Ricardo André Cabaço Santos

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Prova de varrimento de área em competição de veleiros com navegação autónoma

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Luís Filipe Santos Gomes,
Professor associado com agregação,
Universidade Nova de Lisboa

Júri

Presidente: Doutor André Teixeira Bento Damas Mora
Arguente: Doutor Tiago Oliveira Machado de Figueiredo Cardoso
Vogal: Doutor Luís Filipe Santos Gomes

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Prova de varrimento de área em competição de veleiros com navegação autónoma

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Agradecimentos

Agradecer em primeira instância ao professor Luís Gomes pelos seus conselhos e pela disponibilidade e acessibilidade demonstradas enquanto meu orientador.

Gostaria também de agradecer à professora Anikó Costa por ter sido também um apoio importante ao longo desta dissertação.

Enaltecer também a ajuda prestada pelos colegas Pedro Maricato e João Andrade, essenciais para ultrapassar alguns obstáculos que foram aparecendo nesta dissertação.

Agradecer aos meus pais, ao meu irmão, e à Inês, pelo que representam para mim e pelo apoio que sempre me deram.

Por fim, agradecer aos meus fieis colegas, que me acompanharam desde o inicio do curso até à data.

Resumo

A competição internacional de veleiros autónomos WRSC (*World Robotic Sailing Championship*) realiza-se anualmente tentando reunir a comunidade académica ativa na área. Esta competição é composta por quatro provas, sendo que o objetivo desta dissertação foi implementar uma solução para uma delas: a prova de varrimento em área. Esta prova consiste em varrer uma área definida por bóias sinalizadoras e compostas por quadrículas, tendo uma duração máxima de 30 minutos. Cada quadrícula equivale a um ponto e é apurado o vencedor da prova que percorrer um maior número de quadrículas num período mais curto.

Antes de abordar diretamente o problema, foi necessário entender primeiramente o contexto do mesmo, mais especificamente, alguns conceitos de navegação à vela, outros trabalhos relacionados, tecnologias utilizadas e claro, o projeto base que vem sido implementado ao longo dos anos por antigos alunos.

Foi proposta uma solução flexível, onde o veleiro deve ter capacidade de decidir a sua estratégia de navegação consoante a leitura inicial (antes da prova começar) da direção do vento, tendo várias opções de tipos de varrimento. Além disso, também possui um mecanismo que permite recalcular o trajeto de varrimento na ocorrência de dificuldades de navegação, por exemplo, para evitar ser desqualificado (não exceder o tempo máximo da prova).

A solução, depois de implementada, foi submetida a testes, nomeadamente validação do cálculo do trajeto, simulações de navegação e por último um exercício organizado pela Marinha Portuguesa que permitiu o teste do veleiro no mar.

Palavras-chave: Veleiro, WRSC, Varrimento em área, Navegação, Estratégia, Tipos de varrimento, Trajeto

Abstract

The international autonomous sailboat competition WRSC (World Robotic Sailing Championship) is an international competition of autonomous sailboats which is held annually to bring together the academic community active in the area. This competition is constituted by four tests, and the purpose of this dissertation was to implement a solution for one of them: the area scanning test. This test consists of scanning an area marked by buoys and composed of grids, having a maximum duration of 30 minutes. Each grid corresponds to one point and the sailboat that travels the largets number of squares in less time is considered winner.

Before addressing the problem directly, it was necessary to first understand the context of the problem, more specifically some sailing concepts, other related works, technologies used and the basic project that has been implemented over the years by previous students.

A flexible solution has been proposed, where the sailboat must be able to decide its navigation strategy according to the initial (before the competition starts) wind direction reading, having several options of scanning types. In addition, it also has a mechanism that allows the recalculation of the scan route in the event of navigation difficulties, for example, to avoid being disqualified (not exceeding the maximum time of the test).

The solution, once implemented, was tested, namely the validation of the route calculation, navigation simulations and finally an exercise organized by the Portuguese Navy that allowed the sailboat to be tested at the sea.

Keywords: Sailboat, WRSC, Area scanning, Navigation, Strategy, Scanning types, Route

Índice

Lista de Figuras

Lista de Tabelas

Glossário

Checkpoints: Goal predenido que representa uma ”meta” que o veleiro tem de obrigatoriamente atravessar antes de um determinado tempo. Caso contrário, a trajetória de varrimento é recalculada.

Deadline: Tempo máximo estabelecido para o veleiro atravessar o *checkpoint* associado.

Estratégia de navegação: Abordagem (horizontal ou vertical) de navegação à prova de varrimento, em relação à orientação da linha de partida. Esta estratégia é decidida tendo em conta a direção do vento sobre o veleiro antes da prova começar.

Goal: Ponto geográco que representa um destino para o qual o veleiro tem de se deslocar..

Método de varrimento: Conceito composto por um conjunto de *goals* e *trajetórias* que dene o varrimento da área (ex: *Horizontal Fast Scanning*).

Profundidade de varrimento: Conceito associado à quantidade de área/quadrículas percorridas no varrimento. Pode ser: *fast*, *middle*, *deep e full*.

Trajetória: Percurso esperado entre um ponto inicial e um ponto final.

SIGLAS

ACK/NACK *Acknowledgement/Negative-aknowledgement*.

FPGA *Field Programmable Gate Array*.

GPS *Global Positioning System*.

I2C *Inter-Integrated Circuit*.

IDE *Integrated Development Environment*.

INNOC *Austrian Association for Innovative Computer Science*.

PWM *Pulse Width Modulation*.

REX *Robotics EXercise*.

SCA *Serial Data Line*.

SCL *Serial Clock Line*.

UDP *User Datagram Protocol*.

WRSC *World Robotic Sailing Championship*.

Capítulo 1

Introdução

1.1 Motivação e Enquadramento

A navegação marítima é uma prática bastante antiga e um dos meios de transporte primordiais, tendo inicialmente sido adotada para fins comerciais e exploração de território. Milhares de anos mais tarde, com os avanços da Ciência e da Tecnologia, esta prática estende-se por várias áreas, com embarcações e equipamentos altamente sofisticados e eficientes.

Nos últimos anos, o progresso do mundo digital resultou na ambição e posteriormente na necessidade de criar sistemas autónomos capazes de operar sem atuação direta do ser humano, o que levou inevitavelmente ao cruzamento entre o "mundo digital autónomo" e a navegação marítima.

Contextualizando esta dissertação, existem já vários modelos de veleiros capazes de navegar autonomamente, através da combinação de controladores, sensores e atuadores, conseguindo contornar e evitar obstáculos sem intervenção humana, tendo sempre em conta os fatores externos que interferem diretamente na performance da navegação (ondulação, velocidade e direção do vento, etc.).

1.2 Problema

A Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa participou, pela última vez em 2016, numa das maiores competições internacionais de veleiros autónomos - *World Robotic Sailing Championship* (secção 1.2.1) - tendo também possibilidade de voltar a participar nos anos que se avizinham. Esta competição é composta por quatro provas, sendo uma delas a prova de Varrimento em Área (*Area Scanning*), que será a prova na qual esta dissertação irá incidir.

1.2.1 *World Robotic Sailing Championship*

Associada à iniciativa Microtransat, a WRSC (*World Robotic Sailing Championship*) foi organizada pela INNOC (*Austrian Association for Innovative Computer Science*) e teve a sua primeira edição em 2008, na Áustria. Desde então para cá, esta competição é organizada anualmente por diferentes países de ano para ano, sendo Portugal o único país que repetiu a organização da WRSC (em 2009 e 2016).

Tratando-se de uma competição de veleiros autónomos, existem algumas regras que estes devem respeitar [1]:

- Os veleiros não podem ultrapassar os 4 metros de comprimento;
- Têm de ser totalmente autónomos, quer em termos de energia quer em termos de controlo e decisão;
- Cada veleiro tem de ser capaz de carregar ou gerar energia elétrica suficiente para um dia inteiro de operação;
- O único meio de propulsão permitido é o vento;
- Operações remotas podem ser efetuadas excecionalmente e temporariamente para evitar certos tipos de colisão.

O vencedor final é apurado consoante as prestações nas 4 provas desta competição, sendo elas: *fleet race*, *station keeping*, *obstacle avoidance* e *area scanning* (apresentadas nas secções seguintes).

1.2.1.1 *Fleet race* (Regata)

Nesta prova (onde o artigo [2] se encontra relacionado) os veleiros competem entre si num género de corrida por etapas. Estes têm de chegar à linha da meta contornando as bóias sinalizadas e percorrendo um trajeto semelhante ao da figura 1.1.

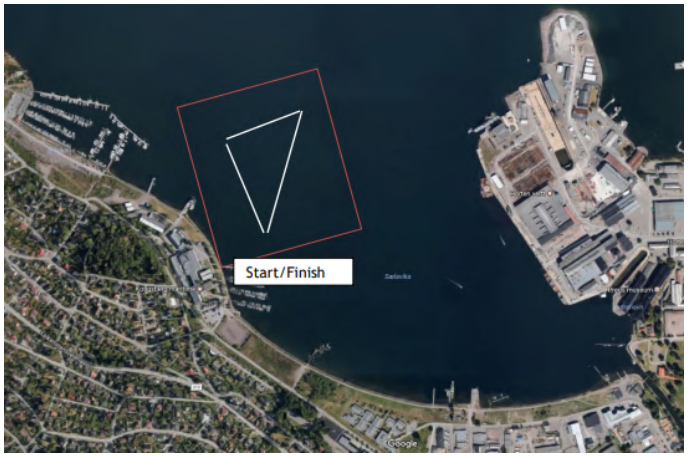

Figura 1.1: Prova *Fleet Race* da WRSC em 2017, em Horten, Noruega [1].

1.2.1.2 *Station keeping* (Ancoragem virtual)

Para avaliar a capacidade do veleiro de ancorar virtualmente, foi introduzida a prova de *Station Keeping*, onde o veleiro tem de se manter o mais próximo possível de um determinado ponto P durante 5 minutos, após entrar num circulo com raio de 20 metros à volta desse mesmo ponto (figura 1.2).

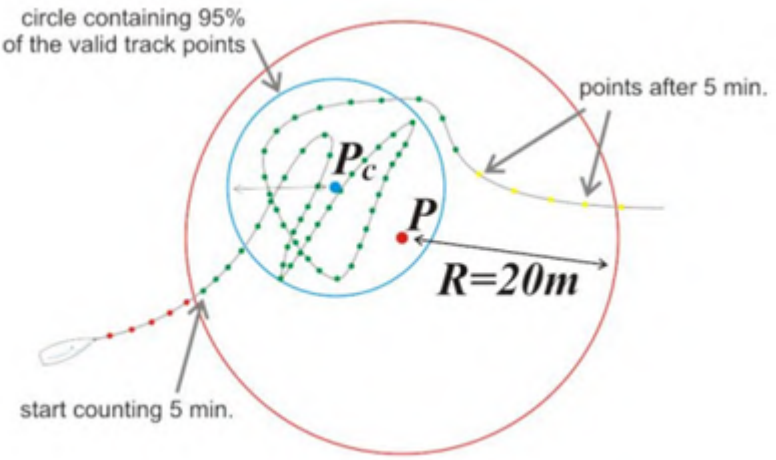

Figura 1.2: Esboço da prova *Station Keeping* [1].

