C, (1 >> 2) % 2);
      digitalWrite(P_EON, 1);
      digitalWrite(P_EOFF, 0);
      Serial.println("Router WIFI - ON");
      delay(1000);
      break;
    case 3:       //liga o BBB
      delay(OnArray [2][j]);
      digitalWrite(P_A, 2 % 2);
      digitalWrite(P_B, (2 >> 1) % 2);
      digitalWrite(P_C, (2 >> 2) % 2);
      digitalWrite(P_EON, 1);
      digitalWrite(P_EOFF, 0);
      Serial.println("Main CPU BBB - ON");
      delay(1000);
      break;
    case 4:       //liga o gps
      delay(OnArray [2][j]);
      digitalWrite(P_A, 3 % 2);
      digitalWrite(P_B, (3 >> 1) % 2);
      digitalWrite(P_C, (3 >> 2) % 2);
      digitalWrite(P_EON, 1);
      digitalWrite(P_EOFF, 0);
      Serial.println("GPS - ON");
      delay(1000);
      break;
    case 5:       //liga os radios
```

```
        delay(OnArray [2][j]);
        digitalWrite(P_A, 4 % 2);
        digitalWrite(P_B, (4 >> 1) % 2);
        digitalWrite(P_C, (4 >> 2) % 2);
        digitalWrite(P_EON, 1);
        digitalWrite(P_EOFF, 0);
        Serial.println("RCRadio - ON");
        Serial.println("SerialRadio - ON");
        delay(1000);
        break;
    }
  }
  digitalWrite(P_A, 0);
  digitalWrite(P_B, 0);
  digitalWrite(P_C, 0);
  digitalWrite(P_EON, 0);
}
//================================================================

//SHUTDOWN=====================================================
//função para o MPU entrar em modo ShutDown
void ShutDown()
{
  Serial.println("\nshutdown");
  int j = 0;
  if (AP == 1) {
    Serial.flush();
    Serial.println("AP_ShutDown - O micro em modo Autopilot");//Serial info
  }
  else {
    Serial.flush();
    Serial.println("SHUTDOWN_1 - Sistema a desligar."); //Serial info
  }
  for (j; j < 5; j++) {
    switch (OffArray [0][j]) {
      case 1:       //desliga baterias
        if (AP == 1) { //reset
          break;
        }
        else {
          delay (OffArray [2][j]);
          digitalWrite(P_A, 0 % 2);
          digitalWrite(P_B, (0 >> 1) % 2);
          digitalWrite(P_C, (0 >> 2) % 2);
          digitalWrite(P_EON, 0);
          digitalWrite(P_EOFF, 1);
          Serial.println("Power Supply - OFF");
          delay(1000);
          break;
        }
      case 2:       //desliga wifi
        delay(OffArray [2][j]);
        digitalWrite(P_A, 1 % 2);
        digitalWrite(P_B, (1 >> 1) % 2);
        digitalWrite(P_C, (1 >> 2) % 2);
        digitalWrite(P_EON, 0);
        digitalWrite(P_EOFF, 1);
        Serial.println("Router WIFI - OFF");
```

```
        delay(1000);
        break;
      case 3:       //desliga o BBB
        delay(OffArray [2][j]);
