ESCOLA NAVAL

talant de bien faire

Apolinário Ferreira Ernesto

Projecto Vent-Sup

Estudo e análise da interoperabilidade e arquitectura robótica para o projecto Vent-Sup

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

Alfeite

2020

Apolinário Ferreira Ernesto

Projecto Vent-Sup
Estudo e análise da interoperabilidade e arquitectura robótica para o projecto Vent-Sup

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

Orientação de: Professor Doutor Bruno Duarte Damas

O Aluno Mestrando, *O Orientador,*

Apolinário Ernesto

Bruno Duarte Damas

Alfeite
2020

Agradecimentos

Agradeço profundamente a todas as pessoas que tornaram esta dissertação possível. Sem elas o trabalho que desenvolvi nunca teria sido possível. Ao meu orientador, Professor Doutor Bruno Duarte Damas, por toda a paciência, dedicação em manter as minhas ideias direccionadas. Com ele aprendi lições que levarei comigo para o resto da minha carreira profissional e pessoal.

À minha família por ter dado força e coragem quando não as tinha durante o meu percurso na Escola Naval.

Aos meus camaradas da Escola Naval por toda a ajuda durante esses cinco anos e principalmente para a elaboração desta dissertação.

Resumo

O Veículo de Superfície Escola Naval (VentSupEN) é um projeto do Centro de Investigação Naval (CINAV) que visa a construção de um Veículo Não Tripulado de Superfície (Unmanned Surface Vehicle, USV), capaz de operar em qualquer ambiente marítimo.

O VentSupEN terá atuadores (motores) e diversos sensores tanto para recolha de dados ambientais como para indicar a sua posição. Os referidos sensores necessitam de trocar dados entre si. Para tal, é necessário um canal de comunicação (Middleware) entre o sistema operativo e as aplicações que interligam diferentes subsistemas compostos por sensores e atuadores.

O presente estudo apoia-se em três objetivos principais: i. estudo dos Middlewares desenvolvidos pela comunidade internacional; ii. propor uma arquitetura de software; iii. desenvolver drivers para os sensores existentes no veículo.

Assim, foram realizados estudos sobre diversos Middlewares, por forma a identificar qual seria o mais adequado a instalar no computador de bordo, com recurso a uma abordagem teórica. De seguida, é proposta a arquitetura de software com base nos sensores existentes, bem como outros sensores estudados em dissertações relacionadas com o desenvolvimento do veículo. Como prova de conceito foram desenvolvidos módulos para os drivers de sistema de localização global (Global Position System, GPS), unidade de medida inercial (Inertial Measurement Unit, IMU) e para o sistema de comunicação. Após implementação, realizaram-se testes que consistem em: comunicação com a estação de comando e controlo (C2), gestão do planeamento e do estado dos sensores.

Palavras-chave: Middleware, ROS, VentSupEN, USV

Abstract

The Naval School Surface Vehicle (VentSupEN) is a project of the Naval Research Center (CINAV) that aims to build a Unmanned Surface Vehicle (USV), capable of operating in any maritime environment.

The VentSupEN will have actuators (motors) and several sensors both for environmental data collection and to indicate its position. These sensors need to exchange data with each other. This requires a communication channel (Middleware) between the operating system and the applications that interconnect different subsystems composed of sensors and actuators.

The present study is based on three main objectives: i. study of the Middleware developed by the international community; ii. propose a software architecture; iii. develop drivers for sensors in the vehicle.

Thus, studies were carried out on several Middleware, in order to identify which would be the most suitable to install on the on-board computer, using a theoretical approach. From below, the software architecture is proposed based on the existing sensors, as well as other sensors studied in dissertations related to the development of the vehicle. As proof of concept modules have been developed for the Global Position System (GPS), Inertial Measurement Unit (IMU) drivers and for the communication system. After implementation, they were carried out consisting of: communication with the command and control (C2) station, management of planning and state of the sensors.

Keywords: Middleware, ROS, VentSupEN, USV

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

GPS	**G**lobal Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
ITU-R	International Telecommunications Union – Radiocommunication
LIDAR	Light Detection And Ranging
LoRa	Long Range
RADAR	Radio Detection And Ranging
USVs	Unmanned Surface Vehicle
UxVs	Unmanned Vehicle
WIFI	Wireless Fidelity
ROS	Robotic Operqting System
ROS-M	Robotic Operating System - Military
DUNE	Unified Navigation Envioronmen
MOOS	Mission Oriented Operation Suite
IvP	Interval Programming
CINAV	Centrode Investigação Naval
C2	Comando e Controlo
VentSupEN	Veículo de Superfície Escola Naval
ASuW	Anti-Submarine Warfare
ASW	Anti-Surface Warfare
SAR	Search And Rescue

Capítulo 1

Introdução

"*Podemos considerar os veículos de superfície não tripulados (Unmanned Surface Vehicles, USVs) um tipo de plataforma móvel inteligente na superfície da água e que é usado para executar diversas missões*" (Zhu, 2013).

Face ao exposto, podem enumerar-se algumas dessas missões: recolha de dados ambientais, hidrográficos, Proteção Portuária, Pirataria, Missões SAR, Guerra Anti-Submarina (Anti-Submarine Warfare, ASW), Guerra Anti-Superfície (Anti-Surface Warfare, ASuW), entre outras. Para atender tais necessidades com eficiência, são implementadas diversas cargas (payload) de sensores e equipamentos, por forma a cumprir com os requisitos de funcionamento para os sistemas de Controle, Comunicação, Navegação e Armas (caso tenha sido planeado com esse propósito). A utilização de USVs justifica-se também com a escassez de recursos humanos, a necessidade de obter meios de menor custo, a redução considerável de materiais logísticos e principalmente a segurança dos militares em missões de maior risco. São esses, os principais fatores, que têm pressionado as comunidades civis e militares, na aquisição dos USVs e (Unmanned Vehicles) UxVs.

O modo como essas plataformas são controladas remotamente e a interligação dos vários subsistemas existentes no veículo, tem levado a comunidade cientifica desenvolver sistemas para a gestão dos mesmos.

1.1 Objetivos

Um veículo não-tripulado é constituído por diversos sub-sistemas interdependentes entre si. Tendo como exemplo, sistema de Navegação (constituído pelos sensores de GPS, IMU, RADAR, LIDAR), sistema de comunicação, unidades de processamento, gestão de energia, entre outros. Uma vez que os sub-sistemas são interdependentes, o veículo necessita de um sistema que faça a gestão e interligação dos mesmos.

O Middleware é um software que está entre o sistema operativo e outras aplicações a ser executados no computador(Microsoft, 2020). Atua como um túnel para a troca de informação entre as aplicações ou entre os sub-sistemas dos veículos não-tripulado. O tema em estudo tem como objetivo principal desenvolver a arquitetura de software que correr no computador de bordo do VentSupEN.

Desta forma, pretende-se:

1. Estudar e analisar as diversas plataforma de Middleware já existente e implementar uma delas;
2. Propor e desenvolver uma arquitetura de software para o veículo;
3. Implementar alguns módulos dessa arquitetura, com base no trabalho já desenvolvido previamente no âmbito deste projeto. Em particular, serão desenvolvidos drivers para acesso à informação proveniente de alguns sensores (GPS e IMU) e será estabelecido o sistema de comunicação com os outros veículos ou com a estação de comando e controlo, usando quer WIFI quer comunicação LoRa.

1.2 VentSupEN

O VentSupEN é um veículo de superfície não-tripulado para a Escola Naval, cujo arranque teve lugar em janeiro de 2016 e até a data da elaboração desta dissertação encontrava-se em construção no Arsenal do Alfeite S.A. O veículo tem 3.5m de comprimento e um calado de 0.31m, possui dois motores Torqueedo Cruise 2.0R de 7,44 kW de potência útil equivalente, sendo expectável uma velocidade máxima de 13 nós.

Desenhado para uma autonomia de 2 horas e percorrer uma distância máxima de duas milhas, permitindo desempenhar as missões de Levantamento Hidrográfico, Reconhecimento Portuário e Costeiro, entre outros(da Costa, 2019).

Após a sua construção, a plataforma ficará a cargo da Escola Naval a qual irá apoiar no ensino e investigação em diversas áreas das Ciências Militares Navais.

FIGURA 1.1: Projeto VentSupEN

1.2.1 Trabalhos Relacionados

O Projeto para além do trabalho exposto na presente dissertação de mestrado, abrange estudos nas áreas de navegação, sistema de propulsão e governo de USVs, sistema de produção, distribuição e armazenamento de energia. Bem como trabalhos já desenvolvidos por outros alunos da Escola Naval, mencionados a seguir.

A dissertação com o tema "Projeto de um Veículo Não-Tripulado de Superfície (VentSupEN) Modular Multi missão", apresentada pelo ASPOF EN-MEC Rui Nuno Pereira Pinto da Costa, incidiu no projeto do casco do veículo em forma de catamaran, a mesma faz alusão ao estudo de estabilidade e resistência estrutural em várias condições de ondulação, definiu o sistema de propulsão do veículo e delimita o peso total que o veículo poderá suportar.

Na área de Comunicação, os estudos foram realizados pelo ASPOF EN-AEL Tiago Vieira Rodrigues com a dissertação "Estudo e Projeto do Sistema de Telecomunicações e Sensores para um Veículo Autónomo". O tema, centrou-se na escolha e aplicação dos sensores necessário para a navegação, sistema de comunicação bem como o dimensionamento das antenas para as frequências de 433 MHz – LoRa e 2.4GHz – WIFI.

1.3 Estrutura da Dissertação

Para atingir os objetivos supracitados, teve-se a necessidade de estruturar a presente dissertação em seis capítulos. No primeiro capítulo, é feito um enquadramento geral do projeto. No segundo capítulo é feito o levantamento do estado da arte neste tema, apresentando e analisando as arquiteturas de software robótica existente, o terceiro capítulo apresenta uma possível arquitetura de software para o veículo. A descrição do código desenvolvido e a sua lógica de funcionamento ao encaminhar os dados dos drives até o supervisor e posteriormente enviado para Estação de Comando e Controlo (C2) (capítulo 4). Para finalizar, apresentarei os resultados dos teste (capítulo 5) e as respetivas conclusões.

Capítulo 2

Estado da Arte

O conceito de interoperabilidade é fundamental para o funcionamento de uma entidade do sistema (Carapau et al., 2017), deste modo é importante definir alguns conceitos que são fundamentais para a compreensão deste trabalho. No presente capítulo são estudadas diferentes arquiteturas de software robótica bem como suas vantagens e desvantagens. Serão analisados e caracterizados os sistemas de bordo de diversos USVs, começando com alguns USVs desenvolvidos por universidades portuguesas, nomeadamente UCAP, ROAZ II e o DELFIM e depois passando para outros USVs desenvolvidos por outras marinhas.

Por fim, será feita uma breve comparação dos softwares robóticos, o Middleware a usar no projeto VentSupEN com base em outros projetos desenvolvidos pelas universidades.

2.1 Conceitos Fundamentais

O Institute of Electrical and Electronics Engineers (IEEE) define a interoperabilidade de sistemas como: A capacidade de dois ou mais sistemas ou componentes de trocar informações e usar as informações trocadas (I. of Electrical & Engineers, 1990).

Standard, define as características, como dimensões, aspetos de segurança ou desempenho, de um produto ou serviço (T. I. of Electrical & (610), 1990). Em geral, os standard, definem um conjunto de regras e modelos que um sistema usa ou deve ter.

Framework, De acordo com o Information Technology Standards and Organizations Glossary, (Target, 2020) é um conjunto de ferramentas ou modelo (template) com diversas funções que podem ser usadas para o desenvolvimento de softwares . Na opinião de (Marques, 2018), o Framework é diferente das camadas de

standards e o mesmo facilita o uso de destas standards.

Denomina-se por Middleware :"*Um software usado para interligação de aplicativos e outras ferramentas diferentes, encontra-se alocado entre sistema operativo e outros aplicativos. O software, fornece métodos de comunicação e gerenciamento de troca de dados*" (T. I. of Electrical & (610), 1990).

TCP : "É um dos protocolo de comunicação de dados, da camada de transporte de rede de computadores do modelo OSI[1]" (Target, 2020).

UDP : (User Datagram Protocol) é um protocolo de comunicação alternativo ao nível do TCP, é usado para estabelecer comunicações entre os aplicativos e a internet de baixa latência e tolerância. Ele permite que a aplicação envie um datagrama encapsulado num pacote IPv4 ou IPv6 a um destino, porém sem qualquer tipo de garantia que o pacote chegue corretamente.

Telecomunicação, segundo o ITU-R[2] , é qualquer transmissão, emissão ou receção de sinais, escrita, imagens ou informação de qualquer género por fio, radio, ótico ou outro sistema eletromagnético.

Arquitetura publish/subscribe é um padrão de mensagens onde os editores transmitem mensagens, sem qualquer conhecimento dos subscritores. Do mesmo modo, os subscritores "ouvem"as mensagens relativas ao tópico/categorias que lhes interessam sem qualquer conhecimento de quem são os editores (Possum, 2015).

Estação de Comando e Controlo (C2), "pode ser definido como o processo de direção por pessoa ou autoridade legalmente ou legitimamente investida na utilização dos recursos colocados à disposição. Embora possa ser utilizada em ambientes civis, normalmente está associada ao meio militar".

[1]Do inglês: Open System Interconnection
[2]Do inglês: International Telecommunications Union – Radiocommunication Sector

2.2 Arquiteturas de Software Robótica para USV

Segundo (David Shaw, 1996), definem que arquitetura de software é um conjunto de componentes computacionais e os relacionamentos entre esses componentes. Para os USVs é quase inviável desenhar uma arquitetura sem pensar no middleware a usar. O Middleware possibilita a comunicação entre aplicações ou hardware e software no caso dos veículos, tem como objetivo diminuir a complexidade dos diversos sistemas existentes.

Esta secção apresentas as vantagens, desvantagens e a arquitetura de alguns middleware.

2.2.1 Unified Navigation Envioronment (DUNE)

DUNE: É um ambiente de tempo real para sistemas não tripulados a bordo de software. É usado para escrever software genérico incorporado no coração do sistema, por exemplo, código ou controle, navegação, comunicação, acesso a sensores e atuadores, etc. Ele fornece uma camada de abstração de plataforma independente do sistema operativo e da arquitetura, escrita em C++, melhorando a portabilidade entre diferentes arquiteturas de CPU e sistemas operativos (LSTS, 2013).