1.2.1.3 *Obstacle avoidance* (Desvio de obstáculos)

A prova de contorno de obstáculos visa avaliar a capacidade do veleiro detetar e posteriormente se desviar de um obstáculo que surja inesperadamente no seu trajeto.

A área de navegação é constituída por 4 pontos, formando um retângulo com as dimensões 150x20 m (figura 1.3). O veleiro deve entrar no retângulo por uma das extremidades (20 m) e navegar até à extremidade oposta, invertendo o sentido depois de a atravessar.

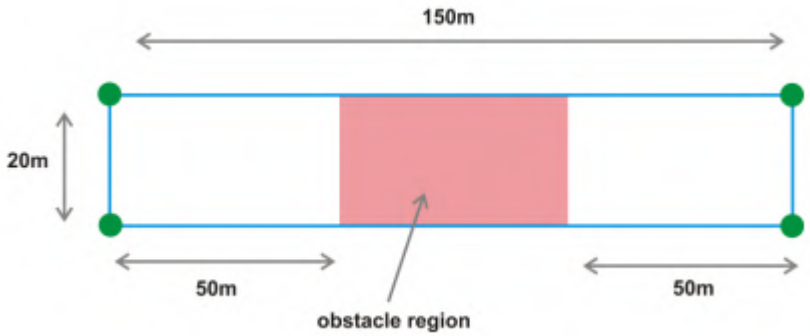

Figura 1.3: Área de navegação da prova *Obstacle Avoidance* [1]

Após concluído este trajeto duas vezes, é colocado um obstáculo no meio do retângulo de navegação. O veleiro deve então repetir o processo anteriormente descrito, mas agora

tendo de se desviar do obstáculo (sem haver contacto físico) e voltar para dentro do retângulo para atravessar a extremidade oposta, invertendo o sentido e percorrendo pelo menos mais uma volta (artigo relacionado [3].

1.2.1.4 *Area Scanning* (Varrimento em área)

Esta irá ser a prova na qual esta dissertação vai incidir. O objetivo desta prova de varrimento é avaliar a capacidade do veleiro percorrer o máximo de área possível dentro de um espaço delimitado (figura 1.4). Este espaço está dividido em quadrados de 20x20 metros e o veleiro deve visitar o máximo de quadrados possíveis dentro do período de tempo disponível (30 minutos). A prova começa quando o veleiro passa a linha de partida e acaba quando cruza a linha de chegada, antes de esgotar o tempo máximo permitido.

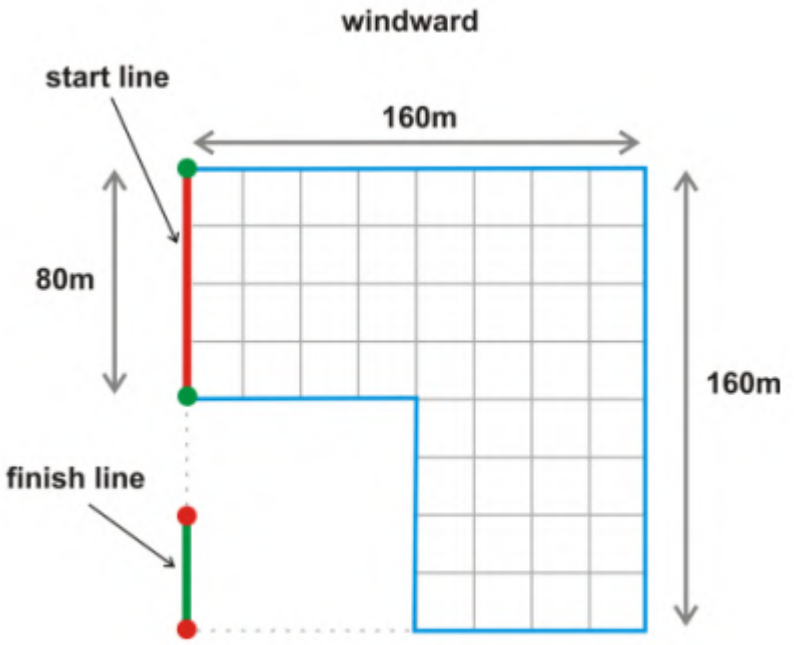

Figura 1.4: Prova de *Area Scanning* da WRSC em 2017, em Horten, Noruega [1].

Scoring: como foi referido acima, quantos mais quadrados o veleiro percorrer no tempo estabelecido, mais pontos arrecada, obtendo um melhor ranking (figura 1.5).

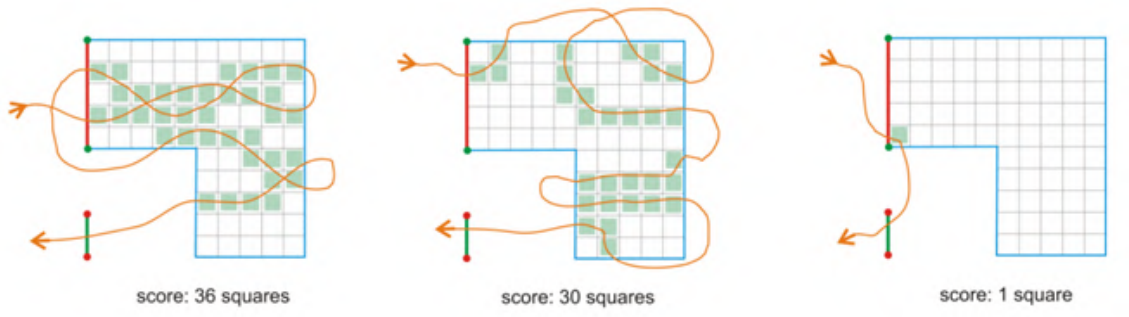

Figura 1.5: Exemplos de pontuações [1].

Em caso de igualdade no número de quadrados percorridos entre dois ou mais veleiros, obtém uma melhor classificação aquele que tiver percorrido um trajeto mais curto, com é possível ver na figura 1.6.

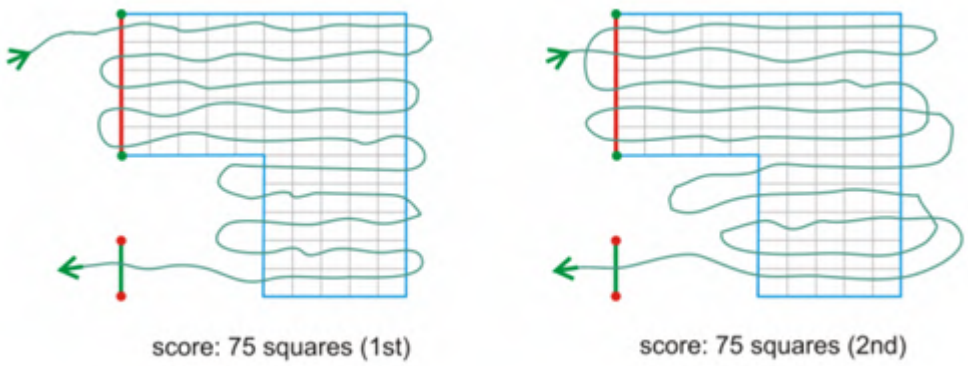

Figura 1.6: Exemplo de empate na pontuação entre dois veleiros [1].

O requisito mínimo para a conclusão da prova é atravessar a linha de partida e mais tarde a linha de chegada. O incumprimento deste requisito resulta na posição mais baixa do ranking desta prova (figura 1.7).

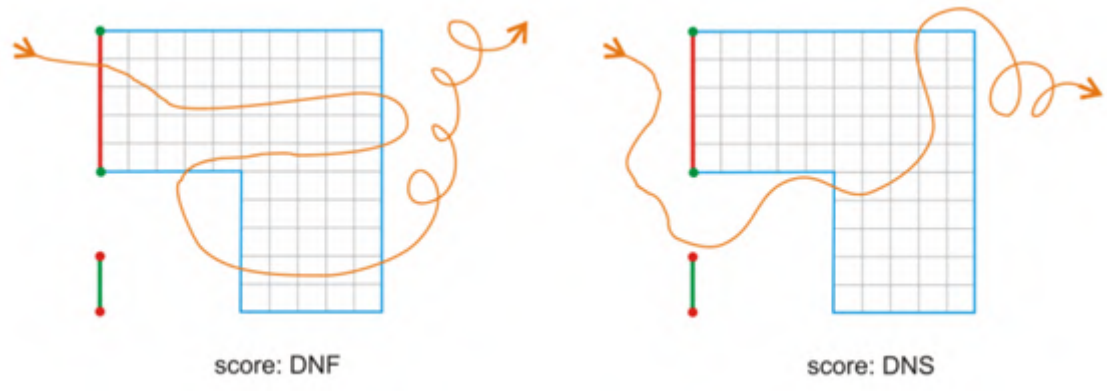

Figura 1.7: Exemplos de DNF (*Did Not Finish*) e DNS (*Did Not Start*) [1].

Nota: . A área representada nas figuras 1.5, 1.6 e 1.7 (200 x 200 metros) diz respeito à classe *"Sailboat"* (veleiros entre 1.5 metros e 4.5 metros de comprimento e menos de 500 kg de peso). No entanto as mesmas regras aplicam-se à área da classe do veleiro desta dissertação, *"Micro-Sailboat"* (comprimento menor que 1.5 metros e peso menor que 100 kg).

1.3 Objetivos

Esta dissertação foi dividida em pequenos objetivos, que foram sendo atingidos sequencialmente de forma a construir a solução para o problema principal. Primeiramente o objetivo passou por compreender alguns conceitos de navegação à vela e também o trabalho realizado em projetos e dissertações anteriores sobre o veleiro que irá ser abordado; depois, perceber o modo de funcionamento e leitura dos sensores associados, assim como todo o processo de interpretação do sistema aos dados recolhidos pelos sensores e a consequente interação com os atuadores (motores). A partir deste ponto, foi necessário refletir e decidir acerca da estratégia/solução para abordar o problema, prosseguindo para a sua implementação. No final, validar e testar o que foi implementado e analisar os resultados.

1.4 Estrutura do documento

O documento apresentado é composto por cinco capítulos: o primeiro capítulo de introdução; o segundo capítulo "Conceitos e Trabalhos Relacionados", onde são apresentados alguns conceitos de navegação à vela, outros projetos no âmbito do problema e explicação do veleiro relacionado com esta dissertação e o projeto associado; o terceiro capítulo onde é apresentada a solução para o problema; o capítulo quatro, que diz respeito à implementação desta solução; o quinto capítulo, que apresenta os testes e validações feitas à implementação e posterior análise; e o capítulo final, onde são feitas as conclusões e considerações finais, abordando também os trabalhos futuros.