        digitalWrite(P_A, 2 % 2);
        digitalWrite(P_B, (2 >> 1) % 2);
        digitalWrite(P_C, (2 >> 2) % 2);
        digitalWrite(P_EON, 0);
        digitalWrite(P_EOFF, 1);
        Serial.println("Main CPU BBB - OFF");
        delay(1000);
        break;
      case 4:       //desliga o gps
        delay(OffArray [2][j]);
        digitalWrite(P_A, 3 % 2);
        digitalWrite(P_B, (3 >> 1) % 2);
        digitalWrite(P_C, (3 >> 2) % 2);
        digitalWrite(P_EON, 0);
        digitalWrite(P_EOFF, 1);
        Serial.println("GPS - OFF");
        delay(1000);
        break;
      case 5:       //desliga o radio
        delay(OffArray [2][j]);
        digitalWrite(P_A, 4 % 2);
        digitalWrite(P_B, (4 >> 1) % 2);
        digitalWrite(P_C, (4 >> 2) % 2);
        digitalWrite(P_EON, 0);
        digitalWrite(P_EOFF, 1);
        Serial.println("RCRadio - OFF");
        Serial.println("SerialRadio - OFF");
        delay(1000);
        break;
    }
  }
  digitalWrite(P_A, 0);
  digitalWrite(P_B, 0);
  digitalWrite(P_C, 0);
  digitalWrite(P_EOFF, 0);
  delay(1000);
  if (MasterOnOff == 1) { //reset
    MasterOnOff = 0;
    Serial.println("Micro em modo PWRDONW. ");
    PowerDown();        //entra em modo PowerDown
  }
}
//=================================================================

//BUTTON=========================================================
//muda o estado da variavel MasterOnOff=1 quando o botão é primido
void Button() {
  MasterOnOff = 1;
  sleep_disable();
  //detachInterrupt (0);
}
//=================================================================

void ReduceAngleto180(int a) {
```

```
  while ( a > 1800 )
    a -= 3600;
  while ( a < -1800 )
    a += 3600;
}

void ReduceAngleto360(int a) {
  while ( a < 0 )
    a += 3600;
  while (a > 3600 )
    a -= 3600;
}

int heading_diff(int h1, int h2) {
  int diff = h1 - h2;
  if ( diff > +180.0 ){
    diff -= 360.0;
  }else{
    if ( diff < -180.0 ){
      diff = 360.0 + diff;
  }}
  return ( diff );
}

int heading_add(int h1, int h2) {
  int hs = h1 + h2;
  ReduceAngleto360( hs );
  return hs;
}
```

Anexo C

Datasheet das pilhas Energizer AA e D

1-800-383-7323 USA/CAN
www.energizer.com

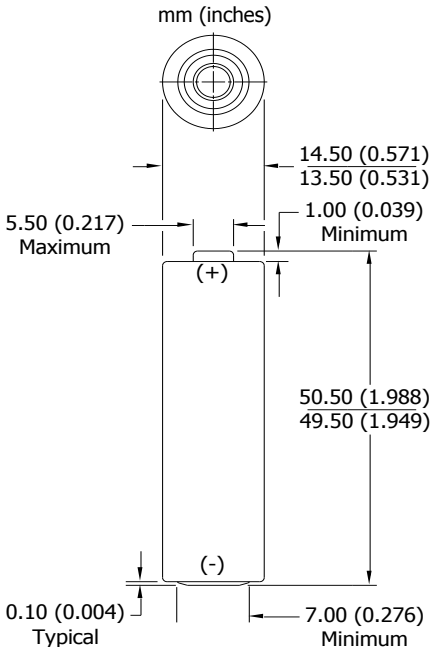
ENERGIZER E91

AA

Specifications

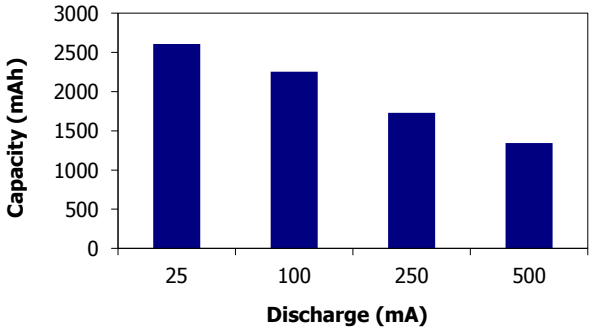
Classification:	Alkaline
Chemical System:	Zinc-Manganese Dioxide (Zn/MnO_2) No added mercury or cadmium
Designation:	ANSI-15A, IEC-LR6
Nominal Voltage:	1.5 volts
Nominal IR:	150 to 300 milliohms (fresh)
Operating Temp:	-18°C to 55°C (0°F to 130°F)
Typical Weight:	23.0 grams (0.8 oz.)