O DUNE utiliza a arquitetura publish/subscribe para fornecer um acoplamento solto entre módulos, que no DUNE são chamados de tarefas. As tarefas DUNE publicam e subscrevem mensagens sem conhecerem quaisquer detalhes sobre as outras tarefas. Efetivamente, a passagem de mensagens é o único mecanismo disponível para trocar informações entre as tarefas. Por exemplo, uma tarefa que interage com um sensor produz uma mensagem de um determinado tipo com uma leitura do sensor (por exemplo, aceleração), que pode mais tarde ser consumida por outra tarefa que integra essa informação e produz uma estimativa do estado (por exemplo, posição estimada), outra tarefa pode consumir essa estimativa e produzir comandos para os atuadores (por exemplo, aumentar o impulso). A Figura 2.1 ilustra este conceito(Pinto et al., 2013).

Desenvolvido pela LSTS (Underwater Systems and Technology Laboratory) da Universidade do Porto, usa o protocolo IMC (Inter-Module Comunication) para a sua comunicação e assim proceder com o envio das mensagens. As tarefas, são executadas entre si por tópicos ou processos separados, na troca de informação usa o mecanismo de barramento de mensagens. Este mecanismo apresenta como vantagens:

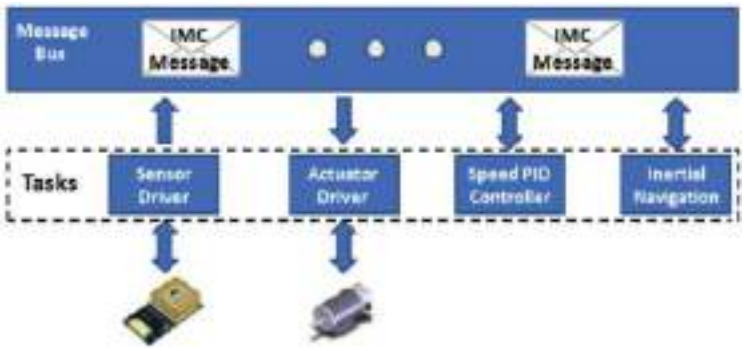


FIGURA 2.1: Representação simplificada do conceito de passagem de mensagens no DUNE

- Deteção e sinalização de Falhas;
- Retransmissão automática de Mensagens;
- Configuração Simples;
- Independente do CPU e SO.

Este protocolo apresenta contudo algumas desvantagens (Marques, 2018):

- O protocolo não é de fácil acesso;
- Não tem muitos veículos que operam com este protocolo.

2.2.2 Mission Oriented Operation Suite - Interval Programming (MOOS-IvP)

O MOOS-IvP é um Middleware de código aberto (open source) baseado em arquitetura publish/subscribe, este Middleware fornece um conjunto de pacotes em C++ de forma dar autonomia em plataformas robóticas, essencialmente nos veículos autónomos de superfície e sub-superfície (Joseph, 2018).

O MOOS-IvP é composto por duas partes distintas, MOOSDB (MOOS Database) e aplicações MOOS (MOOS application). O MOOSDB atua como servidor, é responsável por toda comunicação entre aplicações MOOS que atuam como clientes, significa que todas as comunicações entre aplicações ou nós (que pode ser uma rotina ou sensor) acontece apenas via MOOSDB (Figura 2.2)

A aplicação adicional IvP (Interval Programming), usa a otimização multi-objetivo para implementar a coordenação autónoma de comportamento na plataforma. O IvP usa um paradigma de co-piloto (o controle do veículo é separado da

autonomia), aplicação publica a melhor ação do veículo no MOOSDB com base nas informações de navegação e estado do veículo (Spears, West & Collins, 2013).(Figura 2.3).

O MOOS (Mission Oriented Operating Suite) é um produto do Mobile Robotics Group da Universidade de Oxford e fornece os principais recursos do Middleware e usa topologia em estrela, enquanto o IvP (Interval Programming) desenvolvido pelo MIT, com uma arquitetura baseada em comportamento.

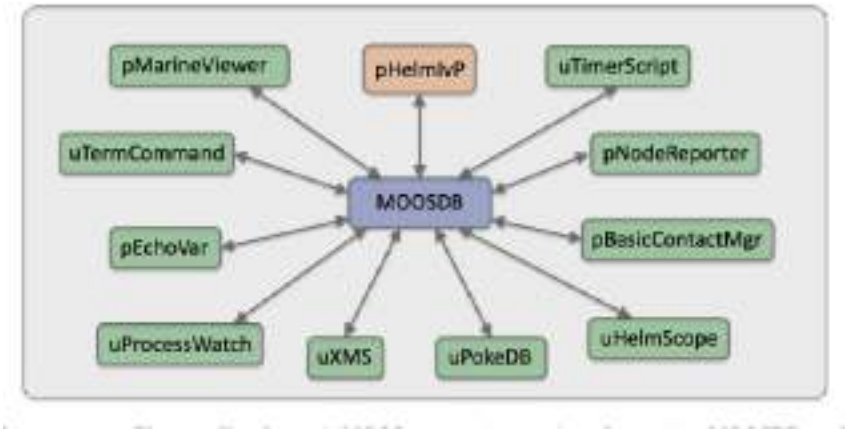


FIGURA 2.2: MOOSDB and MOOS application

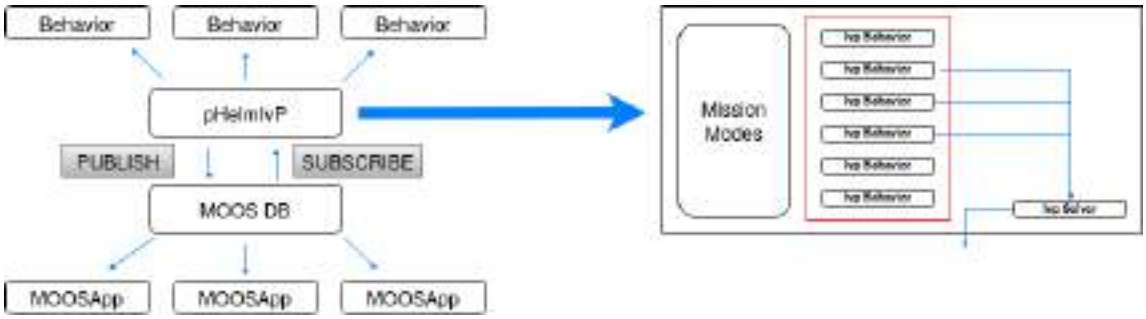


FIGURA 2.3: Modelo do pHelmIvP

O MOOS, tem vantagem com a topologia em estrela. Sendo que não interessa a quantidade de nós ligados no servidor (MOOSDB) ele continua a ser simples, os nós/aplicações MOOS ou clientes não precisam conhecer os outros, cada nó tem um nome exclusivo e o servidor é responsável por gerir toda a entrega de informação. De certo modo, essa vantagem passa a ser também uma desvantagem devido a topologia centralizada, o que pode vir a ser vulnerável ao " bottle-necking" (vulnerável ao congestionamento) (Marques, 2018). Outra desvantagem é como os dados são

transmitidos entre o MOOSDB e um cliente. O significado dos dados depende da função do cliente. No entanto, a forma dos dados é limitada pelo MOOS.

O MOOS permite que os dados sejam enviados apenas em cadeia ou em formato duplo. Os dados são compactados em mensagens (classe CMOOSMsg) que contêm outras informações destacadas(Kim et al., 2017).

2.2.3 Robotic Operating System (ROS)

O ROS foi originalmente desenvolvido em 2007 pelo Laboratório de Inteligência Artificial de Stanford (SAIL) com o apoio do projeto Stanford AI Robot (Marques, 2018), desde essa data o ROS tem recebido vários contributos da comunidade e atualizações no seu código, encontrando-se atualmente na versão 2.0.

É um Middleware de código aberto baseado em Linux para o desenvolvimento de aplicações robóticas, mantido pela (Open Source Robotics Foundation, OSRF). Um dos objetivos deste sistema é facilitar a criação de novas aplicações para robôs, através da exploração de bibliotecas, algoritmos e componentes de hardware. Seu principal objetivo é maximizar a reutilização dos dados de sensores já disponíveis, fusão de sensores e algoritmos de controle. (Yoshida et al., 2015).

Tal como DUNE e MOOS-IvP, o ROS também usa a arquitetura publish/-subscribe de mensagens, no entanto, o ROS tem uma grande vantagens devido à topologia que usa, ponto a ponto (Peer-to-Peer). Esta topologia, requer algum tipo de mecanismo de busca para permitir que os processos se encontrem uns aos outros em tempo de execução e a este serviço chamamos de Master (roscore/rosmaster). Basicamente, o ROS é dividido por: nós, tópicos, mensagens e serviços.

Um sistema baseado neste middleware é composto por vários nós que comunicam entre si e pode ser implementado em vários computadores. Podemos considerar os nós como um determinado sensor ou bloco de código, como mostra a figura 2.4.

Por exemplo, no projeto VentSupEN temos o módulo de navegação, que é constituído pelos sensores de GPS e IMU. Cada sensor no sistema ROS será interpretado como um nó, nó esse onde estarão definidas regras para a recolha, transformação e publicação dos dados. Esses nós são independentes e geralmente por si só não demonstram grande utilidade, no entanto, eles precisam de uma forma de comunicação entre eles e este mecanismo de comunicação são os tópicos e serviços.

Os Tópicos, são identificadores únicos que criam um canal de comunicação entre os nós de forma a transmitir um fluxo de mensagens com um tipo definido. Por exemplo, os dados do sensor GPS podem ser enviados em um tópico chamado gpsPosition, com um tipo de mensagem float16 ou String, enquanto os dados do IMU podem ser enviados sobre um tópico chamado imuRaw, com o tipo int8.

Os tópicos implementam a arquitetura "publish/subscribe", que é uma das maneiras mais comuns de trocar dados em sistemas distribuídos. Antes dos nós começarem a transmitir dados sobre tópicos, eles devem primeiro anunciar ou advertir o nome do tópico e os tipos de mensagens que serão enviadas. Em seguida, eles podem começar a enviar ou publicar os dados reais sobre o tópico. Os nós que desejam receber mensagens de tópico podem se inscrever nesse tópico, solicitando o roscore (Joseph, 2018).

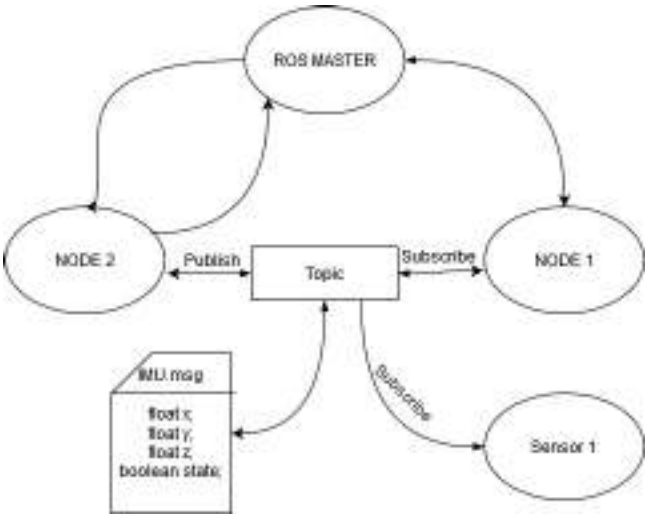


FIGURA 2.4: Funcionamento do ROS

Os serviços são outra maneira de comunicação entre nós no ROS. Os serviços são apenas chamadas de procedimento remoto síncronas; permitem a um nó chamar uma função que executa noutro nó. Definimos as entradas e saídas desta função de forma semelhante à forma como definimos novos tipos de mensagens. O servidor (que fornece o serviço) estabelece uma chamada de retorno para lidar com o pedido de serviço e anuncia o serviço. O cliente (que chama o serviço) acede então a este serviço através de um proxy local. As chamadas de serviço são bem adaptadas a coisas que só precisam de ser feitas ocasionalmente e que demoram um período de tempo limitado a completar (Joseph, 2018).

As mensagens ROS são definidas por ficheiros especiais de definição de mensagens. Estes ficheiros são então compilados em implementações específicas de linguagem que podem ser utilizadas no código, significa que, mesmo que esteja a utilizar uma linguagem interpretada como Python, precisamos executar o comando "catkin-make"[3] caso precisemos definir os próprios tipos de mensagens. Caso contrário, a implementação específica da linguagem não será gerada, e Python não será capaz de encontrar o seu novo tipo de mensagem (ROS, 2020a).

O ROS como middleware, tem como a sua principal vantagens a sua topologia de rede ponto a ponto, evitando a centralização e o congestionamento de informação no master (roscore), para sistemas mais robustos, está vantagem ainda permite que a relação entre os nós e os tópicos seja de muitos para muitos. Sendo que um tópico pode ser subscrito por vários nós e um nó pode publicar vários tópicos. Uma outra vantagens de destaque é o facto de que pode ser escrita em diversas linguagens, atualmente em C++, Java, LISP, MATLAB, Python, JavaScript, Ruby, R, Julia, Haskell (Quigley, Gerkey & Smart, 2015), deixando a critério dos programadores a linguagem que mais se adequa ao projeto.

Como desvantagens, podemos apontar a sobrecarga do sistema de mensagens, que não é tão compacta quanto os outros sistemas, e isso pode ser um problema em grandes sistemas com muitos tópicos e serviços (Marques, 2018).

A arquitetura de referência proposta nesta tese é fortemente influenciada por este padrão.

2.2.4 Robotic Operating System - Military (ROS-M)

Normalmente, o software de defesa funciona em ambientes rigorosamente controlados, onde todos os componentes são geridos diretamente pelo integrador do sistema ou pelo contratante principal (Ernst, Kazman & Bianco, 2018).

ROS-M é um middleware para sistemas autónomos e robóticos (Robotics and Autonomous Systems, RAS) militares baseado em ROS com o objetivo de suprimir as necessidades únicas dos robôs militares. Como o ROS o ROS-M também fornece software personalizado para perfis de interoperabilidade (Interoperability Profiles, IOP), ferramentas de cibersegurança e processos específicos para aplicações de robótica de defesa, adição de simuladores mais robustos e aproveitar os ricos

[3] É uma ferramenta de linha de comando que combina os comandos cmake e make para executar o projecto e executar os pacotes necessários do projecto

repositórios do ROS. As necessidades militares são diferentes das empresas de robótica comercial e dos investigadores, pelo que o ROS-M acrescenta formas de garantir que o software é mais fiável. O ROS-M promove partilha e reutilização de códigos, especialmente para componentes cuja distribuição é restrita devido a preocupações de segurança nacional ou de controlo de exportação.