Capítulo 2

Conceitos e Trabalhos relacionados

2.1 Navegação à vela

A navegação é um conceito muito abrangente. Existem vários tipos de embarcações, que praticam diferentes tipos de navegação e que por conseguinte são controlados de formas distintas.

A navegação à vela é um desses exemplos, diferenciando-se das restantes navegações principalmente por usar o vento como único meio de propulsão (entre outras coisas), onde a embarcação usada nesta prática é o veleiro.

Antes de abordar a parte autónoma, é necessário ter em conta alguns conceitos básicos relativos à navegação à vela, cuja definição é dada pelo deslocamento e controlo de uma embarcação através do manuseamento das velas desta.

2.1.1 O veleiro

Como foi referido acima, veleiro é o nome dado à embarcação usada na navegação à vela [4]. Os principais constituintes deste são:

- Casco - invólucro exterior do veleiro, que permite que este flutue;
- Leme - tem a função de controlar a direção da embarcação;
- Quilha - confere estabilidade ao veleiro e permite que este se desloque sempre em frente;
- Proa - parte da frente do veleiro;
- Popa - parte traseira do veleiro;
- Mastro - estrutura que sustenta as velas;

- Vela Mestre - vela principal;

- Estai - vela secundária;

Figura 2.1: Constituição do veleiro [5].

2.1.2 Vento aparente

O vento é o fator externo mais importante neste tipo de navegação, pelo que é necessário ter em conta o conceito de vento aparente [6]. O vento aparente é o vento que é sentido pelo referencial da embarcação à medida que esta se desloca, resultado da combinação (vetorial) do vento real e do vento induzido (que é simétrico ao vetor da velocidade de deslocação do veleiro), como se pode verificar na figura abaixo.

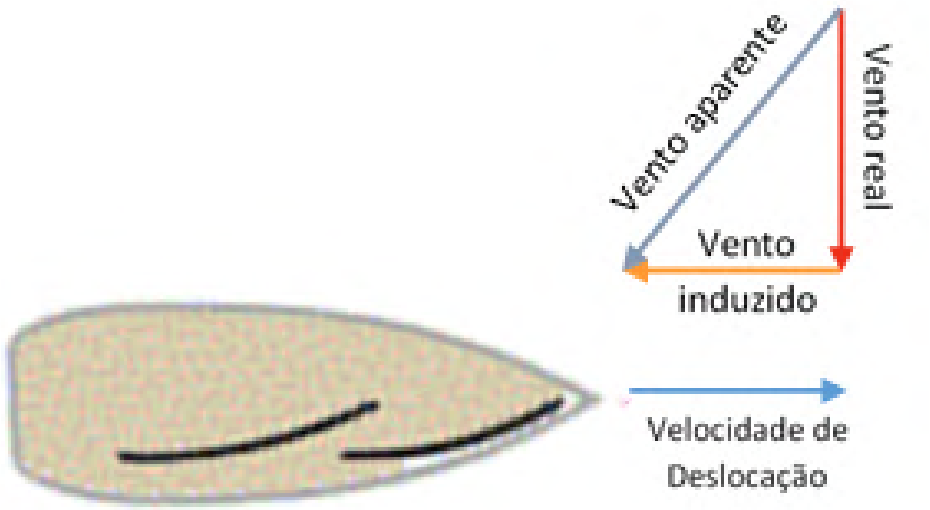

Figura 2.2: Combinação vetorial do vento induzido, aparente e real.

2.1.3 Vento e a influência da sua direção

Sendo o vento o "combustível" de um veleiro, a sua navegação está diretamente condicionada pela direção do vento. Casos como fraca ou nula intensidade, intensidade demasiado forte e inconstante ou direções não desejáveis são características do vento que complicam a navegação à vela e podem até impossibilitar a sua prática.

Assim, é importante estudar e conhecer os diferentes tipos de incidência do vento na embarcação e como se pode tirar partido destes fatores, para uma melhor modelação e implementação da sua navegação autónoma.

É necessário também ter em conta certos conceitos, como arribar e orçar. Quando a proa de um barco se aproxima da direção contrária do vento, diz-se que está a orçar. Por outro lado, quando se aproxima da direção do vento, diz-se que está a arribar.

Na figura abaixo encontra-se uma imagem representativa dos diferentes tipos de direção do vento e as suas nomenclaturas [7].

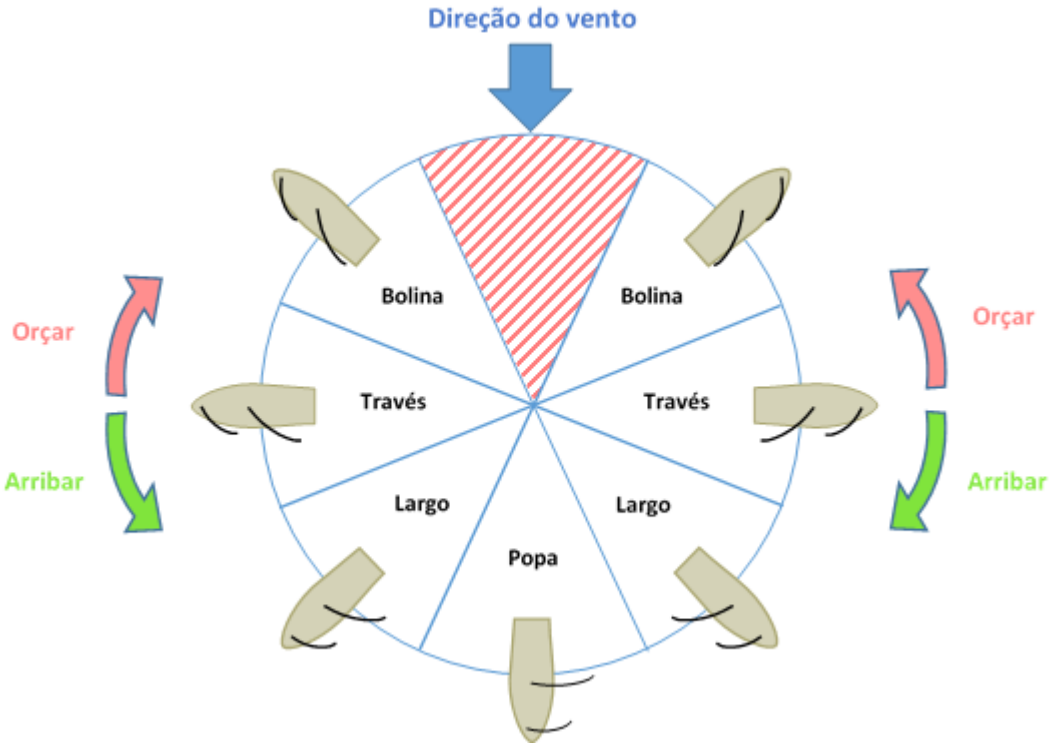

Figura 2.3: Cenários de incidência do vento sobre a embarcação.

Como se pode observar no esquema acima, existem quatro zonas cuja direção do vento é favorável à navegação: Popa, Largo, Través e Bolina.

- **Popa:** situação em que o sentido do vento é da popa para a proa, ou seja, a embarcação e o vento têm a mesma direção. Neste caso, as velas devem ser folgadas (dar folga às velas, para que estas se afastem do bordo e "recebam" mais vento) ao máximo, ficando totalmente perpendiculares à direção do vento;

- **Largo:** cenário em que o vento e o veleiro já não estão alinhados, mas o ângulo de incidência continua a ser elevado, pelo que as velas devem estar menos folgadas que a situação anterior;

- **Través:** o eixo do barco é perpendicular à direção do vento. As velas devem estar meio folgadas/caçadas (caçar é o termo usado para aumentar a tensão nos cabos das velas, de forma a que estas se aproximem do bordo e se tornem mais planas);

- **Bolina:** cenário em que o ângulo de incidência é pequeno, navegando quase conta o vento. As velas devem estar caçadas quase na totalidade, para conseguirem obter alguma impulsão. É a situação em que a navegação é mais lenta.

Não é possível navegar quando a embarcação está em sentido contrário com a direção do vento (zona a vermelho), pois as velas não têm forma de aproveitar a força do vento para aplicar uma velocidade ao veleiro. Assim, nestas situações a solução é navegar em

"ziguezagues", normalmente com um ângulo de 45º (navegar em bolina, portanto), como mostra a figura 2.4.

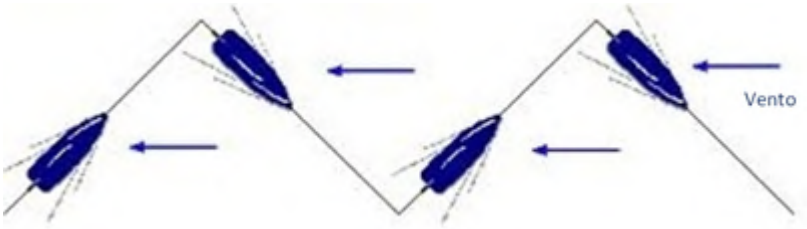

Figura 2.4: Navegação contra o vento [7].

2.1.4 Leme e a mudança de direção

Para compreender como um veleiro é capaz de mudar de direção, é necessário perceber primeiro o efeito que o leme tem na embarcação [8]. A figura 2.5 mostra a situação em que o veleiro se desloca em frente, com o leme orientado de acordo com o eixo da embarcação.

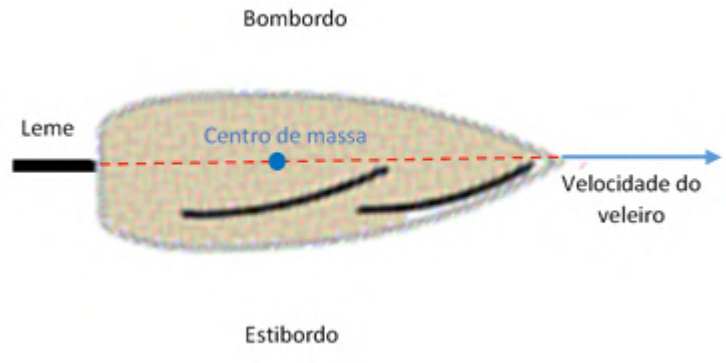

Figura 2.5: Leme orientado de acordo com a direção da embarcação.

No instante em que se gira o leme para estibordo, é exercida uma força sobre este, perpendicular ao eixo do veleiro e orientada para bombordo. Esta força provoca um momento sobre o centro de massa do veleiro, no sentido dos ponteiros do relógio, como ilustra a figura 2.6.