Typical Volume:	8.1 cubic centimeters (0.5 cubic inch)
Jacket:	Plastic Label
Shelf Life:	10 years at 21°C
Terminal:	Flat Contact

Industry Standard Dimensions



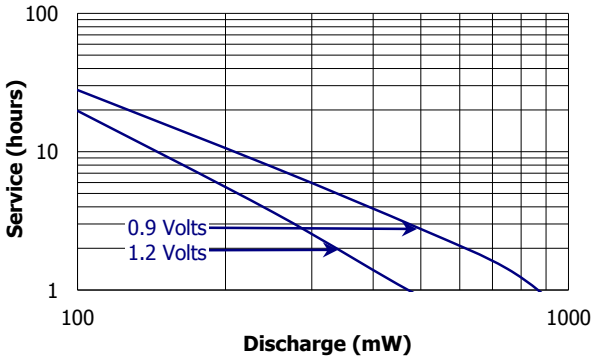
Milliamp-Hours Capacity

Continuous discharge to 0.8 volts at 21°C



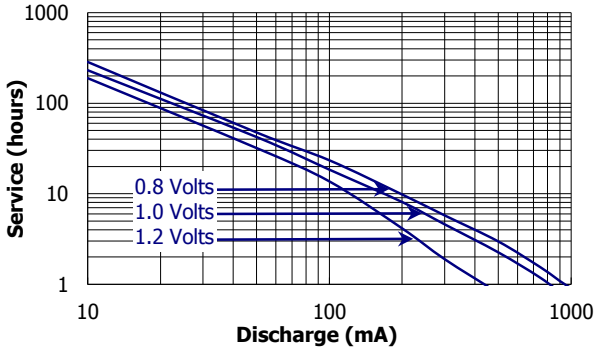
Constant Power Performance

Typical Characteristics (21°C)



Constant Current Performance

Typical Characteristics (21°C)



Important Notice

This data sheet contains typical information specific to products manufactured at the time of its publication.
Physical values are for reference purposes and not intended for specific calculations.

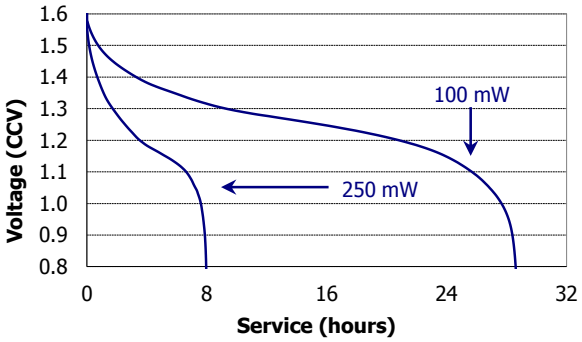
1-800-383-7323 USA/CAN
www.energizer.com

ENERGIZER E91

AA