A integração de um componente requer uma compreensão completa das especificações API do mesmo, significa que o integrador necessita ter um conhecimento amplo do componente a integrar no sistema o que torna uma desvantagem para o projeto VentSupEN. Em vez de concentrar-se em ferramentas para recolher informação, encontra desafios de investigação como (Ernst, Kazman & Bianco, 2018):

- Identificação correta e precisa e modelação do conjunto de componentes e interações num determinado sistema;
- Identificação e medição de indicadores robóticos e de ambiente operacional relevantes para os atributos de qualidade do sistema;
- Agregação destes indicadores para apoiar uma análise trade-off (troca) eficaz.

2.3 Veículos de Superfície não Tripulados

Os veículos de superfície não tripulados têm sido utilizados frequentemente em diversas missões, para tal necessitam de ter determinadas características em seus sistemas embarcados para o cumprimento das mesmas. São citadas as principais características de alguns USVs que podem ser relevante no âmbito do projeto, começando por descrever os desenvolvidos pelas Universidades Portuguesas.

2.3.1 Unmanned Capsule (UCAP)

Com o avanço da ciência e das tecnologias, os institutos têm aproveitado desenvolver meios que ajudam salvar a vida humana no mar neste caso, a UCAP é um USV desenvolvido pela INESCTEC[4] com o objetivo de dar assistência direta aos náufragos, fornecendo-lhes meios de flutuação e proteção térmica. Estes meios consistem em balsas salva-vidas, implantadas e insufladas automaticamente perto das vítimas, fornecendo mais seguranças e uma assistência rápida aos náufragos.

A bordo da UCAP encontra-se instalada duas baterias de 12V 14Ah fornecendo uma energia total de 336Wh. O seu sistema de navegação é baseado em um

[4]Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

modulo PNI Trax AHRS com um IMU [5] e um u-blox NEO 6T GPS[6]. Quando ao sistema de comunicação é via Wi-Fi, usando um router WRT54 conectado a uma antena L-Com de 7dB (Matos et al., 2013).

O software embarcado é composto por vários módulos que comunicam entre si usando um mecanismo de passagem de mensagens. Estes módulos seguem uma arquitetura hierárquica semelhante à utilizada em outros sistemas robóticos do INESCTEC. DUNE é o Middleware que corre em um kernel Linux e é composto por um conjunto de processos independentes. Desta forma, não só a modularidade e robustez do sistema são aumentadas, mas também a sua depuração e recuperação de eventos inesperados é muito mais simples. A comunicação entre os módulos depende da troca de mensagens, usando uma conexão não orientada na troca de dados (protocolo UDP), com redução das despesas de processamento, conforme requerido neste tipo de aplicações (Cruz & Matos, 2008).

Figura 2.5: USV UCAP

2.3.2 ROAZ II

Os USV 's médios são normalmente veículos maiores que requerem sistemas de implantação mais complexos, como guindastes. Podem ser tele operados ou capazes de seguir rotas pré-definidas. Os sensores a bordo mais comuns, além dos usados em USV leves (IMU, GPS, LIDAR), são câmaras térmicas, dispositivos de comunicação e equipamentos de deteção química, biológica, radiológica e nuclear (Marques, 2018). Alguns desses exemplos são o ROAZ II.

[5]Desenvolvido pela PNI, https://www.pnicorp.com/product/trax/
[6]Desenvolvido pela U-Box, https://www.u-blox.com/en/product/neo-6-series

ROAZ II é o nome do veículo de superfície autónomo desenvolvido pela Laboratório de Sistemas Autónomos (LSA) pertence ao Instituto Superior de Engenharia do Porto (ISEP). Este veículo é um catamarã com propulsão elétrica, desenvolvido para operar em mar aberto e desempenhar vários tipos de missões, graças à estrutura entre cascos. As principais missões para que foi desenhado são operações oceanográficas, recolha de dados ambientais, operações de busca e salvamento e de defesa portuária (Rodrigues, 2019).

Devido à missões que pode ser adaptado e quantidade de sensores bem como outros equipamentos existente a bordo é necessário um software que faça a gestão de toda a informação. O middleware implementado no seu computador de bordo foi o ROS, com o seu código escrito em C++.

Figura 2.6: USV ROAZ II

2.3.3 DELFIM

O DELFIM é um USV desenvolvido no ISR / IST para aquisição automática de dados marítimos e para servir como um relé acústico entre embarcação submersa e uma embarcação de apoio (Alves et al., 2006). O USV também pode ser usado como uma unidade autónoma, capaz de manobrar e realizar um percurso preciso ao mesmo tempo em que realiza a aquisição e transmissão automática de dados marítimos para um centro operacional instalado a bordo de um navio de apoio ou em terra. Isto está de acordo com a tendência atual de desenvolver sistemas para reduzir os custos e melhorar a eficiência da operação de navios oceanográficos no mar (Manley et al., 2000).

O Sistema de Controlo de Missão desenvolvido é suportado numa arquitetura informática distribuída. Os processos distribuídos (tanto dentro de um único veículo ou através de vários veículos) são coordenados usando comunicação interprocesso / intercomputador e mecanismos de sincronização implementados sobre CAN

Bus e Ethernet, usando o Protocolo Internet (IP) e outros protocolos de comunicação proprietários. Esta arquitetura de computador distribuído é projetada em torno de PCs (PC104) executando o sistema operacional Windows Embedded NT com micro controladores de 8 e 16 bits (como o Siemens C509L e o Philips XAS3) que se comunicam usando o controlador padrão Intel 82527 Controller Area Network (protocolo CAN 2B)[7].

Todas as placas micro-controladoras foram desenvolvidas no IST/ISR com o objetivo de atender aos rigorosos requisitos de consumo de energia, fiabilidade e custo. A implementação efetiva dos blocos de construção acima referidos é feita recorrendo a uma poderosa linguagem de descrição da rede Petri denominada CORAL+ (propriedade do IST/ISR).

FIGURA 2.7: USV DELFIM

2.3.4 CAT-Surveyor

O CAT-Surveyor é um veículo de superfície não tripulado em forma de catamarã desenvolvido pela (Tech, 2015), com modos tele-operados e/ou autónomos para aquisição de dados hidrográficos ou vigilância de zonas subaquáticas em portos, áreas costeiras e águas interiores.

Graças à sua arquitetura aberta e à sua comunicação de alta velocidade de PC para PC, todos os tipos de sensores executados em Windows podem ser facilmente integrados no CAT-Surveyor. O PC de controlo em terra permite a visualização e controlo em tempo real dos sensores de navegação e de bordo.

[7]O CAN é um protocolo de comunicação serial síncrono. O sincronismo entre os módulos conectados a rede é feito em relação ao início de cada mensagem lançada ao barramento (evento que ocorre em intervalos de tempo conhecidos e regulares)

Figura 2.8: USV CAT-Surveyor

2.3.5 HYCAT

Atualmente, os USVs têm sido bastantes úteis em pesquisas cientificas no âmbito civil, especialmente no estudo da qualidade da água e velocidade da água, sendo essas as principais missões do HYCAT. Desenvolvido pelo (YSI, 2020), este USV tem a forma de catamarã, de reduzidas dimensões, próprio para operar em águas calmas que permite, fazer análise, levantamento e transmissão de dados.

Figura 2.9: USV HYCAT

Na tabela a baixo, estão citadas algumas características do USVs relativamente o software de bordo, sendo que a segunda linha indica quais middleware usam para interoperabilidade de informação, os protocolos de comunicação estão identificados na terceira linha, já a quarta linha indica linguagens de programação usada no desenvolvimento do software, por último, são identificados os sistemas operativos.

- (a) - Informação não disponível

USVs	**UCAP**	**ROAZ II**	**DELFIM**	**CAT-Surveyor**	**HYCAT**
Middleware	DUNE	ROS	COBRAL+	(a)	(a)
Protocolo	IMC-UDP	TCP	CAN 2B	(a)	(a)
Linguagem	C++	C++	Rede de Petri	(a)	(a)
SO	Linux	Linux	Windows NT	Windows	Windows 10

Tabela 2.1: Características dos USVs Civis

2.3.6 MAST

O MAST (Maritime Autonomy Surface Testbed) 13 é um sistema de alta velocidade com de 13m de comprimento, capaz de uma navegação totalmente autónoma. O USV utiliza o sistema de controle autónomo proprietário da ASView da L3Harris e algoritmos avançados desenvolvidos para o Laboratório de Ciência e Tecnologia de Defesa do Reino Unido (Defense Science and Technology Laboratory, DSTL). O Veículo consegue navegar de forma autónoma, para além da linha de vista, para efetuar tarefas de reconhecimento e patrulhamento (L3Harris, 2019).

Figura 2.10: USV MAST

2.3.7 Katana

O veículo de superfície não-tripulado Katana, foi desenvolvido em 2014 para a Marinha de Israel por forma a cumprir com missões de proteção de força, proteção portuária, combate a fogos em portos, busca e salvamento, vigilância e operações anti terrorismo. O veículo tem 11.9m de comprimento, estando projetado para operar maioritariamente em regiões costeiras. Possui dois motores diesel de 560 cavalos que lhe permitem alcançar os 60 nós de velocidade máxima com o seu sistema de hidrojato. O valor máximo da payload equipável no veículo é de 2200 kg, tendo este um deslocamento de cerca de 6 toneladas.

Quanto ao comando e controlo, o veículo pode ser controlado de forma autónoma por um computador de bordo conectado a sistemas Radar, AIS e payloads eletro-óticas por forma a evitar qualquer colisão quando o veículo se encontra a ser operado. Além disto, também é possível controlar o veículo remotamente através de Radio Link com um operador humano (Recognition, 2019).

Figura 2.11: USV Katana

2.3.8 Hovercraft

O hovercraft não-tripulado Hov Pod e os sistemas USV fornecem recursos de superfície estendidos para muitas aplicações comerciais e militares importantes, operando de forma integrada em terra, em qualquer terreno plano. Quando integrado com sistema UAV compatível, esta plataforma oferece a solução de vigilância de superfície mais versátil do mundo (Pod, 2019).

Os sistemas Hov Pod USV utilizam um módulo de sensores para tarefas de observação a bordo da embarcação, integrado com dados de sensores do sistema

de navegação existente. Isso pode ser um radar e um AIS (sistema de identificação automática), combinados com câmaras de luz natural e infravermelhas.

Os USVs Hov Pod podem utilizar um sistema de navegação autónomo que segue um plano de navegação predefinido, mas com certo grau de liberdade para ajustar o planeamento de acordo com a legislação e as regras de navegação e integrar um sistema de vigilância, dando maior autonomia nos sistemas autónomo.

FIGURA 2.12: USV Hovercraft

2.4 Comparação

Neste capítulo foram abordados os diferentes middleware bem como diferentes projetos de USVs tanto nacional como internacional. Tendo em vista os objetivos supracitados fez-se uma analise dos middleware quanto a topologia, arquitetura e linguagens de desenvolvimento. Percebe-se que a maioria partilha o mesmo tipo de arquitetura (publish/subscrible). A linguagem C++ é comum em todos, mas, o ROS destaca-se relativamente aos outros permitindo que programadores utilizem diversas linguagens no seu desenvolvimento.

Na tabela 2.1 é apresentado um resumo sobre as características dos USVs civis, em USVs nacionais a maior parte dos seus sistemas embarcados têm o linux em seu Kernel. A principal diferença em termos de sistemas entre os USVs está no seu protocolo de comunicação no sistema. A diferença básica entre o UDP e o TCP é o facto de que o TCP é um protocolo orientado à conexão e, portanto, inclui vários mecanismos para iniciar, manter e encerrar a comunicação, negociar tamanhos de pacotes, detetar e corrigir erros, evitar congestionamento do fluxo e permitir a retransmissão de pacotes corrompidos, independente da qualidade do meio físicos.

Em conclusão, o ROS tem uma variedade de vantagens em comparação com outras estruturas. As principais são o facto de que este é um sistema modular, composto por nós, e os nós podem ser facilmente alterados, se necessário, ou alternados sem grandes alterações ou problemas de compatibilidade. É também

um sistema de código aberto, que permite ao programador ter vários pacotes disponíveis, desenvolvidos por outros pesquisadores/programadores. Além de ter uma ampla comunidade de desenvolvimento o middleware pode ser escrito em diferentes linguagens de programação.

Capítulo 3

Arquitetura do Sistema

O presente capítulo apresenta as diferentes arquiteturas de software e estrutura eletrónica do veículo tanto para todos os equipamentos existentes como para os equipamentos futuros definidos nas dissertações dos ASPOF's Vieira Rodrigues e Pinto da Costa. Os veículos não tripulados em particular os USV, dependendo das aplicações práticas precisam de vários sensores e atuadores (motores) para desempenharem diversas missões. Existem sensores e sistemas de base que devem ser incluídos nos USV's. Como exemplificado na Figura 3.1.

3.1 Componentes básicos em USV's

O GPS: é um sistema de navegação por satélites que fornece informações de localização e hora do equipamento/veículo independentemente das condições meteorológicas. A taxa de erro para localização do equipamento vária de acordo com o número de satélites visíveis. Este sistema usa uma constelação de 4 à 24 satélites para fornecer a localização (de Abreu Faria, de Melo Silvestre & Correia, 2016). O sistema, utiliza as distâncias entre os satélites e o equipamento/veículo para triangulação e obtenção da posição (El-Rabbany, 2002).

O IMU: é um importante sensor utilizado em sistemas de navegação para veículos não tripulados, submarinos e aeronaves. O equipamento calcula a orientação em três dimensões, aceleração, posição e a velocidade. O referido sensor, quando possível, permite a integração com o GPS dando maior otimização de localização dos veículos referidos (Mahmoud & Atia, 2019).

RADAR Radio Detection and Ranging (RADAR): é um sistema que permite detetar alvos a grandes distâncias, com aplicações diversas tais como: deteção de inimigos, controlo de tráfego aéreo, identificação de objetos, deteção da velocidade de veículos. Este sistema utiliza antenas direcionais para emitir ondas eletromagnéticas

e quando refletida indica a posição do objeto ou alvo que a refletiu no sistema (Melvin & Scheer, 2014).