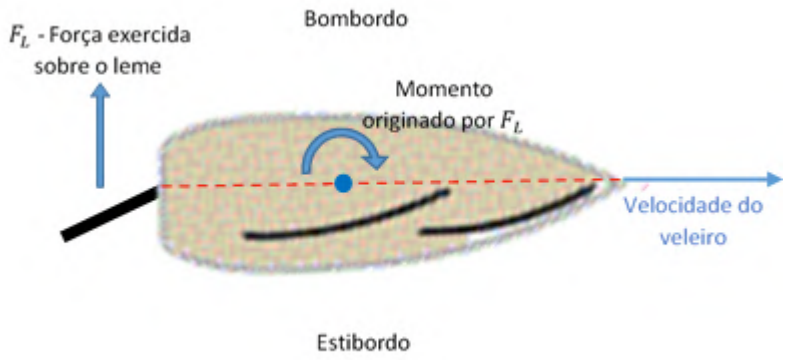

Figura 2.6: Momento sobre o centro de massa do veleiro.

No entanto, este momento sobre o centro de massa é demasiado pequeno para rodar significativamente a embarcação, dado que o leme tem dimensões e massa bastante inferiores comparativamente ao veleiro em si. Assim, a ação anterior resulta apenas numa ligeira rotação do eixo do barco em relação ao seu curso (figura 2.7). Esta ligeira rotação provoca uma decomposição do vetor da velocidade, pois o eixo do veleiro já não está orientado com a velocidade inicial.

Verifica-se então que o veleiro possui uma velocidade (aplicada ao seu centro de massa) perpendicular ao seu eixo, orientada para bombordo, que representa a oscilação resultante da ligeira rotação ocorrida.

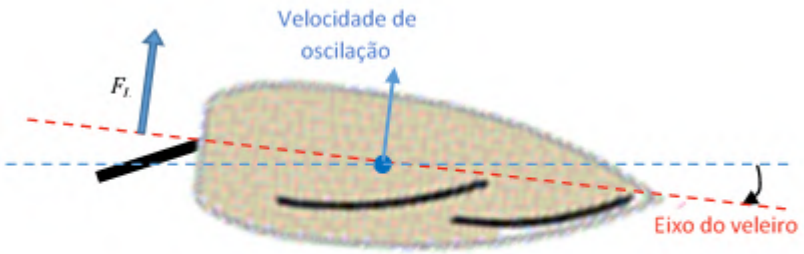

Figura 2.7: Ligeira rotação resultante do momento sobre o centro de massa.

Esta velocidade provoca uma oscilação da embarcação no sentido bombordo. Quando isto acontece, as partículas da água exercem uma força contra o casco do veleiro em sentido contrário, devido à sua inércia (a direção de uma força de inércia é sempre oposta ao sentido do movimento). A força de inércia incidente no casco pode dividir-se em 2 grupos: força incidente na proa e força incidente na popa (figura 2.8).

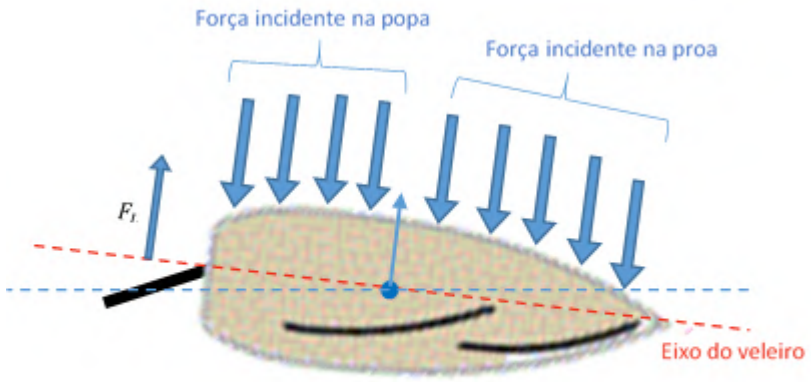

Figura 2.8: Forças de inércia incidentes na popa e na proa.

O casco da embarcação é construído de maneira a que a força de inércia incidente na proa seja maior que na popa. Desta forma, é criado um momento hidrodinâmico em torno do centro de massa do veleiro, originando uma rotação significativa do eixo do veleiro (figura 2.9).

Figura 2.9: Momento hidrodinâmico sobre o centro de massa do veleiro e respetiva rotação.

2.2 Projetos relacionados

2.2.1 FASt (FEUP *Autonomous Sailboat*)

Desenvolvido em 2007 pelo Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto [9], este veleiro (que possui 2,5 metros) teve como objetivo entrar em competições internacionais organizadas pela *Microtransat*.

Figura 2.10: Veleiro FASt na WRSC em 2012 [10].

O sistema eletrónico de navegação e controlo [11] é composto por uma plataforma reconfigurável contendo uma FPGA (*Field Programmable Gate Array*) - Suzaku SZ130 - ligada aos sensores que fazem parte do sistema:

- GPS;
- Anemómetro - sensor usado para medir a velocidade do vento;
- Sensor de direção do vento;
- Sensor medidor do ângulo da vela;
- Bússola;
- Inclinómetro.

Além destes sensores, o veleiro também possui uma antena de Rádio Comando (para controlar o mesmo), uma antena *Wi-Fi* para transmissão de dados a curtas distâncias, uma antena IRIDIUM para comunicação por satélite, baterias para alimentar o sistema e um painel solar para gerar energia.

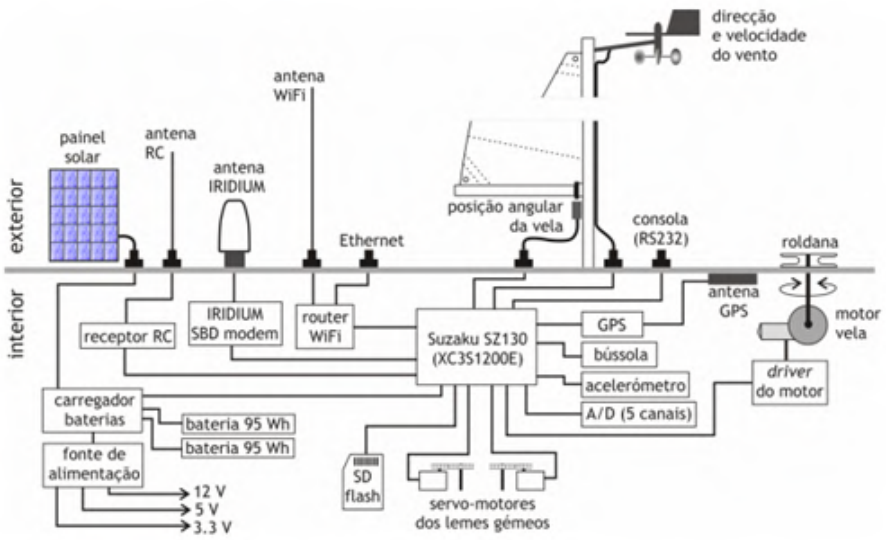

Figura 2.11: Arquitectura do *hardware* do FASt [11].

No que toca ao software, este foi desenvolvido em C com o auxílio de bibliotecas padrão de Linux (também disponíveis no uCLinux). O sistema foi repartido em módulos, cada um com a sua própria função (ex: interface, módulo de *input/output*, etc), como se pode observar na figura 2.12. No que toca aos módulos de controlo e navegação foram divididos em processos que comunicam entre si através de *sockets* UDP.

Foi também desenvolvida uma aplicação virtual para aceder diretamente a todos os periféricos do sistema - vapif - permitindo testar vários cenários de navegação, juntamente com outra aplicação - vaplotter - para a monitorização em tempo real do veleiro, registando graficamente a posição da embarcação e todos os parâmetros relevantes para o controlo e a navegação.

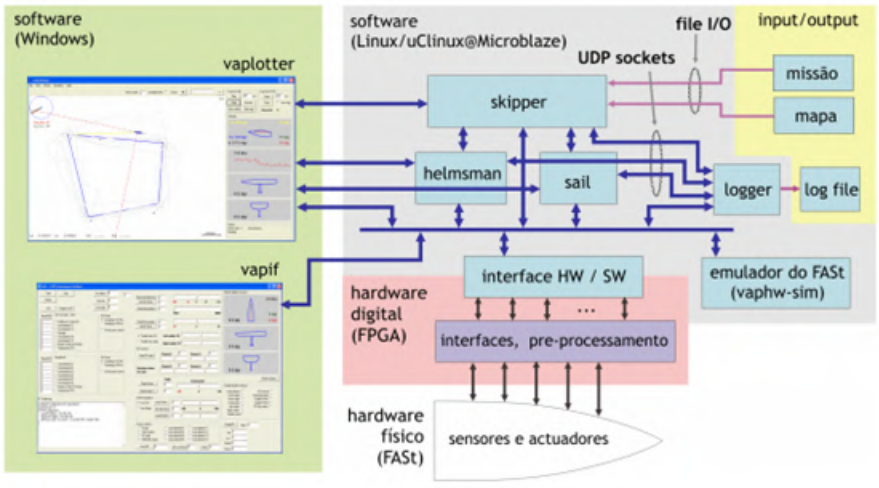

Figura 2.12: *Software* de modulação e implementação do FASt [11].

2.2.2 Avalon - *Autonomous Sailboat*

Avalon é um veleiro autónomo que foi desenvolvido pelo Laboratório de Sistemas Autónomos (ASL) do Instituto Federal de Tecnologia (ETH) de Zurique, para participar também na *Microtransat Challenge*, tendo como principal objetivo atravessar o oceano Atlântico.

Figura 2.13: Avalon [12].

O seu *hardware* [13] é constituído por um computador de bordo (MPC21 da *Digital-Logic*) que controla todos os sensores e periféricos (o "cérebro" do sistema), por um GPS, IMU (*Inertial Measurement Unit* - fornece a informação sobre a velocidade e aceleração em todos os 6 graus de liberdade), sensor de direção e velocidade do vento, AIS (*Automatic Identification System* - recebe dados relativos à posição e orientação de outras embarcações, evitando assim colisões com estes) e um modem IRIDIUM 9522-B para comunicações via satélite. No que toca à alimentação do sistema, o Avalon possui 4 paneis solares que armazenam a energia em baterias de lítio-manganês. O *software* implementado baseou-se em DDX (*Dynamic Data eXchange*), que é uma plataforma capaz de lidar com todas as atividades de armazenamento, representando a ligação entre uma memória central partilhada e os vários subprogramas independentes (figura 2.14).