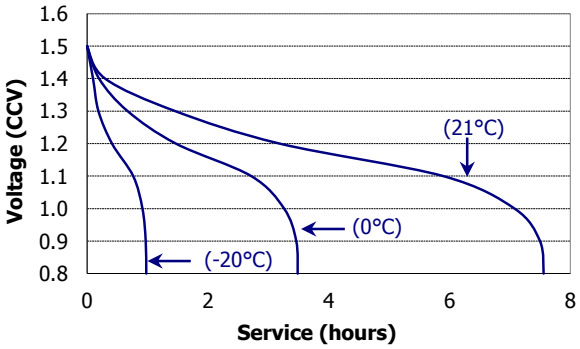
Constant Power Performance

Discharge Characteristics (21°C)



Constant Current Performance

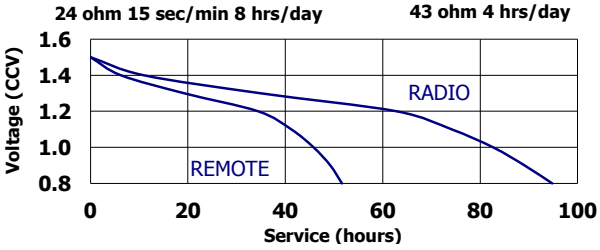
250 mA Discharge (-20°C / 0°C / 21°C)



Industry Standard Tests (21°C)

REMOTE
24 ohm 15 sec/min 8 hrs/day

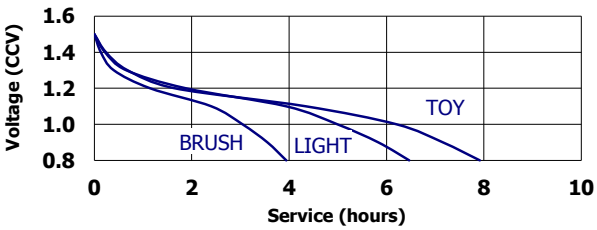
RADIO
43 ohm 4 hrs/day



TOOTHBRUSH
500 mA 2/13 min 24 hrs/day

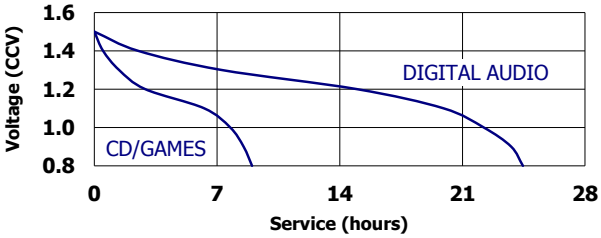
PORTABLE LIGHTING
3.3 ohm LIF

TOY
3.9 ohm 1 hr/day



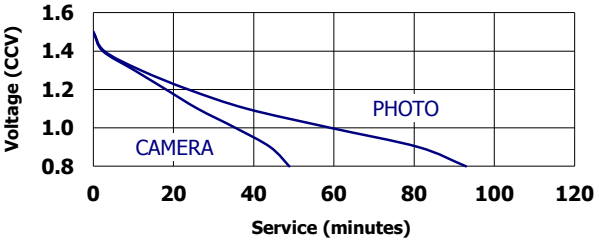
CD/GAMES
250 mA 1 hr/day

DIGITAL AUDIO
100 mA 1 hr/day



DIGITAL CAMERA
1.5K/.65K mW 2./1./27.9s 10x

PHOTOFLASH
1K mA 10 sec/min 1 hr/day

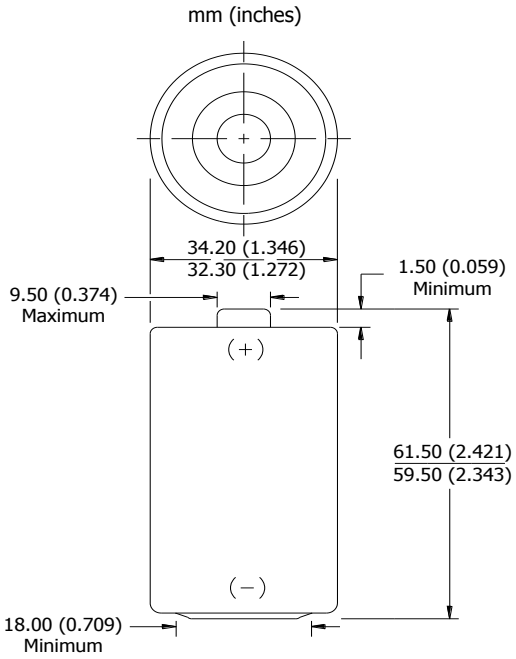


1-800-383-7323 USA/CAN
www.energizer.com

ENERGIZER E95

D