LIDAR: (Light Detection and Ranging) "*é um sistema, que emite intenso, focado, feixes de luz e mede o tempo necessário para que as reflexões sejam detetadas pelo sensor. Esta informação é utilizada para calcular intervalos, ou distâncias, a objectos*". O processo só é possível graças a união com os sensores GPS e IMU, o que permite a criação de uma nuvem de n pontos geograficamente referenciados, dando a possibilidade de construir mapas com múltiplas camadas (Miltiadou et al., 2015).

CÂMARA EO/IR: "*é um sistema que contém sensores capazes de captar radiação eletromagnética e transformá-la numa imagem. O espetro eletromagnético está dividido em várias gamas de radiação. Ainda que o ser humano seja apenas sensível à radiação visível, existem outras gamas de radiação passíveis de ser usadas por ele com recurso a algumas ferramentas de ajuda. A radiação infravermelha é visível e mais utilizada em imagiologia e vigilância*" (Belbachir, 2010).

SONAR (Sound Navigation and Ranging): tal como o RADAR e o LIDAR este equipamento também mede distância ao alvo mas, em vez de usar ondas eletromagnéticas usa ondas sonoras que se propogam no meio aquático para localizar o alvo. O SONAR pode ser ativo, que emite ondas sonoras para localizar objetos/alvos e passivo que usa o som que os objetos/alvos emitem no meio aquático para os localizar.

A sonda é um equipamento utilizado em embarcações com o objetivo de medir profundidades das águas, desde o local onde se encontra a sonda até ao fundo do mar. Este sensor utiliza ondas sonoras que se propagam no meio aquático, ou seja, o tempo total gasto desde a transmissão até à receção do eco equivale à distância percorrida, conhecida a velocidade do som na água (R.Sathishkumar, 2013).

A fiabilidade dos sistemas de comunicação com estações de controlo terrestres e outros veículos é de grande importância para o controlo cooperativo, mas também, comunicação a bordo com ou sem fios com uma variedade de sensores, atuadores e outros equipamentos. Os sensores IMU e GPS auxiliam no posicionamento do veículo fornecendo os dados essenciais pelos canais de comunicação disponíveis com a estação de comando e controlo.

O sistema permite monitorizar e operar o USV em diferentes condições meteorológicas e verificar o estado dos equipamentos (como por exemplo, temperatura, humidade a bordo e consumo de energia). Sistemas de propulsão e energia, dirigido

por dois motores independentes acoplados no casco ambos alimentados por cada fonte de alimentação (baterias).

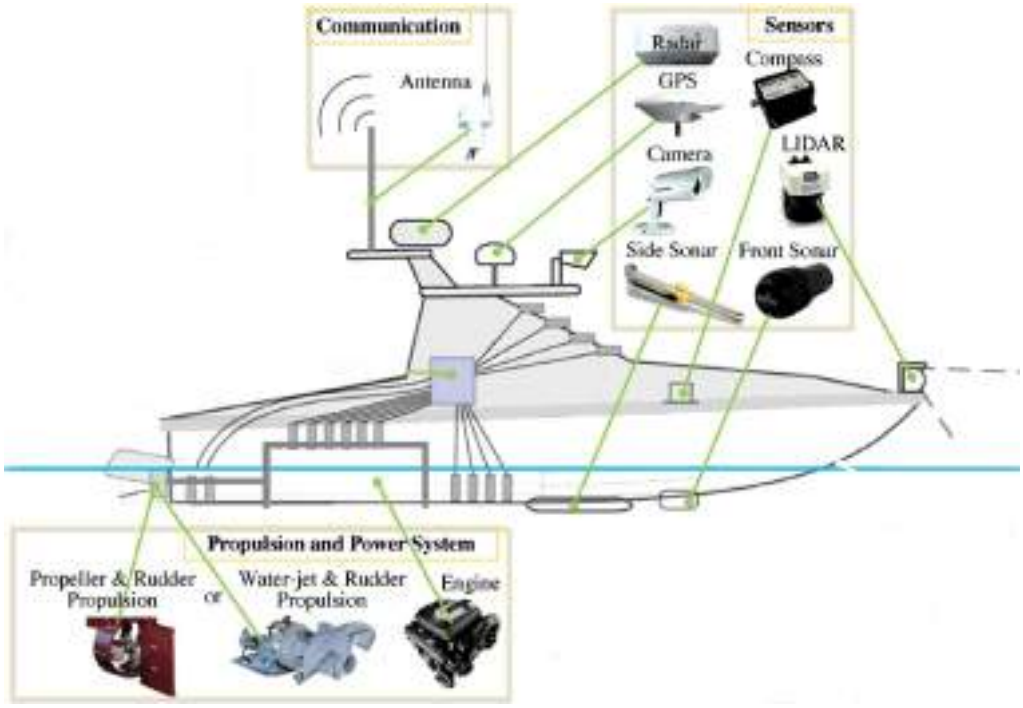


FIGURA 3.1: Sistemas básicos dos USVs

Um outro elemento que não é apresentado na figura 3.1 mas que se deve ter em conta é a estação de comando e controle (C2), que pode ser localizada numa instalação em terra, num veículo móvel ou num navio. Em geral, as missões são atribuídas a USV 's através de sistemas de comunicação sem fios.

3.2 Estrutura Eletrónica

A arquitetura eletrónica existente e a proposta apresentada nesta dissertação, teve como premissas os diversos sensores e equipamentos estudados pelos ASPOF 's Vieira Rodrigues e Pinto da Costa nas suas respetivas dissertações.

3.2.1 Estrutura Eletrónica Proposta

A proposta apoia-se nos sensores, equipamentos já previstos para o projeto, mas que até a data não foram adquiridos ou não se encontravam disponíveis. (Ver Apêndice A). Os equipamentos serão descritos em módulos seguindo a mesma lógica apresentada nas dissertações citadas acima.

Módulo Propulsão e Energia

O sistema de propulsão terá dois motores elétricos Torqueedo Cruise 2.0R (cada alimentado por uma bateria de 12V 35Ah). Os referidos motores, estarão conectados nos arduino e estes ligados no RPY via USB. Para o sistema de energia elétrica, é sugerido que tenha duas linhas de distribuição, a principal dimensionada para distribuir corrente a todo veículo durante a sua operação, a secundária com uma corrente mais baixa, alimentará apenas os equipamentos considerados essenciais para navegação, *Raspberry Pi (RPY)*, ESP32 e a câmara Flir Duo. O propósito na escolha dos equipamentos deve-se à importância que têm para a navegação, comunicação e baixo consumo de energia dos mesmos.

Módulo de Navegação

Composto pelos sensores de GPS, IMU, RADAR e Sonda.

Os dois últimos de acordo com a arquitetura estarão ligados apenas na linha principal de corrente. Caso falhe a linha principal de corrente o veículo deixará de navegar com os equipamentos, originando uma limitação para a navegação. Ligados ao computador de bordo pela interface RS232. O veículo deve possuir um HUB com portas RS232 para conexão de outros equipamentos

Módulo Visual

Constituído por dois sistemas. A câmara eletro-ótico (EO) modelo Flir Duo R, conectado a linha secundária de energia elétrica. O outro sistema é o LIDAR (Light Detection and Ranging) modelo livox mid-40. Encontra-se ligado ao computador de bordo por USB e como qualquer outro subsistema do veículo ligado a linha principal de energia

Limitações

Nesta fase do desenvolvimento, o veículo, apresenta diversas limitações como a indisponibilidade ou aquisição dos equipamentos. Até a elaboração desta dissertação existia o módulo de telecomunicações e de navegação básica desenvolvido pelo ASPOF Vieira Rodrigues e é com base nesses dois módulos que se desenvolveu arquitetura de software e posteriormente a construção do algoritmo para esta dissertação.

- Equipamentos indisponíveis

    1. Casco (Em construção no Arsenal do Alfeite)

2. Módulo de Propulsão e Energia (motores e baterias)

- Equipamentos estudados mas não adquiridos
  1. Módulo Visual (Lidar, Câmara EO/IR)
  2. Módulo de Navegação incompleto (Radar, Sonar e Sonda)
- Equipamentos disponíveis
  1. Módulo de telecomunicação (LoRa, Wi-Fi)
  2. Módulo de navegação básica (GPS, IMU)

3.2.2 Estrutura Eletrónica Existente

Na figura 3.4 ilustrada os dispositivos atuais que compõem o VentSupEN, constituídos por Computador de bordo (Raspberry Pi, RPY), (microcontrolador ESP32) por sua vez conectada ao RPY via RS232 e o Router cujo o único objetivo de estabelecer comunicação Wi-Fi com a estação C2 ou com eventuais veículos. É de referir que já foi apresentada uma arquitetura semelhante pelo ASPOF Vieira Rodrigues na sua dissertação.

Computador de Bordo

O sistema computacional a bordo é baseado no Raspberry Pi 3 Modelo B+ (Figura 3.2), alimentado por 5V/2.5A DC via micro USB, um processador quad core de 64 bits que funciona a 1,4 GHz com 1GB LPDDR2 SDRAM de memória.

Tem interface para USB 2.0 (débito máximo 300 Mbps) 4 portas USB 2.0 e 2.4 GHz e 5 GHz IEEE 802.11.b/g/n/ac LAN sem fios.

Suporta formato Micro SD card para armazenar o sistema operativo como também os dados recolhidos durante as operações.

A placa conta com cabeçalho extendido com 40 pin GPIO (General Purpose Input/Output) para adição de outras placas ou outras interfaces de comunicação (pi Foundation, 2020), por exemplo o módulo RS232 Board - WS (Rodrigues, 2019).

Microcontrolador ESP32

O Microcontrolador, "modelo Wemos TTGO t-beam é composto por um processador dual-core de 32 bits (Xtensa) com uma velocidade de CPU de 240 MHz.

Possui uma memória interna SRAM de 520kB e uma memória flash externa de 4MB. Com portas multiplexadas, é capaz de permitir: 18 canais ADC (Analog to Digital Converters), 3 interfaces SPI (Serial Peripheral Interface), 3 interfaces UART (Universal Asynchronous Receiver Transmitter), 2 interfaces I2C (Inter Integrated Circuit), 2 interfaces I2S (Integrated Interchip Sound), 16 saídas PWM (Pulse Width Modulation) e 2 canais DAC (Digital to Analog Converters)" (Rodrigues, 2019).

O mesmo funciona como placa de aquisição de dados para os sensores de GPS u-blox M8M acoplado a placa, IMU Bosch BNO055 conectado a placa nos pinos GPIO 21 e 22. Também acoplado encontramos o módulo transrecetor (Semtech SX1276) para comunicação LoRa (Long Range).

Os sistemas de comunicação já foram desenvolvidos pelo ASPOF Vieira Rodrigues (433MHz e 2.4GHz), bem como as antenas fabricadas e a interface LoRa implementada no microcontrolador ESP32.

FIGURA 3.2: Raspberry Pi

FIGURA 3.3: Microcontrolador ESP32

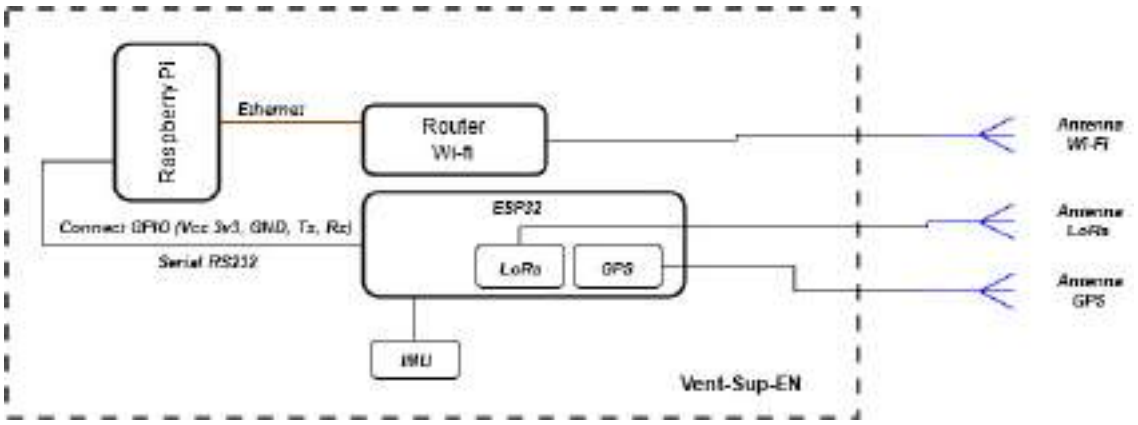


FIGURA 3.4: Arquitetura Eletrónica do VentSupEN

3.3 Arquitetura de Software

A gestão da informação é um fator chave para o planeamento de um sistema autónomo. Tal como numa organização a necessidade humana de comunicar é indispensável para as tomadas de decisão. Os subsistemas dos veículos também requerem a mesma necessidade, para melhor tomadas de decisão, devido seu funcionamento interno com ambiente externo. O sistema operativo instalado no Raspberry Pi (RPY) é o Ubuntu Mate, encontra-se instalado o ROS, que vai ser desenvolvida a interface com sensores, gestão de telecomunicações do veículo, bem como implementação de decisão e controlo do sistema. O C++ foi a linguagem de programação escolhida para o desenvolvimento.

Módulo é um conjunto de nós que contribuem para uma mesma funcionalidade. O conceito de nó apresentado está relacionado com o "node" na arquitetura "ROS".

Na arquitetura de software proposta bem como a desenvolvida, os nós ou módulos foram divididos em camadas de acordo com o processamento de informação no sistema. Na referida arquitetura as camadas podem aumentar ou os nós trocarem de camadas, visto que, não se pode considerar definida a arquitetura devido a limitação do hardware o que implica o desenvolvimento de drivers para determinados sensores. As camadas estão organizadas da seguinte forma:

- Camada 0 (Aquisição de dados)
- Camada 1 (Processamento de dados)
- Camada 2 (Controlo e decisão)

- Camada 3 (Supervisor)

3.3.1 Arquitetura de Software proposta

A arquitetura apresenta uma abordagem geral com a implementação dos diversos sensores/equipamentos que o veículo possuirá. Sendo que na camada 0 são ilustrados os drivers respetivos para os equipamentos e as interfaces de comunicação com o hardware. A ideia principal na camada 1 é que os nós compilem as informações provenientes dos drivers. Os mesmos são denominados com nomes dos módulos para facilitar a integração de novos drivers no sistema, todos os nós que darão apoio e decisão ficam localizados na camada 2 e por último a camada que terá os nós responsáveis por supervisionar o sistema e comunicar com o exterior (camada 3). Ver apêndice B

3.3.2 Arquitetura de Software desenvolvida

Os nós (motores e controlador) não serão implementados devido à inexistência do equipamento. Contudo, apresenta-se a ideia principal da arquitetura para ser implementada em trabalhos futuros.