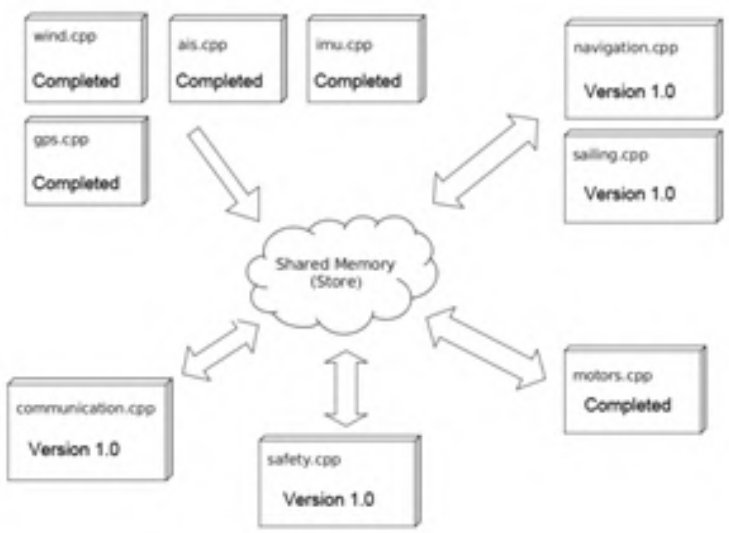

Figura 2.14: DDX - memória partilhada e os subprogramas independentes [13].

2.2.3 Dewi'r Ddraig - *AberSailbot*

O veleiro Dewi'r Ddraig [14] começou a ser projetado em 2012 por um grupo de alunos da Universidade de Aberystwyth, País de Gales, para competir nas competições WRSC e IRSR (*International Robotic Sailing Regatta*).

O hardware do Dewi'r Ddraig possui um Raspberri Pi como micro controlador central, tendo sido usado o BoatD como software de gestão de controlo do sistema e Python como linguagem de programação.

Figura 2.15: Dewi'r Ddraig, possui 1.3 metros de comprimento [14].

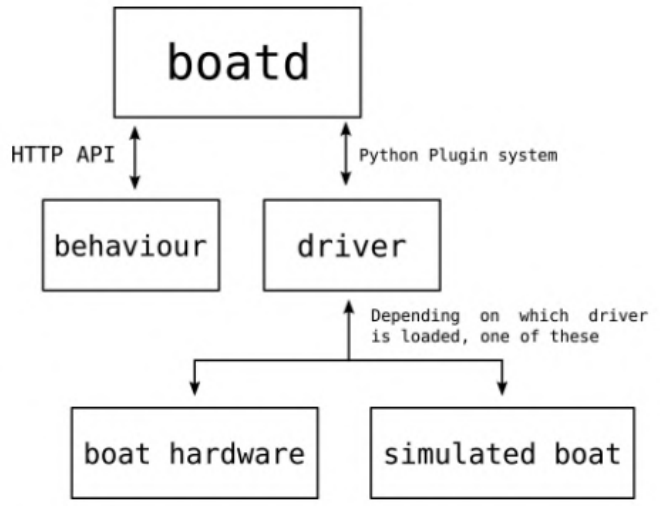

Figura 2.16: Arquitetura do BoatD [14].

2.2.4 Black Python

O projeto Black Python [15] foi iniciado pela *Southampton Sailing Robot Team* (maioritariamente formada por alunos da Universidade de Southampton). O modelo do barco possui 1 metro de comprimento, e o seu *hardware* é constituído por um micro controlador Raspberri Pi, GPS, IMU, uma câmara e um sensor de direção do vento.

Figura 2.17: Black Python [16].

O micro controlador trabalha com o ambiente ROS (*Robot Operating System*) e o seu código é escrito em Python.

2.3 eVentos - veleiro relacionado com esta dissertação

O veleiro eVentos foi desenvolvido pela Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa com o objetivo de participar na competição WRSC, sendo um projeto que esta dissertação visa complementar de forma a participar na prova de varrimento desta mesma competição.

Para que seja possível a navegação autónoma do veleiro, este tem de possuir um controlador. Neste caso, o controlador foi implementado segundo três níveis hierárquicos [17]:

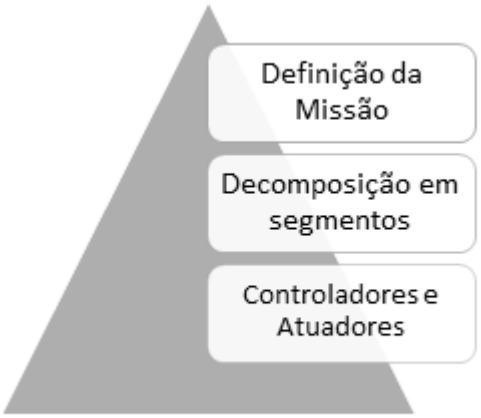

Figura 2.18: Níveis hierárquicos do sistema eVentos.

A camada superior é onde é definido o trajeto, através de uma sequência de segmentos que são disponibilizados ao nível inferior.

O segundo nível tem a função de decompor a missão em segmentos e definir qual o melhor método de navegação para cada segmento, tendo em conta os fatores externos como a intensidade e direção do vento.

O nível mais inferior diz respeito aos controladores do sistema de navegação do veleiro, nomeadamente o controlo das velas e do leme [18].

2.3.1 *Hardware*

2.3.1.1 Arquitectura

Tendo em conta os níveis hierárquicos descritos acima, a arquitetura de *hardware* do veleiro foi definida da seguinte forma:

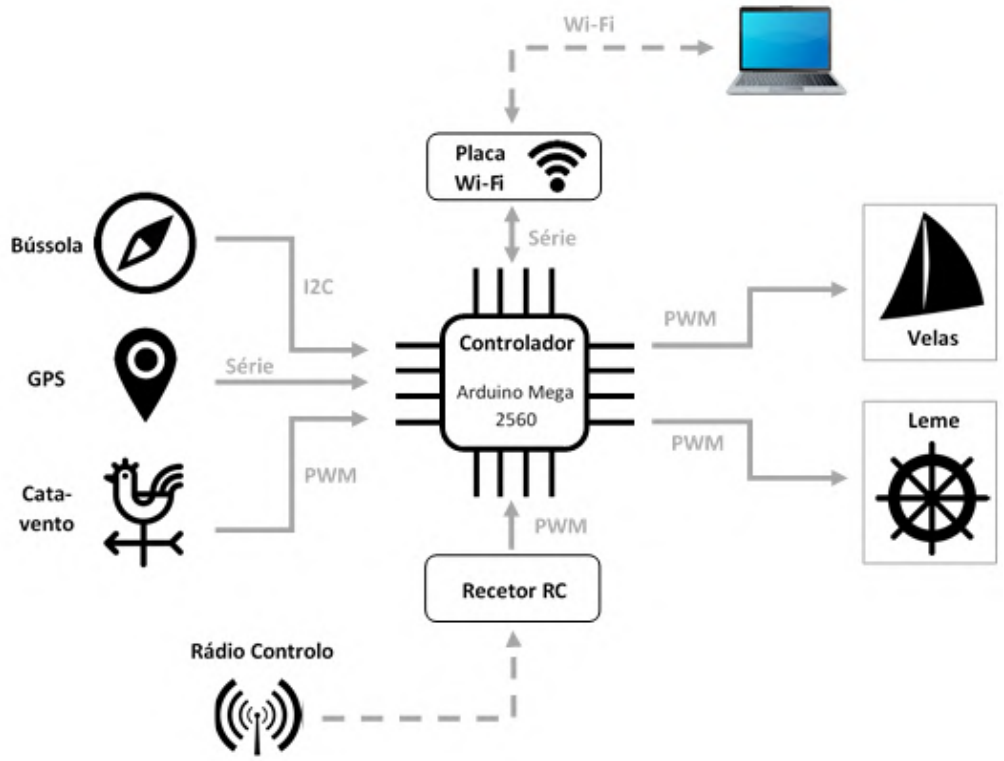

Figura 2.19: eVentos - arquitetura.

O controlador (Arduino Mega 2560) recebe a informação do mundo exterior através de três sensores: bússola, GPS e sensor de direção do vento (cata-vento). Além disso, o sistema possui um recetor rádio para possibilitar o controlo remoto do veleiro e uma placa *Wi-Fi* para trocar dados entre o sistema e o computador. O Arduino controla também dois motores servos, que atuam nas velas e no leme da embarcação.

2.3.1.2 Protocolos de comunicação

Comunicação Série:

Este tipo de comunicação opera apenas numa única linha de transmissão de dados, onde só é enviado um bit de cada vez. A comunicação série [19] pode ser síncrona, havendo o mesmo sinal de *clock* para os diferentes dispositivos, ou pode ser assíncrona, onde não existe um sinal de *clock* para coordenar a transmissão de dados, pelo que é necessário conciliar as velocidades de relógio diferentes entre o controlador e o periférico.

I2C:

O I2C [20] é um protocolo que se baseia nos conceitos de *"Master"* e *"Slave"*. Estes comunicam através um bus que consiste em duas linhas de série: SCL (*Serial Clock Line*) e SDA (*Serial Data Line*). A transmissão de dados na SDA acompanha o ritmo da SCL (o clock é gerado pelo *Master*, contudo o *Slave* pode abrandar o sinal de relógio para ter tempo de preparar o *"request"*), onde a mensagem é dividida em duas *frames*: a primeira com o endereço do *Slave* e a segunda com a informação destinada ao *Slave*.

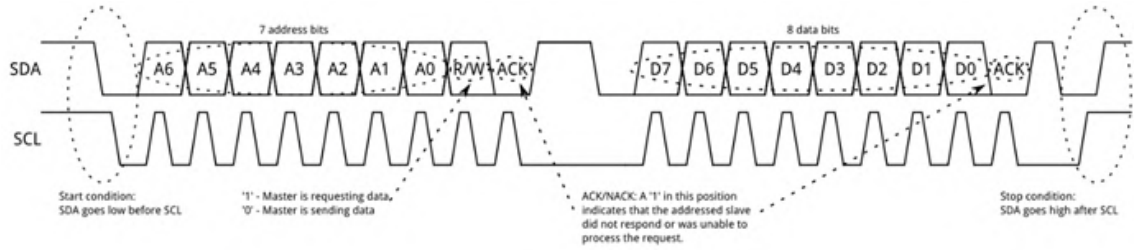

Figura 2.20: SDA e SCL [21].

Para iniciar a comunicação com um *Slave*, o *Master* coloca o SCL a "*high*" e o SDA a "*low*", fazendo com que todos os *Slaves* escutem a comunicação. O resto da primeira *frame* é composta pelo endereço do *Slave* destino (7 bits), pelo bit de R/W (*Read/Write*) e o ACK/NAK (*Acknowledge/Not Acknowledge*).

A *frame* seguinte contém os dados da mensagem a enviar, seguidos de um bit ACK/-NAK, podendo ser originada pelo *Master* ou pelo *Slave*, dependendo do bit de R/W da *frame* anterior.

PWM - Pulse Width Modulation:

PWM [22] é um método de modulação que usa um sinal digital constituído por pulsos para controlar um dispositivo analógico.

A ideia deste método é modelar um pulso mantendo o período do sinal intacto e variando o tempo em que o pulso se encontra com o valor lógico "*high*". Esta relação chama-se *Duty Cycle*, que matematicamente se representa por: *Duty Cycle* = Período "high" do pulso / Período do sinal.