Industry Standard Dimensions

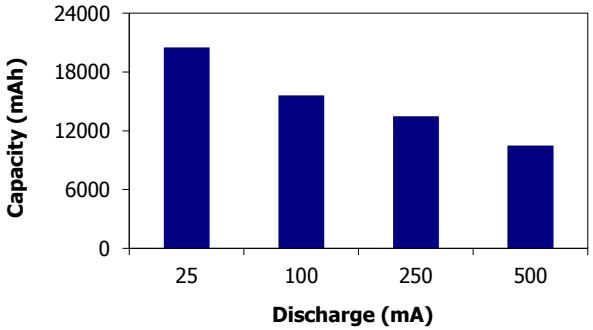


Specifications

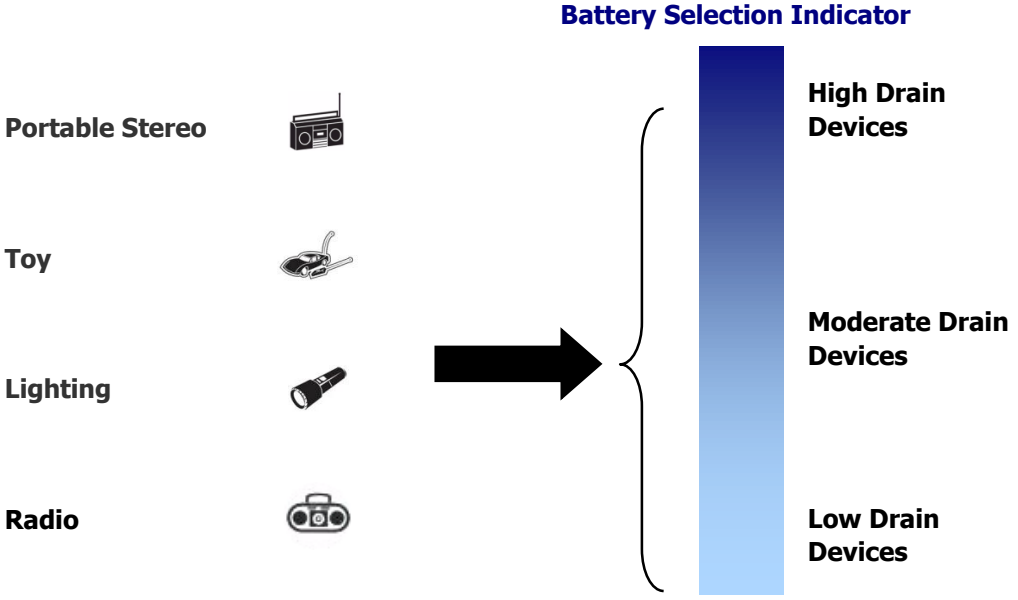
Classification:	Alkaline
Chemical System:	Zinc-Manganese Dioxide (Zn/MnO_2) No added mercury or cadmium
Designation:	ANSI-13A, IEC-LR20
Nominal Voltage:	1.5 volts
Nominal IR:	150 to 300 milliohms (fresh)
Operating Temp:	-18°C to 55°C (0°F to 130°F)
Typical Weight:	144.0 grams (5.1 oz.)
Typical Volume:	56.0 cubic centimeters (3.4 cubic inch)
Jacket:	Plastic Label
Shelf Life:	10 years at 21°C
Terminal:	Flat Contact

Milliamp-Hours Capacity

Continuous discharge to 0.8 volts at 21°C



Device Selection Guide:



Important Notice

This data sheet contains typical information specific to products manufactured at the time of its publication.