Módulo de Hardware

Designou-se módulo de hardware, pelo fato dos nós descritos estarem directamente relacionados com a comunicação do Hardware.

O "`status_hardware_node`", supervisiona e informa ao supervisor todos os estados de hardware existente a bordo do veículo.

O nó "`read_rs232`", faz a leitura das portas RS232 e USB's conectadas.

Módulo de Navegação básica

O "`Navegation_node`"reúne as informações de "`gps_node`"e "`imu_node`"compila e disponibiliza informações pertinentes para o planeamento e o supervisor. O mesmo tem a função de reduzir ao máximo os erros obtidos por GPS e recalcular a posição com sistemas inerciais na ausência do GPS.

"`Planning_node`", este tem a função de gerir todo o planeamento inserido no veículo

Módulo de Comunicação

O "`lora_node`"e o "`wi-fi_node`", são os canais de comunicação que o veículo vai usar para comunicar com a estação C2 e eventualmente com outros veículos.

O nó (node) "`gest_comunicações`"assegura a comunicação e partilha os tópicos ROS entre máquinas sejam outros veículos ou estação C2. O nó usa o canal de comunicação mais adequado para o envio de dados. Quando houver comunicação Wi-Fi usa o multimaster (múltiplos mestres) devido a largura de banda, mas quando essa ligação é perdida usa a comunicação LoRa.

Módulo Supervisor

O "`supervisor_node`"supervisiona continuamente o comportamento do veículo e gera alertas para outros nós se as margens de segurança forem excedidas ou se ocorrerem acontecimentos inesperados. Neste módulo são definidas as mensagens que serão enviadas para outros veículos ou estação C2.

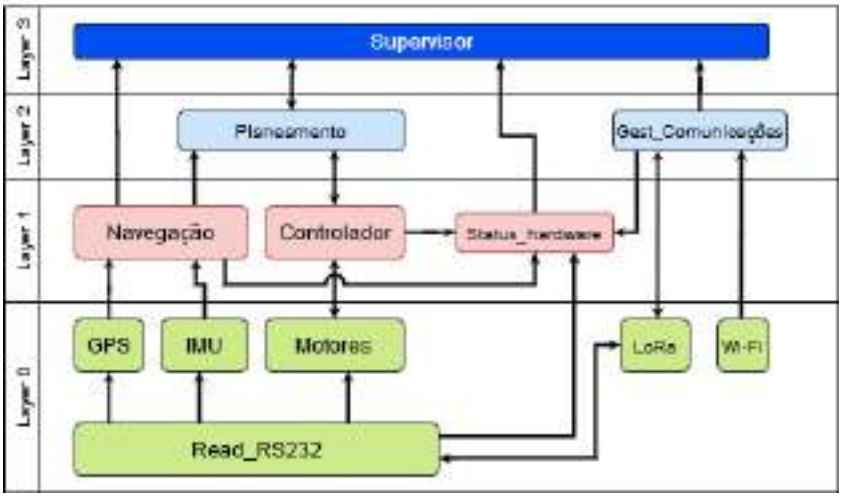


FIGURA 3.5: Arquitetura do Sistema Implementado

3.3.3 Validação

Arquitetura de Software apresentada no presente capítulo tem como base as teorias de (Birnberg, 2011) e de (Ferreira & Otley, 2009) que definem o seguinte: "*Na gestão de uma unidade, que tem por base a obtenção e utilização de recursos de forma eficiente, para se atingir os objetivos organizacionais, é necessária informação a três níveis* ".

- Nível Estratégico (nível superior) - São tomadas decisões estratégicas; são complexas e exigem informação bastante variada e ao nível das relações do meio envolvente. A informação provém de fontes externas à organização e também dos outros níveis hierárquicos.
- Nível Tático (nível intermédio) – Têm lugar as decisões táticas e que exigem informação pormenorizada, com algum tratamento, havendo responsabilidades na interpretação da informação, que provém de fontes internas e sendo obtida com alguma frequência.
- Nível Operacional (nível de base) – Aqui são tomadas as pequenas decisões ou as decisões operacionais. Decisões para problemas bem definidos cuja resolução é, muitas vezes, baseada em dados factuais programáveis e através da aplicação de rotinas informáticas. São necessárias informações pormenorizadas e bem definidas, provenientes essencialmente do sistema interno, com vista a ações imediatas

FIGURA 3.6: Arquitetura do Sistema Implementado

As camadas mais baixas (layer 0 e 1) estão relacionados com entrada e distribuição de informação correta e de acordo com a pirâmide tratar-se de informação a nível operacional. A nível tático ou intermédio estão os nós responsáveis por processar a informação ao pormenor e tomar decisões de controlo do veículo, como por exemplo: planeamento (layer 2) envia proas para o controlador (layer 1). A última camada, supervisiona todo os sistemas e comunica com o ambiente externo em auxílio dos nós de comunicação (Nível Estratégico).

Capítulo 4

Implementação

Nesta dissertação não serão implementados todos os nós descritos no capítulo 3, por falta de hardware e pelo veículo estar em construção. Os nós desenvolvidos são:

- `read_rs232`
- `status_hardware`
- `gps_node`
- `imu_node`
- `navigation_node`
- `planning_node`
- `lora_node`
- `wifi_node`
- `gest_comunic_node`
- `supervisor_node`

Alguns nós, vão além do tema da dissertação (por exemplo: `planning_node` e `navigation_node`). Contudo foram implementados para deixar um pequeno esboço de um nó funcional que deverá, em trabalhos futuros, ser desenvolvido com mais detalhes.

Sublinha-se que o algoritmo desenvolvido na realização da dissertação tem como principal objetivo de implementar drivers ROS para comunicação dos subsistemas atualmente existentes. Com esta implementação, não só pretende-se partilhar a maior quantidade informação dos sensores (GPS, IMU) entre os nós desenvolvidos,

bem como a utilização da informação já formatada e o respetivo envio pelos canais de comunicação (LoRa e Wi-Fi) para a estação.

Roscore: é um conjunto de nós e programas que são pré-requisitos de um sistema baseado em ROS. O mesmo permite que os nós ROS se comuniquem quando iniciados, significa que antes de executar os nós dos sistemas o roscore é o primeiro a ser iniciado usando o comando roscore (ROS, 2020b).

Roslaunch é um ficheiro onde se pode definir os nós que vão correr quando é executado o ficheiro. O roslaunch inicia automaticamente o roscore se detetar que ainda não está em execução. E para iniciar o roslaunch usa-se o comando "roslaunch [package] [filename.launch]" e para este projeto os nós são iniciados da seguinte forma "roslaunch navigation esp32lora.launch". Caso não ocorram erros, então, o sistema está pronto para usar os comandos "rosnode list", "rostopic list" entre outros para verificar como a informação é publicada. O roslaunch permite definir quais os nós a ser iniciado na máquina remota, usando a tag <machine>. Esta configuração no ficheiro torna-se bastante útil para iniciar os sistemas em vários veículos a partir da estação C2.

4.1 Firmware ESP32

O microcontrolador usado pelo ASPOF Vieira Rodrigues, executava um firmware que permitia a aquisição e processamento dos dados GPS e IMU bem como a receção e envio de informação pelos canais de comunicação LoRa e Wi-Fi. Contudo, teve-se a necessidade de alterar o firmeware do microcontrolador com o intuito de torna-lo compatível com a implementação feita em ROS.

O firmware tem as seguintes prioridades:

1. Leitura do `LoRa.parsePacket`;
2. Leitura do `Serial2`;
3. Escrever o `readData`

O LoRa.parsePacket foi definido como primeiro por forma a priorizar as comunicações LoRa. A cada ciclo do firmware verifica-se primeiro se há receção de mensagens, caso se verifique receção de mensagem o algoritmo passa a compor a mensagem em formato LoRa definido e encaminha para `Message_Struct`. No caso de não verificar receção de mensagem LoRa é executada a segunda prioridade.

O algoritmo verifica se existe um pedido do RPY que pode ser envio de mensagens via LoRa para estação de C2 bem como outros veículos ou reiniciar o próprio controlador.

Após verificação entrada e saída de mensagens ou pedidos, é executado o readData como última prioridade do firmware. O readData reúne todos os dados de GPS, IMU, e condições do Hardware e encaminha para o `Message_Struct`.

De forma a proporcionar alguma robustez e clareza à comunicação entre o microcontrolador e o computador de bordo, o ASPOF Viera Rodrigues criou o código que efetua troca de informações com recurso a mensagens pré-formatadas, onde são introduzidos o cabeçalho, dados a enviar e um rodapé de forma encapsulada, ou seja, é no `Message_Struct` que está definido o protocolo de envio de mensagem pela porta RS232.

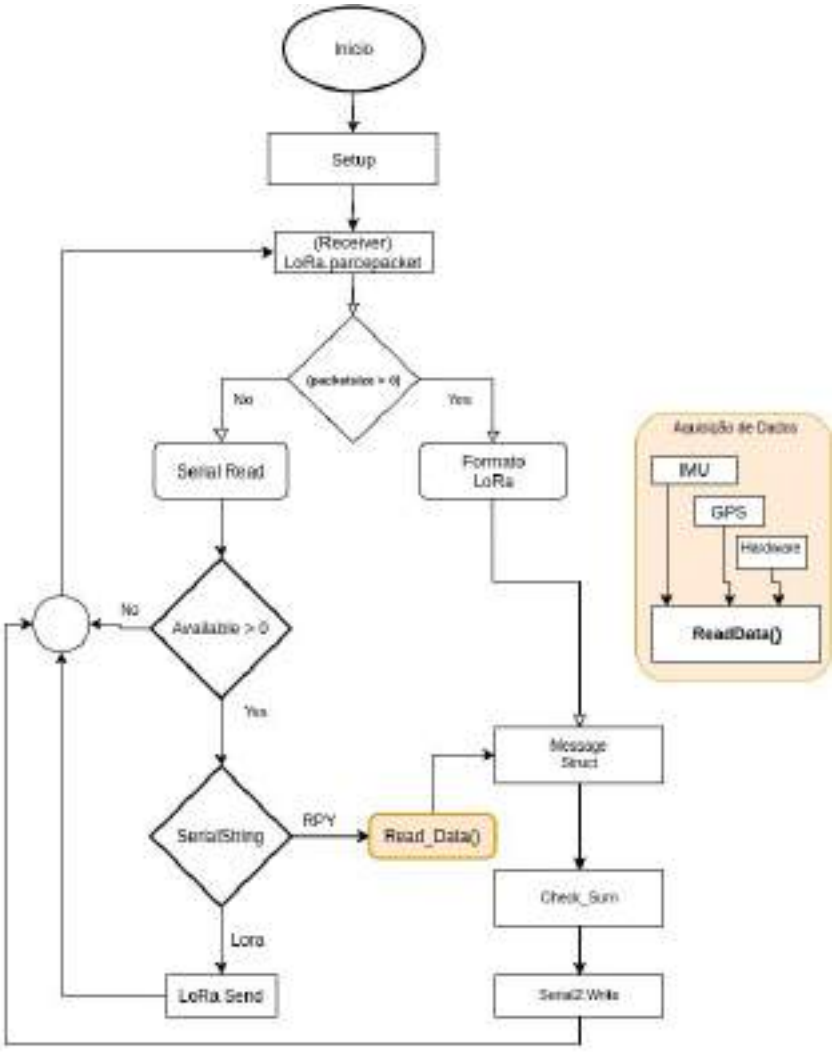


FIGURA 4.1: Fluxograma para aquisição de dados no ESP32

O algoritmo implementado está disponível na plataforma do Github em https://github.com/apo360/Ros_VentSup/

4.2 Modulo de Hardware

No presente módulo, foram criados dois nós: O nó `read_rs232` e o nó `status_device_node`.

4.2.1 read-rs232

O nó `read_rs232`, é responsável pela ligação de todas as portas séries (RS232 e USB) no computador de bordo. Este nó foi desenvolvido com capacidade de gestão de multiportas, deixando de lado a necessidade de definir regras ou sequências de ligação de tais portas. Para desenvolver o nó utilizou-se um código aberto existente na plataforma do GitHub desenvolvido por (Woodall, 2014).

O script permite ler as portas séries do computador de bordo sem usar o ROS. No entanto, para integrar com o ROS, foram criados dois ficheiros, o primeiro (portas.h) é um ficheiro pelo qual são registados números de séries de cada dispositivo e tópicos dos quais deve ser publicada a respetiva informação. O segundo, verifica quais os dispositivos conetados ao computador de bordo (online). Lê os dados e publica-os no tópico definido. O ESP32 (TTGO - Tbean), publica os dados em várias Strings no tópico `/rs232_out`, denominado como tópico principal do sistema.

O nó não foi testado para múltiplas portas RS232 e nem mais de 3 portas USB em funcionamento simultâneo.

4.2.2 status-device-node

O `status_device_node`, tem como objetivo principal recolher a informação do estado de todo hardware existente a bordo afim de compilar toda a informação e envia-la para o supervisor. O nó deve subscrever-se nos nós que compilam as informações dos drivers, os mesmos deverão ter tópicos dedicados a publicar informações relativamente às condições dos seus hardwares.

O tópico `/rs232_out` além de publicar informações sobre o GPS, IMU e LoRa também publica informações dos hardwares integrados ou conectados no microcontrolador ESP32. A string iniciada "$Hardware"é a que tem relevância para a leitura do algoritmo, uma vez que, contêm dados do estado do IMU (nome e temperatura), largura de banda, temperatura média da placa e carga da placa (exemplo:

"$Hardware:BNO055 30 Conectado, LoRa-433000000.00 Conectado, Battery Voltage :3.69") após a receção dos dados o nó separa e trata, de seguida publica os dados pelo tópico /`harware_status_info`.