Este tipo de sinal digital possui um valor médio (figura 2.21) que depende diretamente do *Duty Cycle*: quanto menor o *Duty Cycle*, menor é o valor médio do sinal, e vice-versa.

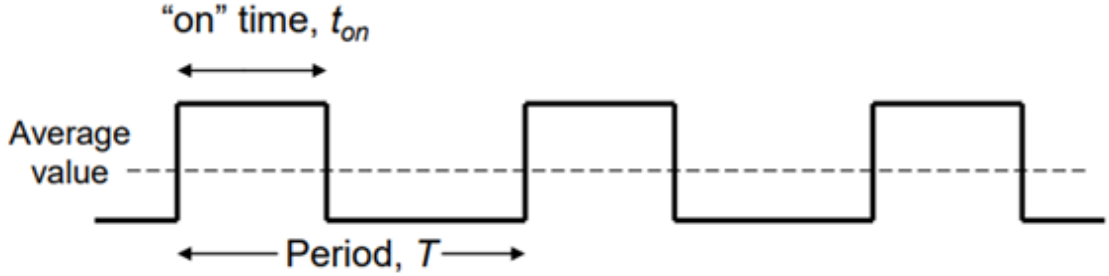

Figura 2.21: Valor médio de um sinal composto por pulsos [22].

Através de um filtro passa-baixo, o dispositivo recetor consegue extrair o valor médio do sinal, que é usado para decidir qual o valor analógico que este terá como *output*.

2.3.1.3 Controlador

O dispositivo central deste sistema é um Arduino Mega 2560 [23]. É um micro controlador composto por 54 pinos digitais de *input/output* (15 destas podem ser usadas como saídas PWM), 16 entradas analógicas e 4 portas série.

O software usado para programar este micro controlador é o Arduino Software (IDE), comum para qualquer placa de Arduino, usando C/C++ como linguagens de programação.

Figura 2.22: Arduino Mega 2560 [23].

2.3.1.4 Sensores

Cata-vento:

O cata-vento é usado para calcular a direção do vento. Como se pode observar na figura 2.23, a estrutura possui uma parte rotativa (o cata-vento propriamente dito) e a sua orientação varia consoante a direção do vento. Por baixo da peça rotativa, encontra-se um sensor de Hall (AS5161 [24]), que analisa a variação do campo magnético. Por outras palavras, o cata-vento induz um campo magnético que é analisado por este sensor e convertido para um valor de leitura que será posteriormente enviado para o Arduino (por PWM). Assim, é obtido um valor diferente para cada orientação do cata-vento.

Figura 2.23: Estrutura do cata-vento.

GPS:

O modelo MediaTek MT3329 [25] foi o sensor GPS escolhido para lidar com a localização do veleiro. Este possui uma antena que capta os sinais de um conjunto de satélites e faz a triangulação destes, para calcular a localização do dispositivo, em coordenadas geográficas (latitude e longitude).

Figura 2.24: Dispositivo de GPS.

Bússola:

A bússola fornece informação acerca da orientação da embarcação em relação ao norte magnético da Terra. O sensor usado neste caso foi o CMPS11 [26], que comunica com Arduino por I2C e fornece dados relativos à rotação nos 3 eixos, rpy (*roll*, *pitch and yaw*). Tendo em conta que também possui um acelerómetro, também se pode obter informação relativa à aceleração em x, y e z.

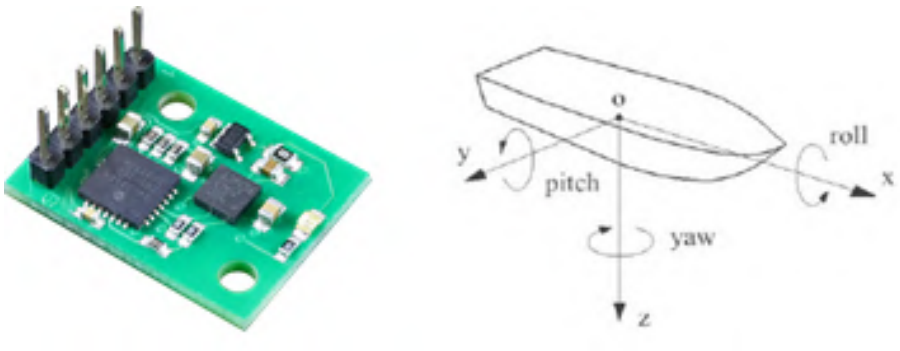

Figura 2.25: CMPS11 [26] e rpy [27].

2.3.2 Projeto NOVALANTIA

O projeto associado ao veleiro desta dissertação tem o nome de NOVALANTIA, e é composto por vários ficheiros, dos quais os principais (e os alteráveis) são:

- **WRSC_NOVALANTIA.ino** - ficheiro principal do projeto;
- **Bowline_math.cpp** - contém as funções associadas aos cálculos de bolina;
- **ConstSailboat.h** - *header* file que contém as constantes utilizadas no projeto;
- **Data_acquisition.cpp** - ficheiro que contém as funções associadas à leitura dos sensores;
- **remote.cpp** - contém as funções associadas ao controlo remoto.

Com o objetivo de facilitar a compreensão do conteúdo deste projeto, foram explicadas as principais ideias e algum código relativo ao ficheiros acima mencionados.

2.3.2.1 WRSC_NOVALANTIA.ino

Este ficheiro é composto maioritariamente pelas funções características do Arduino: *setup()* (que é executada apenas uma vez pelo sistema, antes do *loop()*) e *loop()* (executada ciclicamente pelo sistema).

Setup:
Na secção I.1 do anexo I encontra-se a representação desta função, onde são explicados os 5 principais blocos de código e as suas funções.

Loop:
Esta função funciona como um ciclo: executa as instruções que contém e, após terminar todas as instruções, volta a repetir este processo vezes sem conta, enquanto a placa se mantiver ligada.
A secção I.2 do anexo I contém a explicação da implementação desta função.

2.3.2.2 Data_acquisition.cpp

Como foi referido anteriormente, este ficheiro contém funções que estão associadas aos dados recolhidos pelos sensores (entre outras), que são explicadas no anexo II.

2.3.2.3 ConstSailboat.h

Este ficheiro contém apenas constantes que são usadas por funções de outros ficheiros do projeto, como por exemplo:

- DEBUG_SERIAL_BAUDRATE - *baudrate* da comunicação série do Arduino;
- SAIL_MIN_PULSE_WIDTH - valor mínimo do servo das velas;
- RUDDER_MAX_PULSE_WIDTH - valor máximo do servo do leme;
- RUDDER_SERVO_PIN - pino de conexão do servo do leme ao Arduino;
- NUMBER_BUOYS - número de bóias (*regatta*) ou *goals* (*Area Scanning*);
- etc.

2.3.2.4 Bowline_math.cpp

As funções deste ficheiro estão relacionadas com as áreas e as retas de navegação (figura 2.26), definida em dissertações anteriores [28]. De uma forma geral, para ir do ponto A (posição atual do veleiro) para o ponto B (*goal*) o veleiro deve navegar nas áreas 1 e 2 definidas na figura, evitando as áreas 3, 4, 5 e 6. O sistema deteta que já chegou ao destino quando o veleiro atravessa a baliza definida pelos pontos B1 e B2.

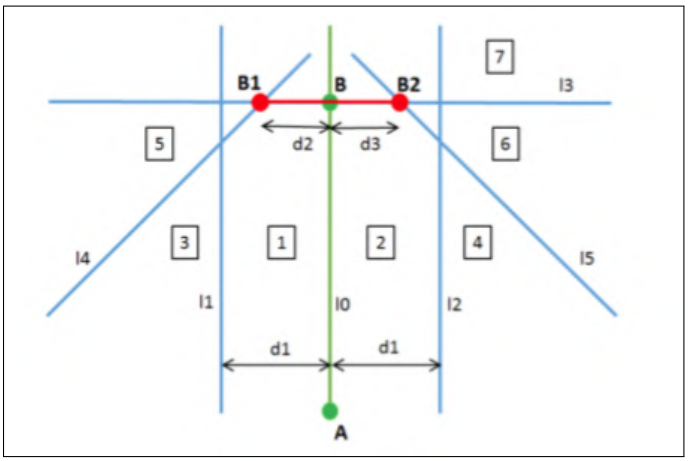

Figura 2.26: Esquema das áreas de navegação [28].

Desta forma, o ficheiro Bowline_math.cpp tem como principais funções:

- ***arrival_destination()*** - detecta se o veleiro atravessou a "baliza" do *goal* atual. Recebe como parâmetros de entrada a latitude e longitude atuais e retorna o valor lógico "*true*" caso isto se verifique. Caso contrário retorna "*false*";

- ***navigation_edges()*** - função que calcula as equações das retas que definem as áreas de navegação;
- ***navigation_area()*** - função que verifica em que zona de navegação se encontra o veleiro, associada à navegação à bolina;
- ***fill_struct()*** - recebe como parâmetros de entrada informações acerca da localização do *goal*, da localização atual, da direção do veleiro, etc. Dentro desta função é feito o mapeamento destas informações com as variáveis deste ficheiro. No final é executada a função *navigation_edges()*.

2.3.2.5 remote.cpp

Este ficheiro trata do envio e da receção de informações entre o rádio e o Arduino. Contém essencialmente as três funções seguintes:

- ***radioFeedbackFullInfo()*** - recebe os dados dos sensores, coordenadas do *goal* e localização atual (entre outros), e envia uma *string* composta por todos estes parâmetros para o rádio. Esta função é usada essencialmente para *debug*, pois torna-se possível observar os dados recolhidos em tempo real, sem ser necessário ter o Arduino conectado ao dispositivo de visualização (desde que esse dispositivo possua o recetor de rádio corretamente conectado a uma das portas série);
- ***sendPlannedTrack()*** - esta função é semelhante à anterior, mas neste caso é enviado o vetor que contém as coordenadas dos *goals* (*route_race*), juntamente com o vetor *cone_parameters*;
- ***remoteCommands()*** - nesta função são processados os comandos provenientes do controlo remoto, que são posteriormente processados, atualizando as variáveis que afetam os servos das velas e do leme.

Capítulo 3

Solução proposta

Inicialmente a solução para a realização da prova *Area Scanning* passaria por adotar uma abordagem mais simples e direta. No entanto, posteriormente foram surgindo novas ideias e possibilidades para um varrimento mais complexo e versátil.