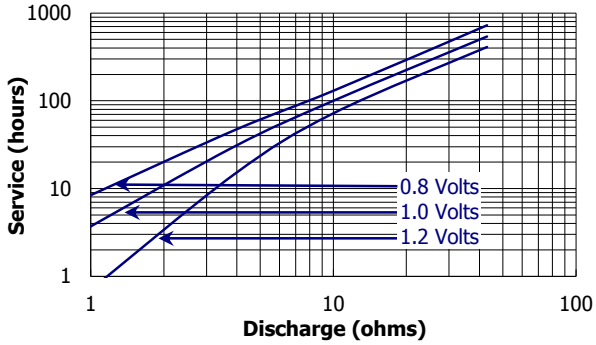
Energizer
1-800-383-7323 USA/CAN
www.energizer.com

ENERGIZER E95

D

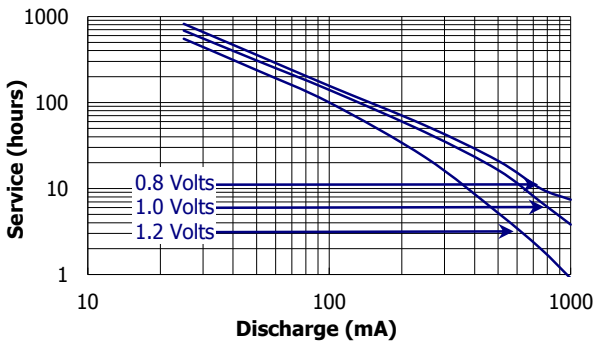
Constant Resistance Performance

Typical Characteristics (21°C)



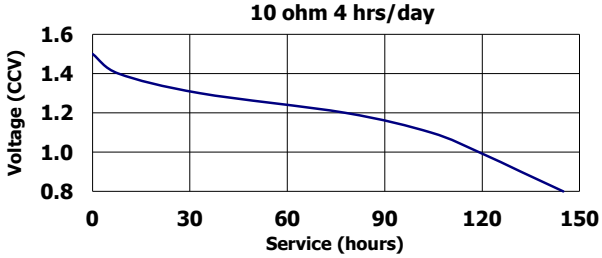
Constant Current Performance

Typical Characteristics (21°C)



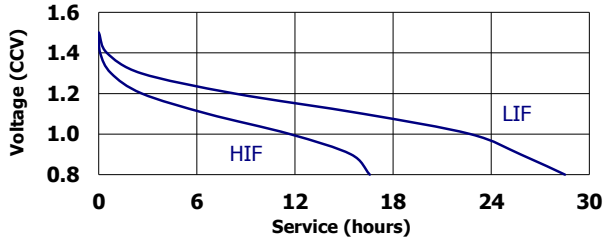
Industry Standard Tests (21°C)

RADIO
10 ohm 4 hrs/day

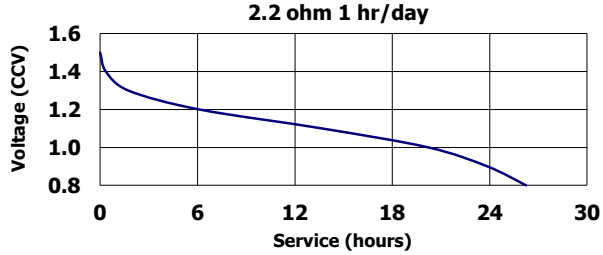


LIGHTING
1.5 ohm HIF

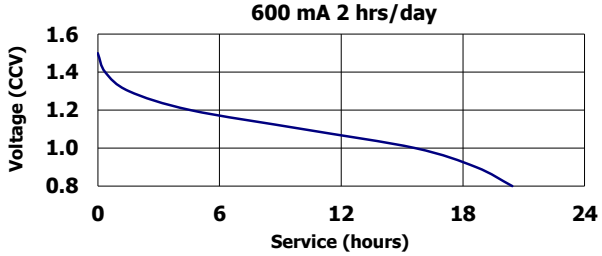
LIGHTING
2.2 ohm LIF



TOY
2.2 ohm 1 hr/day



PORTABLE STEREO
600 mA 2 hrs/day



Referências

[1] How many oceans are there?, Janeiro 2015. URL: http://oceanservice.noaa.gov/facts/howmanyoceans.html/company/.

[2] Oceanos do planeta terra, Janeiro 2015. URL: http://planeta-terra.info/oceanos-do-planeta-terra.html.

[3] A. Einstein. *The World As I See It*. 2006. URL: https://books.google.pt/books?id=JFXWosy8ywYC.

[4] Eship-1, Janeiro 2015. URL: http://www.enercon.de/en-en/2224.htm.

[5] Skysails, Janeiro 2015. URL: http://www.skysails.info/english/company/.