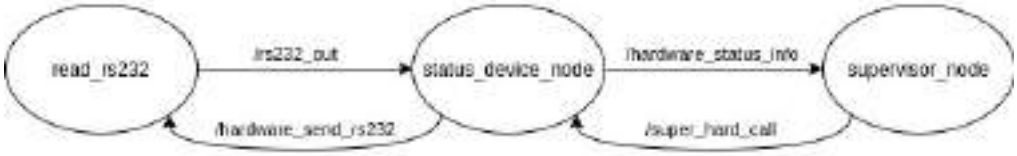


FIGURA 4.2: Diagrama de ligação dos nós de hardware

4.3 Módulo de Navegação

A aquisição e processamento de dados é importante para localização e controlo do veículo (VentSupEN).

O módulo é constituído por quatro nós: `gps_node`, recebe os dados do GPS; `imu_node`, recebe os dados pelo IMU; `navigation_node`, executa diversos algoritmos para estimar a posição com o menor erro possível; `planning_node`, responsável por organizar, dirigir e controlar o planeamento de navegação do VentSupEN.

4.3.1 gps-node

O nó subscreve-se no tópico `rs232_out` para leitura dos dados. A string que começa por $GPGGA é a que tem significância para o algoritmo, a mesma contém os valores atuais para a latitude e longitude, UTC (Universal Time Coordinated), data atual, número de satélites, velocidades e rumo relativamente ao fundo do mar: SOG (Speed Over Ground); COG (Course Over Ground).

Após captar a string, os dados são separados e publicados nos diferentes tópicos do nó. Exemplo da string: "$GPGGA:114904Z,38.66258 N,9.14814 W,0.05,18.00,7/2/2020,12,". O nó publica os seguintes tópicos:

- `/gps_position` (latitude e longitude)
- `/gps_sog_cog` (velocidade e rumos verdqdeiros em relação o fundo do mar)
- `/gps_satelites` (número de satélites)
- `/gps_utc` (data)

4.3.2 imu-node

Também subscrito ao tópico inicial do sistema, este nó lê as strings que começam por `$HCHDT`. Na string, encontramos dados agrupados como: bússola, aceleração, giroscópio, magnético e gravítico com os seus eixos de orientação (X Y e Z).

Exemplo: HCHDT:orientX orientY orientZ, gravityX gravityY gravityZ, gyroX gyroY gyroZ, accelX accelY accelZ, magnetX magnetY magnetZ, . Como `gps_node`, `imu_node` também separa os dados em tópicos diferentes de forma agrupada:

- `/imu_orientation`
- `/imu_gravity`
- `/imu_gyroscope`
- `/imu_linearaccel`
- `/imu_magnetometer`

4.3.3 navigation-node

Como especificado a cima, o nó terá objetivo de calcular a posição do VentSupEN com o menor erro possível aplicando formulas para a diminuição dos ruídos, como por exemplo: aplicação de filtros de Kalmam para navegação usando sistemas inerciais (Inertial System Navigation, INS). Subscreve-se nos tópicos: `/gps_position`, `/gps_satelites`, `/imu_orientation`, `/imu_magnetometer` e usa os dados recebidos para calcular ou executar os diversos algoritmos que forem implementados.

No futuro, o nó poderá correr algoritmos que ajudam a realizar dois dos três métodos de navegação autónoma (de Guerra Portuguesa, 2012):

- Geonavegação, aquela que usa sistemas visuais, RADAR e GPS.
- Navegação Estimada, aquela que usa os sistemas inerciais para estimar a posição

O tópico /`gps_satelites` publica informações relativamente número de satélites que o veículo está conetado e este valor é usado como parâmetro de verificação de conexão com GPS, por forma a executar os diferentes tipos de navegação. Sempre que o algoritmo verificar que o número de satélites é menor que quatro, o algoritmo passa a executar a função para navegação Inercial/métodos de INS para calcular a posição e assim continua a navegar além de publicar continuamente sua condição de navegação pelos tópicos /`info_super_position` e /`position_vehicle`.

Tanto os nós `gps_node`, `imu_node` e `navigation_node` não precisam de nenhum tipo de configuração inicial

4.3.4 planning-node

Este é um dos nós mais importante para navegação autónoma do veículo, com o objetivo de gerir completamente o planeamento de todas as missões que lhe forem introduzidas. Inicialmente, pode ser introduzido 200 way point para cada missão que o veículo executar. Os pontos podem ser introduzidos de várias formas: Um, nas configurações inicias do veículo ao carregar um ficheiro.txt na pasta do planeamento localizada em documentos.

Outra forma é o momento em que o veículo está em missão, a estação tem a possibilidade de carregar os pontos necessários no planeamento via tópico. O tópico responsável para introdução dos pontos é o /`super_plano`, publicado pelo supervisor, o mesmo encontra-se disponível para comunicação com outras máquinas nos dois canais de comunicação.

Quando o planeamento é carregado, o nó faz vários cálculos que foram definidos durante o seu desenvolvimento. Calcula a distância, rumo, tempo, abatimento (indica se o veículo se encontra a EB ou BB do planeamento), tempo (tp) de adiantamento ou atraso do planeamento, distância total e tempo total da missão.

O nó usa a Formúla de Haversine[8] para calcular a distância e rumo entre dois pontos.

[8] A fórmula de Haversine é uma importante equação usada em navegação, fornecendo distâncias entre dois pontos de uma esfera a partir de suas latitudes e longitudes

$$x = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi 2 - \phi 1}{2}\right) + \cos\phi 1 * \cos\phi 2 * \sin\left(\frac{\lambda 2 - \lambda 1}{2}\right)}\right) \tag{4.1}$$

$$\beta = \arctan\left(\frac{\sin(\lambda 2 - \lambda 1) * \cos\phi 2}{(\cos\phi 1 * \sin\phi 2) - (\cos\phi 2 * \sin\phi 1 * \cos\Delta\lambda)}\right) \tag{4.2}$$

$$\lambda - Longitudes$$

$$\Delta\lambda - VariaçãodeLongitudes$$

$$\phi - Latitudes$$

$$r - RaiodaTerra$$

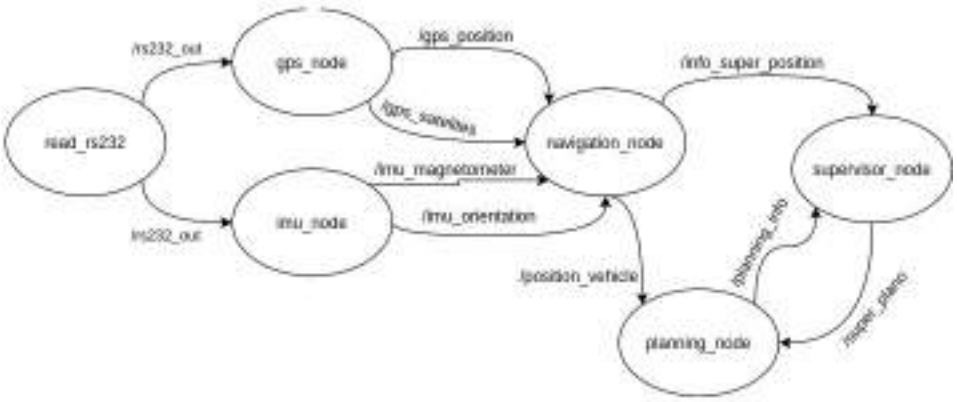


FIGURA 4.3: Diagrama de ligação dos nós (Módulo de Navegação)

4.4 Módulo de Comunicação

Tão importante quanto o módulo de navegação, este mostra a capacidade de comunicação e de controlo remoto que o veículo terá com a estação de comando e controlo, bem como a comunicação com outros veículos da rede.

Aqui serão descrito os dois métodos de comunicação que o veículo terá com a estação.

4.4.1 lora-node

O nó começa por receber todos os dados que estiverem identificados como lora pelo tópico /`rs232_out`, verificando se a mensagens recebida é para o veículo. Compara o IP do destinatário com o seu sistema, caso verifique igualdade reencaminha a mensagens respetiva para o gestor de comunicações usando o tópico /`lora_-out`. O formato da mensagem tem o seguinte formato: Lora:from,to,$msg. Como Exemplo: Lora:192.168.0.101,192.168.0.102,$/`super_plano`/4,(lat1;lng1;Vel1) ,...,(latn,lngn,Veln).

4.4.2 Multimaster

O sistema multimaster ROS serve para partilhar os tópicos entre diferentes máquinas, neste caso referimos em veículos ou estação C2. O multimaster consiste em duas ou mais redes ROS em que cada rede tem os seus próprios nós roscore. Dado que cada veículo utiliza a mesma arquitetura de software e tem os mesmos nós, surgem alguns problemas devido ao mesmo nome de nó em toda a rede. Este problema pode ser resolvido utilizando diferentes namespaces para cada veículo. Graças a esta estrutura, é possível controlar o movimento de vários veículos por Wi-Fi, utilizando apenas um computador de monitorização.

Para esse propósito, precisamos de um pacote chamado `multimaster_fkie` (Hernandez, Fernando & Cotarelo, 2015). Este pacote pode ser facilmente instalado como mostrado abaixo.

sudo apt-get install ros-melodic-multimaster-fkie

Este pacote oferece um conjunto de nós para gerir uma rede multimaster. Isto requer uma configuração mínima. As alterações são automaticamente identificadas e sincronizadas. O `multimaster_fkie` permite que dois nós essenciais: o `master_discovery` e `master_sync` funcionem ao mesmo tempo.

As principais características do `master_discovery` é enviar regularmente mensagens múltiplas à rede ROS para que os outros ambientes roscore tenham conhecimento da sua presença. Também verifica as alterações na rede local e informa todos os roscore sobre as mesmas. O `master_sync` permite selecionar quais os hosts, tópicos e serviços que devem ser sincronizados ou ignorados entre os diferentes roscore. Ajuda a registar e atualizar informações de tópicos e serviços para o roscore local (Hernandez, Fernando & Cotarelo, 2015).

Configurações

Antes de configurar o pacote na rede deve-se ter em consideração uma rede bem configurada(cada veículo tem o hostname que corresponde à um IP), por forma evitar eventuais conflitos na rede. A segurança na rede também é importante para transferência de informação, temos como exemplo de segurança o (Secure Socket Shell) SSH. Na presente dissertação recorreu-se a criptografia assimétrica, consiste em criar um par de chaves conhecidas como chave-pública e chave-privada. A chave-privada permanece no servidor (neste caso estação de comando e controlo) enquanto que a chave-pública é distribuída no veículo ou veículos da mesma rede. Após confirmados os IP e a segurança da rede sucede-se então para as configurações do pacote.

- `$` export `ROS_MASTER_URI` = http://<host_ip>:11311, é uma configuração necessária que informa aos nós onde eles podem localizar o master;
- Iniciar o roscore;
- `$` rosrun `master_discovery_fkie master_discovery _mcast_group`:=224.0.0.1, este deve ser iniciado a cada veículo;
- `$` rosrun `master_sync_fkie master_sync`

4.4.3 gest-comun-node

O responsável pela gestão de comunicação dos nós lora e wifi, este nó identifica quando uma informação ou mensagem deve ser enviada por lora e quando existe uma ligação wifi com a estação. Espera sempre a mensagem do nó supervisor pelo tópico `/super_Comuni_Out` para enviar nos nós respetivos. Este, inicia os parâmetros de verificação de comunicação LoRa quando o sistema é iniciado. É de referir que quando não há comunicação Wi-Fi , o nó encarrega-se de passar a informação dos tópicos críticos através de LoRa.

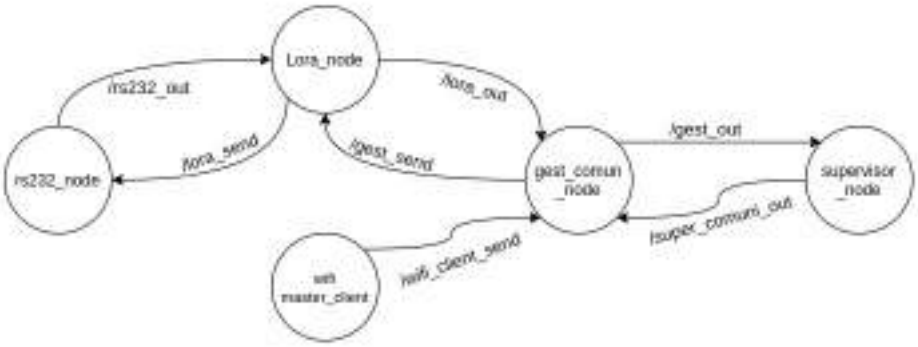


FIGURA 4.4: Diagrama de ligação dos nós de Comunicação

4.5 Módulo Supervisor

4.5.1 supervisor-node

Como já mencionado no capítulo anterior, o supervisor verifica o comportamento do veículo continuamente subscrevendo-se nos tópicos `/navigation_node`; /planning; `/gest_comun_node` e `status_device_node`;

O nó é o intermediário do sistema interno com o exterior, toda a informação seja ela de interno para exterior e vice-versa passa obrigatoriamente pelo supervisor. O mesmo publica os tópicos `/super_comuni_out`, `/super_comuni_master`, `/super_plano` e `/super_hard_call`.

Todo o código implementado pode ser encontrado no repositório do GitHub no seguinte endereço https://github.com/apo360/catkin_ventsup_ws

Capítulo 5

Testes e Análise de Resultados

Com o objetivo de avaliar o algoritmo desenvolvido para esta dissertação, foram realizados testes nos módulos de navegação e comunicação. O teste efetuado no módulo de navegação serve para averiguar a posição (GPS), orientação (IMU), planeamento e falhas de hardware no veículo. Para concluir, é feito um teste geral cujo objetivo é avaliar o sistema com um planeamento com mais pontos.

5.1 Módulo de Navegação

5.1.1 GPS e IMU

Para verificar se o veículo indicava a sua posição corretamente fez-se a comparação com recurso a ferramenta online Google Maps. Na ferramenta, foi marcada a posição onde o veículo se encontrava e de seguida analisou-se os dados GPS que o mesmo publicava, a informação apresentada a verde (figura 5.1) indica a posição do veículo na ferramenta. A cor azul representa a informação que o veículo publicava no momento em que se realizavam os testes. Nota-se claramente que os dados fornecidos pelos dois sistemas são iguais, o que indica a eficiente capacidade que o veículo tem em fornecer a sua localização. Com isso, percebe-se que o sistema implementado funciona de forma satisfatória.