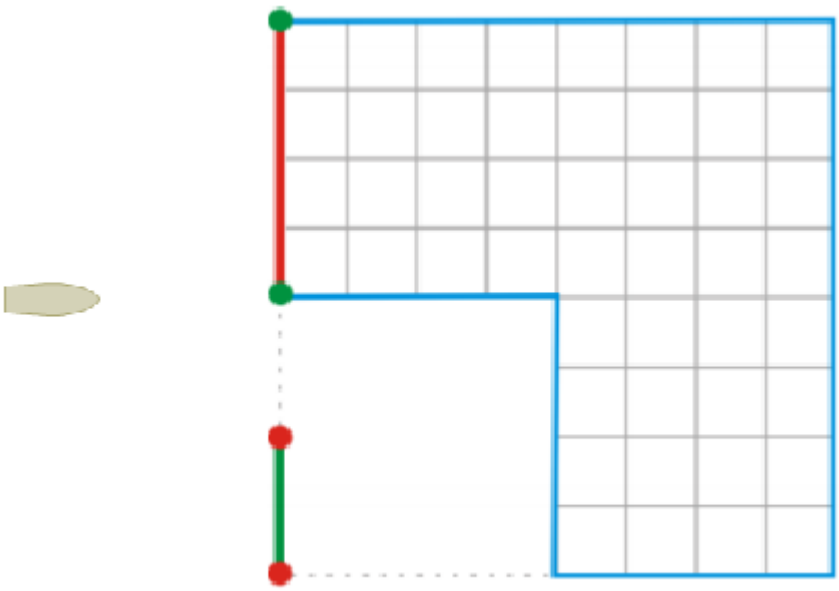

Figura 3.1: Cenário inicial [1].

De forma a que o veleiro tenha uma maior adaptabilidade às condições atmosféricas (nomeadamente o vento), foi proposta uma abordagem de varrimento (aproximadamente) perpendicular à direção do vento.

Para tal, é necessário calcular primeiramente o ângulo de incidência do vento sobre a

proa do barco (a orientação do veleiro na posição inicial necessita de ser aproximadamente perpendicular à linha de partida, como representado na figura 3.1).

3.1 Interpretação da direção do vento

Para uma melhor compreensão dos conceitos de incidência do vento ”perpendicular” e ”paralela”, são descriminadas na figura 3.2 as áreas do circulo trigonométrico (equivalente à vista superior do cata-vento) correspondentes a estes conceitos.

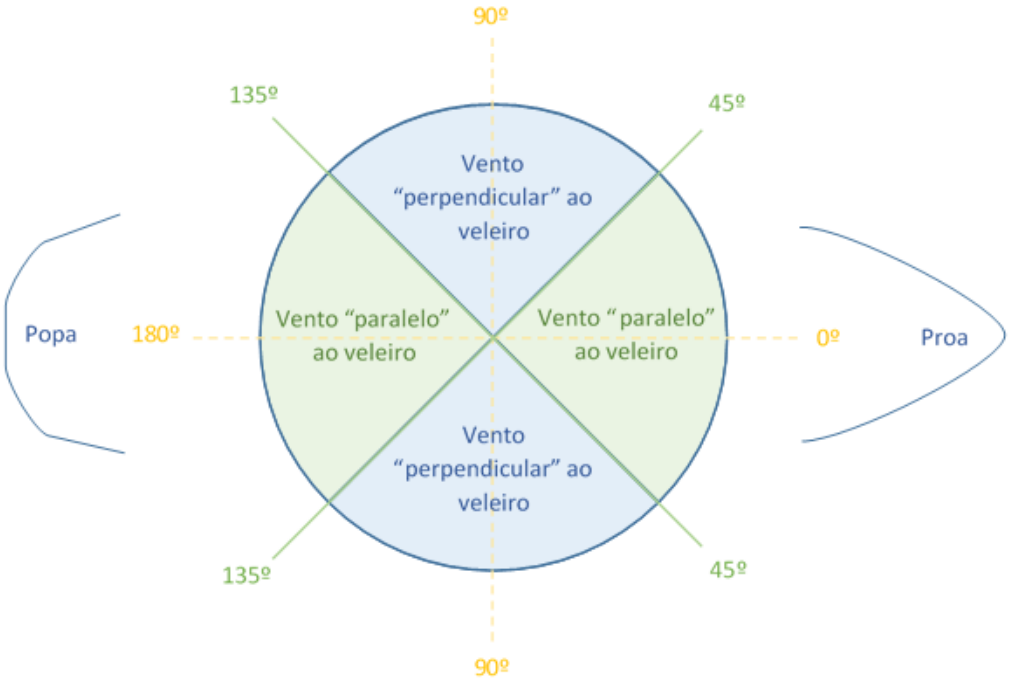

Figura 3.2: Representação dos ângulos de incidência do vento em relação à orientação do veleiro.

Nota: O circulo trigonométrico acima representado é ”espelhado” sobre o seu eixo horizontal, ou seja, o hemisfério superior e inferior são ambos limitados de 0º a 180º (ao invés de 0º a 180º no superior e 180º a 360º no inferior). Esta representação foi necessária de forma a ficar coerente com o valor lido do cata-vento pelo sistema, que será explicado na secção 4 - Implementação.

De uma forma resumida, considera-se que o vento incide paralelamente sobre o veleiro quando o ângulo formado entre a sua direção e a orientação do barco está compreendido entre 0º e 45º, ou entre 135º e 180º. Por outro lado, se este ângulo estiver compreendido entre 45º e 135º, considera-se que o vento incide perpendicularmente sobre a embarcação.

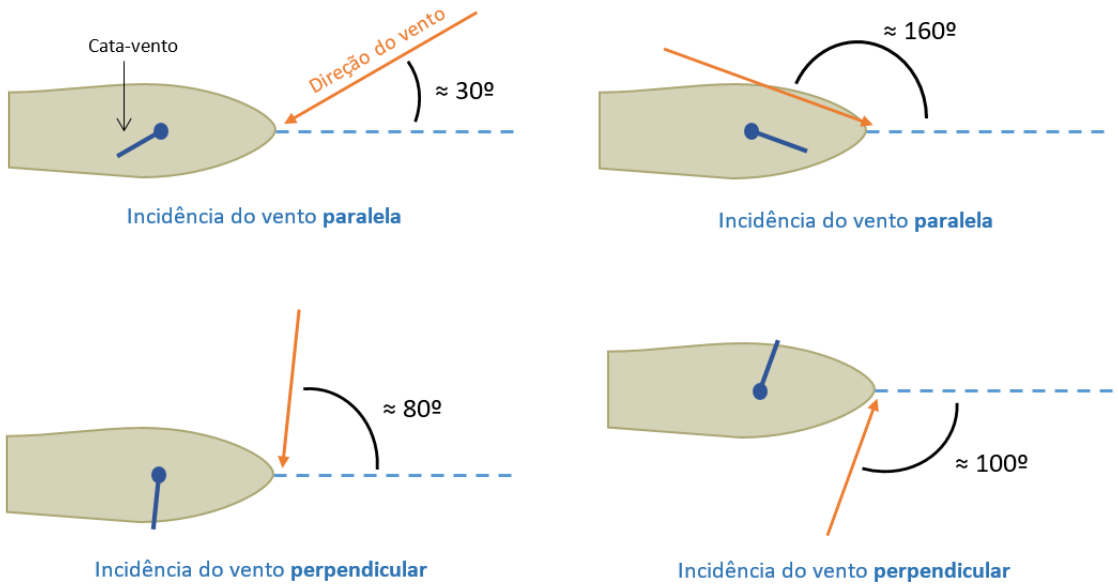

Figura 3.3: Exemplos de ângulos de incidência do vento sobre o veleiro.

3.2 Decisão da estratégia de navegação

Após a leitura e interpretação do ângulo de incidência do vento em relação ao barco, o sistema decide que estratégia de navegação irá adotar, que pode ser de dois tipos: horizontal ou vertical.

Esta decisão tem por base a necessidade de evitar navegar contra o vento. Tendo em conta que o vento é o único meio de ”propulsão” deste veleiro, nestas condições a navegação é bastante instável e pode tornar-se até impossível (mesmo navegando à bolina), impedindo a embarcação de completar o trajeto desejado. Assim, a estratégia passa por navegar de forma a que o veleiro tenha (quase) sempre um vento perpendicular ao seu eixo.

3.2.1 Navegação Horizontal

Se a incidência do vento for ”perpendicular” (ou seja, com um ângulo entre 45º e 135º) aquando a sua leitura (cenário da figura 3.1) a navegação será **horizontal**.

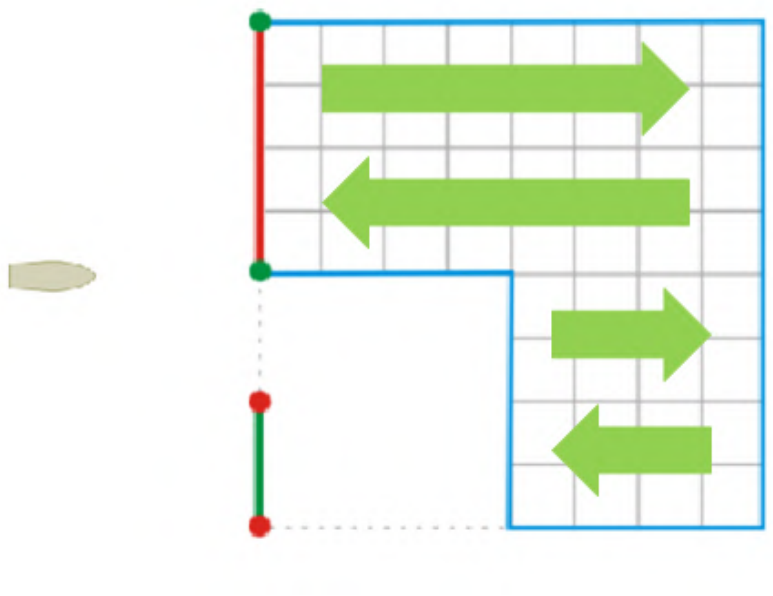

Figura 3.4: Navegação Horizontal.

Entende-se por ”navegar horizontalmente” como varrer a área de prova alternadamente para a frente e para trás, através de um trajeto aproximadamente perpendicular à linha de partida, como mostra a figura 3.4.

3.2.2 Navegação Vertical

Se, por outro lado, a direção do vento for paralela em relação ao eixo do veleiro (ângulo de incidência entre 0º e 45º ou 135º e 180º) a navegação será **vertical**.

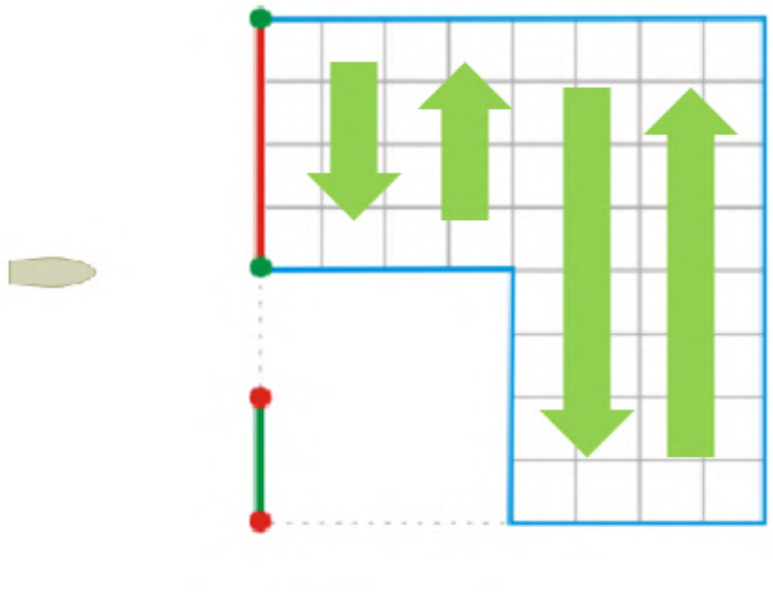

Figura 3.5: Navegação Vertical.