[6] Navios de propulsão eólica, 2015. URL: http://meioseculodeaprendizagens.blogspot.com/2011/05/navios-de-propulsao-eolica.html#ixzz3eFwCtdCxl.

[7] J. C. Alves e N. A. Cruz. Ocean sampling and surveillance using autonomous sailboats. páginas 21–29, May 2008.

[8] Saildrone, Junho 2015. URL: http://www.saildrone.com/.

[9] Saildrone: Moving forward, Junho 2015. URL: http://mstfoundation.org/story/Saildrone2.

[10] Wrsc/irsc 2009, Janeiro 2015. URL: http://paginas.fe.up.pt/~jca/wrsc/.

[11] L. Le Bars, F. e Jaulin. An experimental validation of a robust controller with the vaimos autonomous sailboat. Em *Robotic Sailing 2014*, páginas 73–84. Springer International Publishing, 2015.

[12] Asv roboat, Junho 2015. URL: http://www.roboat.at/en/home/.

[13] Project: Beagle-b, Junho 2015. URL: https://www.aber.ac.uk/en/cs/research/ir/robots/beagle-b/.

[14] M. Sauzé, C. e Neal. Moop: A miniature sailing robot platform. Em *Robotic Sailing 2014*, páginas 40–53. Springer International Publishing, 2011.

[15] About the microtransat, Janeiro 2015. URL: `http://www.microtransat.org/`.

[16] International robotic sailing regatta, Junho 2015. URL: `http://sailbot.org/`.

[17] The challenge, Junho 2015. URL: `http://ubcsailbot.org/the-challenge/`.

[18] Designed to go everywhere., 2015. URL: `http://liquidr.com/technology/waveglider/sv3.html`.

[19] S. Wood. Autonomous underwater gliders. páginas 21–29, May 2008.

[20] Scarlet night's cross-atlantic journey, Junho 2015. URL: `http://www.ioos.noaa.gov/glider/welcome.html`.

[21] Zdenka. Z-gram. December 2009.

[22] Pike D. Autonomous surveillance from harbour wing, Jan 2015. URL: `http://www.maritimejournal.com/news101/onboard-systems/security-and-alarm-systems/autonomous-surveillance-from-harbour-wing`.

[23] Harbour wing - defense applications, Janeiro 2015. URL: `http://www.harborwingtech.com/products_defense.htm`.

[24] J. C. Alves e N. A. Cruz. An fpga-based embedded system for a sailing robot. Em *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, páginas 830–837, Aug 2009.

[25] J.C. Alves e N.A. Cruz. Fast - an autonomous sailing platform for oceanographic missions. Em *OCEANS 2008*, páginas 1–7, Sept 2008. doi:10.1109/OCEANS.2008.5152114.

[26] T. Curtin, J. Bellingham, J. Catipovic, e D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 6(3):86–94, 1993.

[27] Asturias M. e M. Gagen. *Desktop and Portable Systems*. Apple training series. 2007.

[28] O.W. Linzmayer. *Desktop and Portable Systems: A Guide to Supporting, Servicing, and Troubleshooting Apple Computers*. Apple training series. 2005.

[29] Sauzé C. e Neal M. *Simulating Sailing Robots*. 2011.

[30] Arduino uno, Janeiro 2014. URL: https://www.arduino.cc/en/Main/ArduinoBoardUno.

[31] N. Santos. Arduino introdução e recursos avançados. 2009.

[32] Hush little microprocessor... avr and arduino sleep mode basics, Janeiro 2015. URL: http://www.engblaze.com/hush-little-microprocessor-avr-and-arduino-sleep-mode-basics/.

[33] Ramos T. Alves, J. C. e N. A. Cruz. A reconfigurable computing system for an autonomous sailboat. Em *Proceeding of the International Robotic Sailing Conference, Breitenbrunn, Austria*, páginas 13–20, May 2008.

[34] J. C. Alves e N. A. Cruz. Metasail – a tool for planning, supervision and analysis of robotic sailboat missions. Em *Robotic Sailing 2014*, páginas 57–64. 2015.