No teste de IMU feito, o objetivo era de verificar se com a variação da proa (rodar a estrutura tanto a EB como a BB) o sistema conseguia publicar a informação da mesma naquele momento. A figura 5.2 mostra a variação dos dados consoante rotação da estrutura.

FIGURA 5.1: Teste de dados de informação GPS

FIGURA 5.2: Teste de dados de informação IMU

5.1.2 Planeamento

No teste em causa introduziu-se no sistema um plano que tinha dois way points para simular a eficácia do planeamento, que consistia em trocar de way point quando atinge a distância mínima do anterior para tal, foi necessário introduzir os dados GPS manualmente. Iniciou-se todo sistema (os nós) do veículo e a seguir usou-se o comando "rosnode kill `navigation_node`"para cancelar o nó, afim de impedir que o nó `planning_node` receba os dados do GPS automaticamente. Foram iniciadas duas janelas "terminal". No terminal 1 executou-se o comando "rostopic echo `/planning_info`" este tópico publica dados relativamente ao comportamento da navegação do veículo e no terminal 2 usou-se o tópico "`/position_vehicle`" para enviar dados do GPS manualmente. No nó `planning_node` definiu-se que quando a distância ao ponto mais próximo fosse menor que cinco metros o algoritmo mudava para o próximo a executar e assim sucessivamente.

O terminal 2 serviu para introduzir a informação GPS e o terminal 1 para visualizar o comportamento do veículo em deslocamento durante a simulação. No terminal 1 temos informações do way point em que o veículo está a dirigir-se (informação sublinhada a linha continuo verde), rumo, tempo, abatimento (Abat) e a distância ao way point (informação sublinhada a tracejado verde - Dist). O Dist calcula a distância em que o veículo está desde a sua posição até ao próximo way point. Cada vez que o nó recebe a informação GPS pelo tópico "`/position_vehicle`" a distância é recalculada. Como pode-se verificar nas caixas a verde da figura 5.3. A caixa a verde do terminal 2 mostra informações GPS inseridas com posições diferentes e na caixa verde do terminal 1 mostra os cálculos feitos de acordo com os dados recebidos do tópico.

Pode-se verificar que, para um nó parcialmente funcional já dispõem da capacidade de mudar de um way point para o outro sempre que atinge a distância mínima do próximo way point. Conforme pode-se ver nos retângulos a vermelho (terminal 1).

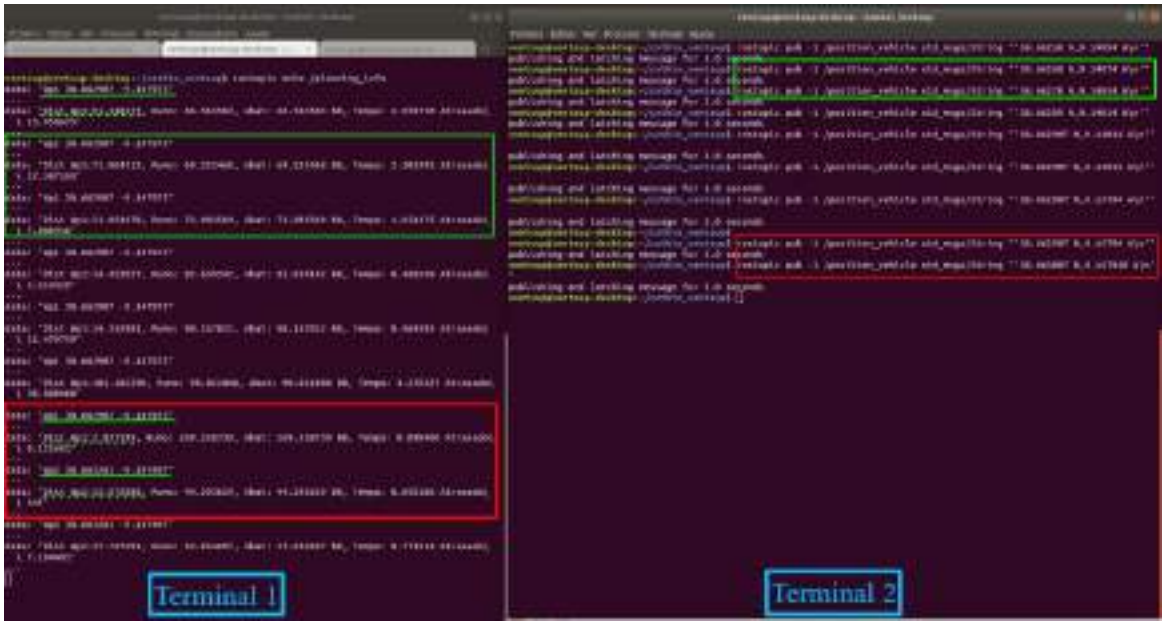


FIGURA 5.3: Teste do Planeamento

5.2 Falha no Hardware

Relativamente a falha de hardware não foi possível testar cada componente pelas seguintes razões:

- i. O GPS está integrado no microcontrolador e o IMU conectado ao mesmo;
- ii. O GPS, IMU e o microcontrolador têm a mesma interface de comunicação com RPY;
- iii. Não foram realizados estudos para o acesso de cada componente a partir do microcontrolador.

Por outro lado, realizou-se teste no microcontrolador que consistiu em desconectar a porta RS232 para a verificação do comportamento do sistema. A referida ação, origina de imediato um erro nó `read_-rs232` e em consequência o nó é automaticamente cancelado e a estação C2 deixa de receber as informações do microcontrolador.

Mediante o exposto, não é expectável que tal erro ocorra quando o veículo se encontra a navegar, visto que seria necessário a ação humana para desencadear o erro. Pelo facto do GPS possuir uma antena, foi possível realizar o teste de GPS ao desconectar a antena que ao mesmo tempo simula o GPS conectado com menos de quatro satélites. Os resultados são apresentados na secção 5.3 Integração Final.

5.3 Integração Final

A parada da Escola Naval (Contra-Almirante Rodrigues Sarmento) foi o local escolhido para o teste de integração final. Nesse teste foi criada uma estação C2 e a estrutura que simulou o VentSupEN em movimento e a estação C2 estava localizada no DCT (Departamento de Ciências e Tecnologia) sala do CINAV. O plano de teste teve como objetivo principal percorrer a parada em cinco way points inicialmente definidos. Ver Figura 5.4. O teste consistia em gerar várias simulações como: a perda do sinal Wi-Fi, GPS e a verificação do percurso do veículo. Todos os testes ocorreram em um único percurso cerca de 370,7 metros. O teste de comunicação teve como foco em analisar capacidade do sistema continuar a transmitir informações essenciais por LoRa quando o mesmo perde comunicação Wi-Fi.

Inicialmente, o veículo, estava conectado com a estação por Wi-Fi e LoRa. Na medida que se afastava em direção ao way point 1 (105 m da estação) o sinal ficou cada vez mais fraco até que a estação perdeu ligação Wi-Fi e os tópicos disponibilizados pelo Multimaster. Ainda assim, a estação continuou a receber as informações relevantes por LoRa.

No way point quatro a antena do GPS foi desligada, logo, o sistema identificou que o número de satélites era menor que quatro e passou imediatamente para o método de navegação inercial onde emitiu avisos contínuos para a estação C2 com vista a comunicar a sua condição. Ver Figura 5.5.

Todo o percurso teve como o objetivo executar o planeamento que foi introduzido no sistema. Com isso, pode-se considerar que o sistema executou o algoritmo implementado com satisfação. O algoritmo trocava os way points sempre que estivesse a menos de cinco metros de cada way point de chegada, dando uma proa nova, a distância pela qual se encontrava, bem como, o tempo de chegada ao próximo way point. Ver Figura 5.5.

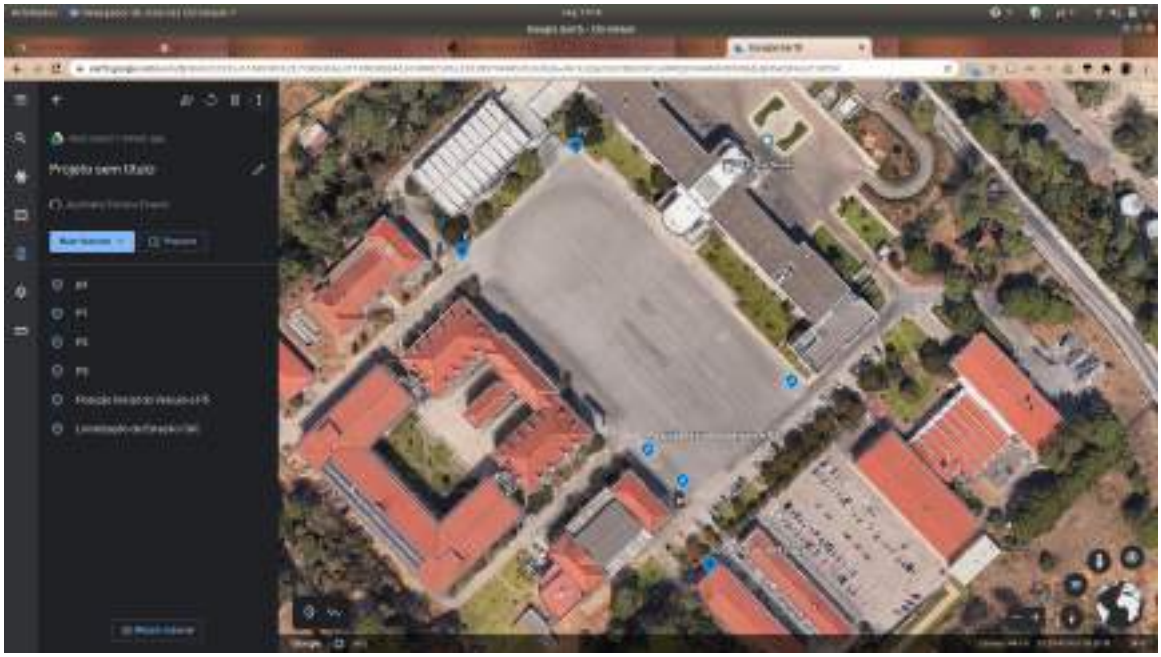


FIGURA 5.4: Local de Testes

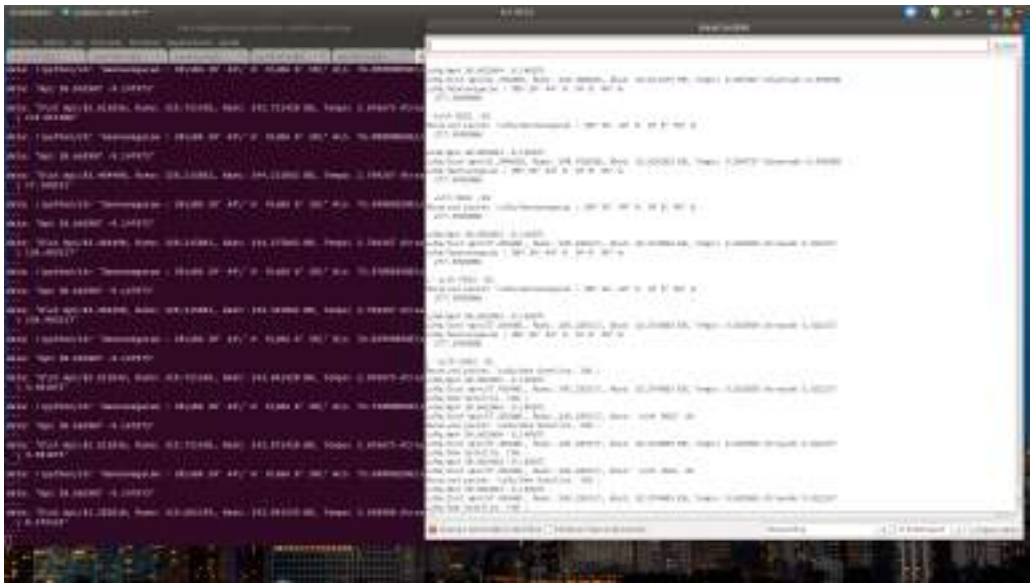


FIGURA 5.5: Simular perda de dados GPS

FIGURA 5.6: Parada da Escola Naval

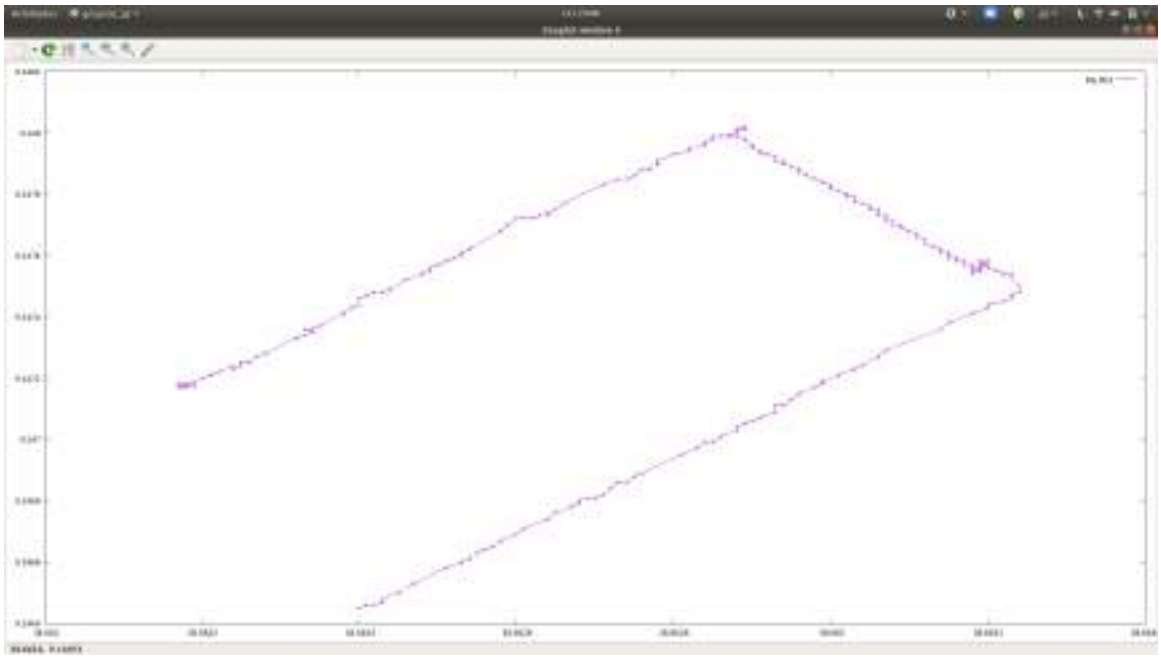


FIGURA 5.7: Percurso do Veículo

Capítulo 6

Conclusão e Trabalhos Futuros

Depois de um longo processo de investigação são extraídas as seguintes conclusões gerais:

1. O desenho deste projeto foi muito complexo por envolver o uso da linguagem ROS que exige um conhecimento prévio na área de programação e desenvolvimento de sistemas. Com isso, desencadear-se um processo que requer investimento, tempo e empenhamento dos seus programadores;

2. Apesar da complexidade, foi possível desenvolver os módulos mencionados no capítulo quatro dos quais foi possível alcançar os objetivos definidos para esta dissertação;

3. Para ir de encontro com os pressupostos definidos no início deste projeto começou-se com a elaboração de um planeamento (estudo e desenho do componentes do sistema) até o desenvolvimento do sistema robótico, do qual faz a gestão dos sub-sistemas do veículo em alusão, o que permitiu observar os impactos em diferentes fases e a relevância na implementação da linguagem dinâmica para gerir problemas complexos.

4. A oportunidade de estar encarregue de desenvolver o sistema de bordo do projeto VentSupEN, permitiu tomar decisões que facilitam a implementação do projeto (por exemplo, a alteração do firmware do micro-controlador ESP32 por forma facilitar a integração com o ROS).

5. Com este projeto pretende-se culminar com um instrumento de apoio no registo de aspetos importantes do VentSupEN, podendo ser também utilizado como um manual de consulta para futuros trabalhos.

6. No futuro, considera-se importante desenvolver trabalho no sentido referente à Sistema de Navegação Inercial com recursos aplicação de filtros de Kalmam, de forma a reduzir erros na navegação do veículo.

7. Ainda para trabalhos futuros: a encriptação de informação; desenvolvimento de algoritmos para encriptar e desencriptar dados para comunicação entre veículos ou estação de comando e controle.

Bibliografia

Alves, J., Oliveira, P., Oliveira, R., Pascoal, A., Rufino, M., Sebastiao, L. & Silvestre, C. (2006). Vehicle and Mission Control of the DELFIM Autonomous Surface Craft, IEEE. https://doi.org/10.1109/MED.2006.328689

Belbachir, A. N. (2010). Imaging Technologies and Architecture of Smart Cameras, *1*, 245–257.

Birnberg, J. G. (2011). Robert N. Anthony: A Pioneering Thinker in Management Accounting. *Accounting Horizons*, *25*(3), https://meridian.allenpress.com/accounting-horizons/article-pdf/25/3/593/1590381/acch-50025.pdf, 593–602. https://doi.org/10.2308/acch-50025

Carapau, R. S., Rodrigues, A. V., Marques, M. M., Lobo, V. & Coito, F. (2017). Interoperability of unmanned systems in military maritime operations: Developing a controller for unmanned aerial systems operating in maritime environments, IEEE. https://doi.org/10.1109/OCEANSE.2017.8084862

Cruz, N. A. & Matos, A. C. (2008). The MARES AUV, a Modular Autonomous Robot for Environment Sampling, IEEE. https://doi.org/10.1109/OCEANS.2008.5152096

da Costa, N. P. R. P. (2019). *Projeto VENT-Sup EN*. Escola Naval Portugal.

David Shaw, G., Mary. (1996). *Software architecture : perspectives on an emerging discipline* (N. : P. H. ©1996. Upper Saddle River, Ed.).

de Abreu Faria, L., de Melo Silvestre, C. A. & Correia, M. A. F. (2016). GPS-Dependent Systems: Vulnerabilities to Electromagnetic Attacks. *Journal of Aerospace Technology and Management*, *8*, 423–430. https://doi.org/10.5028/jatm.v8i4.632

de Guerra Portuguesa, M. (2012). *MINISTÉRIO DA DEFESA NACIONAL MARINHA ESTADO-MAIOR DA ARMADA DISPOSIÇÕES GERAIS E CONCEITOS FUNDAMENTAIS DA NAVEGAÇÃO INA 2*.

El-Rabbany, A. (2002). *Introduction to GPS: The Global Positioning System*. Artech House. https://books.google.pt/books?id=U2JmghrrB8cC

Ernst, N. A., Kazman, R. & Bianco, P. (2018). Towards rapid composition with confidence in robotics software. *Proceedings - International Conference on Software Engineering*, 44–47. https://doi.org/10.1145/3196558.3196567

Ferreira, A. & Otley, D. (2009). The design and use of performance management systems: An extended framework for analysis. *Management Accounting Research*, *20*(4), 263–282. https://doi.org/https://doi.org/10.1016/j.mar.2009.07.003

Hernandez, S., Fernando, J. & Cotarelo, H. (2015). Multi-master ROS systems. http://digital.csic.es/bitstream/10261/133333/1/ROS-systems.pdf

Joseph, L. (2018). *Robot Operating System for Absolute Beginners*. Apress. https://doi.org/10.1007/978-1-4842-3405-1

Kim, T., Lee, Y., Park, J. & Choi, H.-T. (2017). A study on the effectiveness of MOOS-IvP for Unmanned Surface Vehicle, IEEE. https://doi.org/10.1109/UT.2017.7890289

L3Harris. (2019). *L3Harris Technologies Delivers New Advanced Autonomous Vehicle Capability to UK's Defence Science and Technology Laboratory*. https://www.asvglobal.com/l3harris-technologies-delivers-new-advanced-autonomous-vehicle-capability-to-uks-defence-science-and-technology-laboratory/

LSTS, P. (2013). *DUNE source code repository. [Online]*. https://github.com/LSTS/dune

Mahmoud, A. & Atia, M. M. (2019). Hybrid IMU-aided approach for optimized visual odometry. *GlobalSIP 2019 - 7th IEEE Global Conference on Signal and Information Processing, Proceedings*, 0-4. https://doi.org/10.1109/GlobalSIP45357.2019.8969460

Manley, J., Marsh, A., Cornforth, W. & Wiseman, C. (2000). Evolution of the autonomous surface craft AutoCat, IEEE. https://doi.org/10.1109/OCEANS.2000.881292

Marques, M. M. (2018). *Reference Model for Interoperability of Autonomous Systems*. FCT.

Matos, A., Silva, E., Cruz, N., Alves, J. C., Almeida, D., Pinto, M., Martins, A., Almeida, J. & Machado, D. (2013). Development of an Unmanned Capsule for large-scale maritime search and rescue. *OCEANS 2013 MTS/IEEE - San Diego: An Ocean in Common*.

Melvin, W. L. & Scheer, J. A. (2014). *Princlpes of Modern Radar* (D. R. Kay, Ed.; Vol. III:). SciTech Publish.

Microsoft, A. (2020). *Middleware Definition*. https://azure.microsoft.com/pt-pt/overview/what-is-middleware/

Miltiadou, M., Warren, M. A., Grant, M. & Brown, M. (2015). ALIGNMENT OF HYPERSPECTRAL IMAGERY AND FULL-WAVEFORM LIDAR DATA

FOR VISUALISATION AND CLASSIFICATION PURPOSES. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-7/W3*, 1257–1264. https://doi.org/10.5194/isprsarchives-XL-7-W3-1257-2015

of Electrical, I. & Engineers, E. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*, 1. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342

of Electrical, T. I. & (610), E. E. .-. I. (1990). Standard Computer Dictionary.

pi Foundation, R. (2020). Raspberry Pi 3 model B+ specifikationer, 2. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

Pinto, J., Dias, P. S., Martins, R., Fortuna, J., Marques, E. & Sousa, J. (2013). The LSTS toolchain for networked vehicle systems, IEEE. https://doi.org/10.1109/OCEANS-Bergen.2013.6608148

Pod, H. (2019). *Hovercraft não tripulado*. https://hovpod.com/pt/aplica%C3%A7%C3%B5es-de-hovercraft/hovercraft-comercial/hovercraft-n%C3%A3o-tripulado-usv/

Possum, A. (2015). Publish Subscribe Architecture Group Name: Awesome Possum, *1*, 4. https://www.student.cs.uwaterloo.ca/~cs446/1171/Arch_Design_Activity/PublishSubscribe.pdf

Quigley, M., Gerkey, B. & Smart, W. D. (2015). *Summary for Policymakers* (I. P. on Climate Change, Ed.; Vol. 53). Cambridge University Press. https://doi.org/10.1017/CBO9781107415324.004

Recognition, N. (2019). *Israel Aerospace promote its Katana unmanned surface vessel*. https://www.navyrecognition.com/index.php/news/naval-exhibitions/2019-naval-exhibitions/imdex-2019-news-online-show-daily/7084-imdex-2019-israel-aerospace-industries-to-promote-its-katana-unmanned-surface-vessel.html

Rodrigues, T. V. (2019). *Instrumentação do Projeto VENT-SUP-EN*. Escola Naval Portugal.

ROS. (2020a). *ROS MSG*. http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

ROS. (2020b). *ROS ROSCORE*. http://wiki.ros.org/roscore

R.Sathishkumar, M. B., T.V.S Prasad Gupta. (2013). Echo Sounder for Seafloor Object Detection and Classification. *Journal of Engineering, Computers Applied Sciences*, *121990*, 32–37. https://pdfs.semanticscholar.org/d850/de02961034553815ec28adc937024c2ede43.pdf

Spears, A., West, M. & Collins, T. (2013). Autonomous control and simulation of the VideoRay Pro III vehicle using MOOS and IvP Helm. *OCEANS 2013*

MTS/IEEE - San Diego: An Ocean in Common, 1–10. https://doi.org/10.23919/OCEANS.2013.6741205

Target, T. (2020). *Information Technology Standards and Organizations Glossary*. https://whatis.techtarget.com/definition/framework

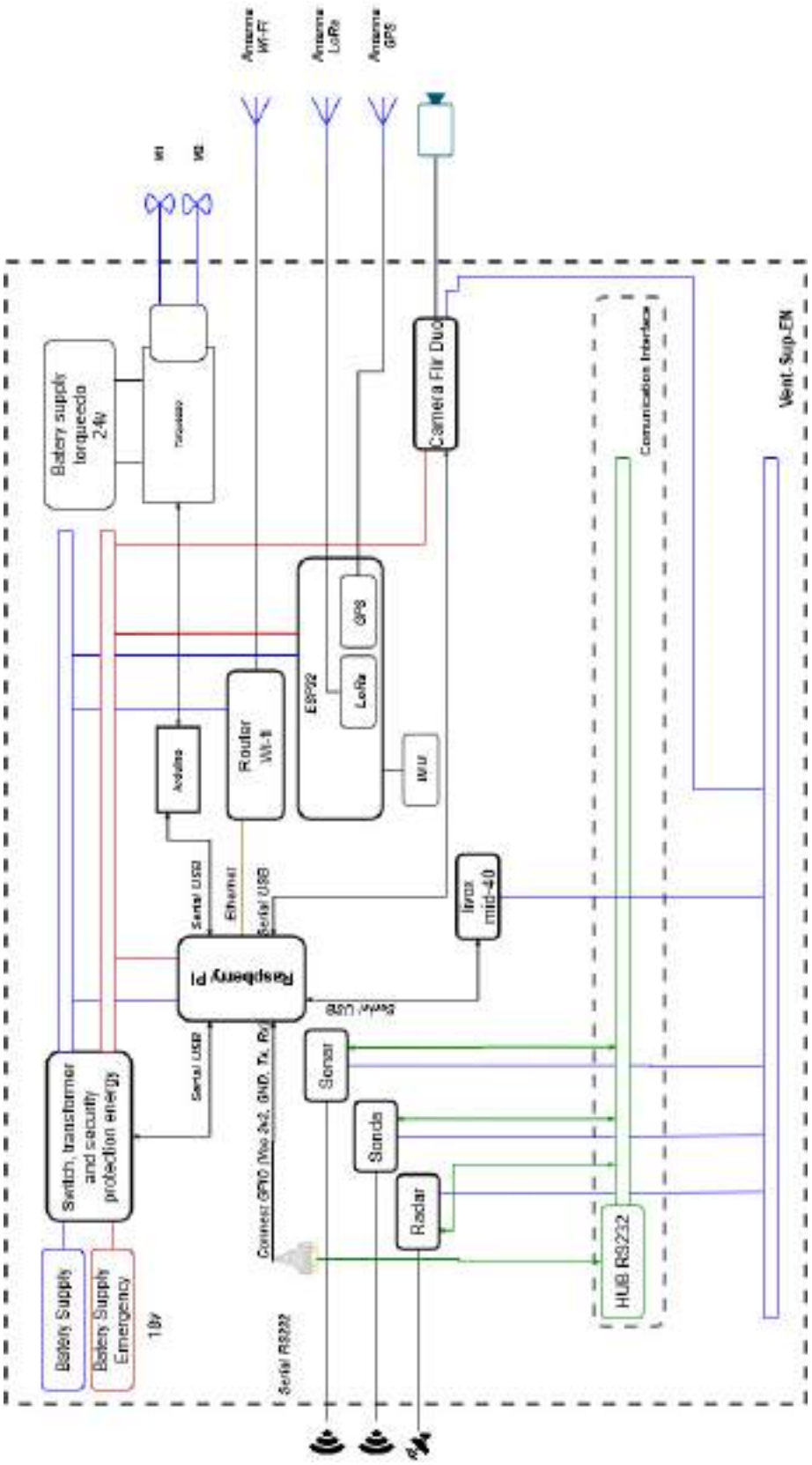
Tech, S. (2015). *CAT-Surveyor [Brochure]*. https://www.subsea-tech.com/cat-surveyor/

Yoshida, H., Fujimoto, H., Kawano, D., Goto, Y., Tsuchimoto, M. & Sato, K. (2015). Range extension autonomous driving for electric vehicles based on optimal velocity trajectory and driving braking force distribution considering road gradient information. *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 4754–4759. https://doi.org/10.1109/IECON.2015.7392843

YSI. (2020). *USV Hycat*. https://www.ysi.com/hycat

Zhu, Q. (2013). Design of control system of USV based on double propellers, IEEE. https://doi.org/10.1109/TENCON.2013.6719057

Apêndice A - Arquitetura de Hardware



Apêndice B - Arquitetura de Software