Entende-se por navegar ”navegar verticalmente” como varrer a área de prova alternadamente para a frente e para trás, através de um trajeto aproximadamente paralelo à linha de partida, como mostra a figura 3.5.

Na tabela 3.1 são resumidos os conceitos acima explicados, de acordo com os valores lidos pelo cata-vento e a interpretação do sistema.

Tabela 3.1: Decisões de Navegação consoante os ângulos lidos

Ângulo de incidência	**Interpretação**	**Decisão de Navegação**
0º a 45º	Vento ”paralelo”	Vertical
45º a 135º	Vento ”perpendicular”	Horizontal
135º a 180º	Vento ”paralelo”	Vertical

3.3 Trajetória e *Goals*

É necessário entender os conceitos de trajetória e de *goals* no contexto do problema, para compreender a solução proposta e a sua posterior implementação.

Um *goal* é um ponto geográfico, representado por coordenadas (latitude, longitude), que é interpretado pelo veleiro como um ”objetivo” para o qual ele tem de se deslocar.

A trajetória é o percurso que o barco tem de realizar para chegar ao *goal*, desde o ponto inicial. De notar que este percurso é o ideal (ou seja, o mais curto), sendo portanto uma mera previsão e não o trajeto exato que o veleiro irá realizar.

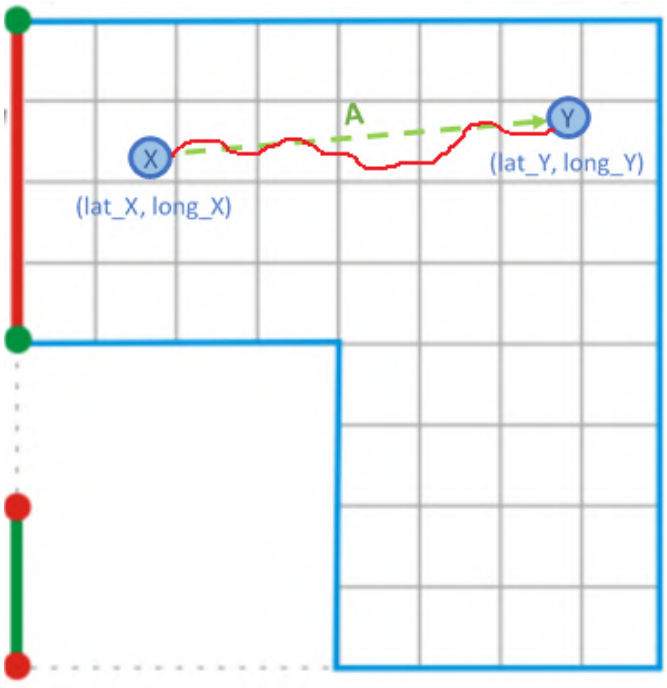

Figura 3.6: *Exemplo de um goal e uma trajetória.*

Na figura 3.6 está representado um cenário em que o veleiro se encontra no ponto X, com coordenadas lat_X e long_X, e tem como *goal* o ponto Y (lat_Y, long_Y). A linha a tracejado verde (A) representa a trajetória ideal entre os dois pontos, enquanto que a linha a vermelho representa o percurso real efetuado pela embarcação para chegar ao ponto Y.

Desta forma, para o veleiro completar a prova, é atribuído um conjunto de *goals* para que este percorra sequencialmente até chegar ao *goal* final (meta), como se poderá observar nas secções seguintes.

3.4 Profundidades de varrimento

Além da decisão da melhor estratégia de navegação (Horizontal ou Vertical) a abordar consoante as condições externas (vento), foram desenvolvidas várias camadas de profundidade de varrimento da área da prova para cada estratégia.

Desta forma, em cada estratégia de navegação existem 4 profundidades de varrimento: *Fast Scanning*, *Middle Scanning*, *Deep Scanning* e *Full Scanning*.

3.4.1 *Fast Scanning* (Varrimento Rápido)

Este é o varrimento mais rápido e básico. O sistema calcula uma trajetória curta e direta, mas pouco vantajosa em termos de pontuação, pois atravessa poucas quadriculas. É considerado o método de varrimento mais seguro, no que ao cumprimento das regras da competição diz respeito (atravessar a linha de partida e depois a de chegada em menos de 30 minutos), tendo em conta que possui uma previsão de duração de prova baixa e uma trajetória simples (com um baixo número de curvas), estando assim menos susceptível a cometer erros comparativamente a varrimentos mais longos ou mais complexos. Assim sendo, é o mais indicado quando as condições atmosféricas são adversas, como forte vento ou a sua inexistência, chuva e fortes correntes.

Vantagens:

- Susceptibilidade a erros: baixa;
- Duração de prova prevista: baixa;

Desvantagens:

- Pontuação prevista: baixa;

3.4.1.1 *Horizontal Fast Scanning*

Na figura abaixo é possível observar o método *Fast Scanning* aplicado à navegação Horizontal, constituído pelo *goal* inicial (0), um *goal* intermédio (1) e o *goal* final (2). É de todos os métodos o mais rápido e aquele que apresenta o menor trajeto desde a linha de partida até à linha de meta.

Nota: A conjugação de uma determinada profundidade de varrimento com uma determinada estratégia de navegação, origina aquilo que se definiu de tipo/método de varrimento. Por exemplo: uma profundidade *fast* conjugada com a estratégia horizontal, dá origem ao tipo/método de varrimento *Horizontal Fast Scanning*.

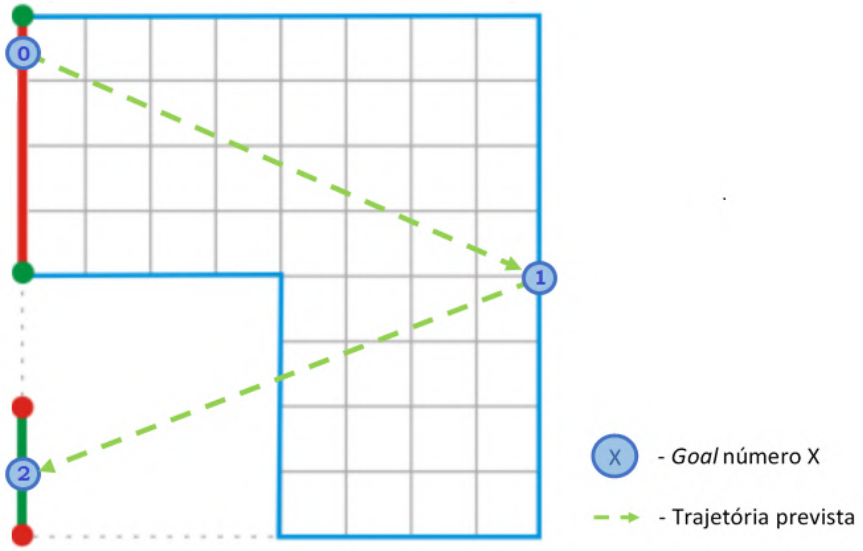

Figura 3.7: *Horizontal Fast Scanning*.

Previsões:

- Duração de prova (máximo 30 minutos): **5 a 10 minutos** (referência: 160 metros = 1 a 6 minutos);
- Pontuação (máximo 48 pontos): **16 pontos** (1 quadrícula = 1 ponto);
- Distância a percorrer (soma da distância aproximada dos troços da trajetória): **340 metros**;

3.4.1.2 *Vertical Fast Scanning*

O método *Vertical Fast Scanning* é o método mais rápido e curto da navegação Vertical. Além do *goal* inicial e do final, possui dois *goals* intermédios (1 e 2), tendo por isso um trajeto ligeiramente mais longo que o homólogo da navegação Horizontal.

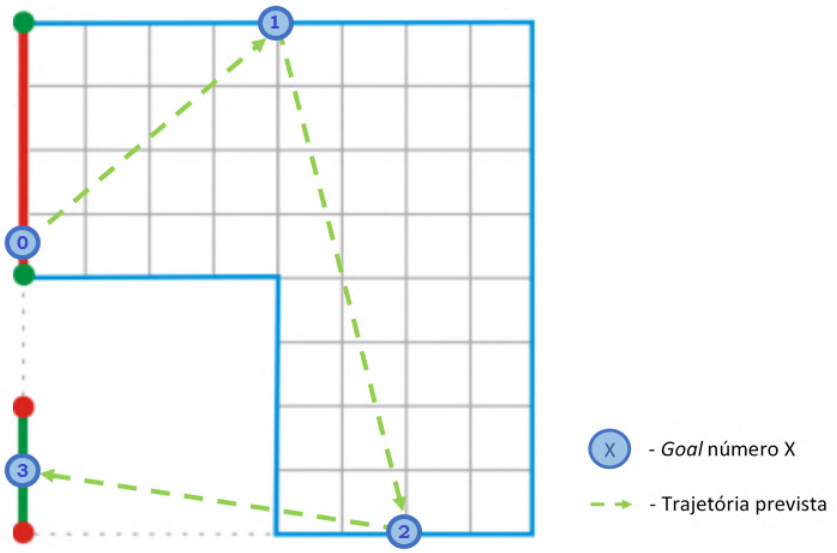

Figura 3.8: *Vertical Fast Scanning*.

Previsões:

- Duração de prova: **7 a 12 minutos**;
- Pontuação: **17 pontos**;
- Distância a percorrer: **420 metros**;

3.4.2 *Middle Scanning* (Varrimento Médio)

Como o próprio nome indica, esta é a profundidade de varrimento mais equilibrada. Permite obter uma melhor pontuação em relação ao *Fast Scanning*, mas não tão boa comparativamente ao *Deep Scanning* (explicado mais à frente). Em termos de distância de trajeto e duração de prova, é ligeiramente superior ao *Fast Scanning*.

Vantagens:

- Susceptibilidade a erros: média;
- Duração de prova prevista: média;

Desvantagens:

- Pontuação prevista: média;

3.4.2.1 *Horizontal Middle Scanning*

Como se pode observar em baixo, este método apresenta uma melhoria significativa em termos de pontuação, em relação ao *Horizontal Fast Scanning* (14 pontos de diferença),

sendo no entanto esse incremento compensado negativamente com uma maior duração e distância do trajeto a percorrer.

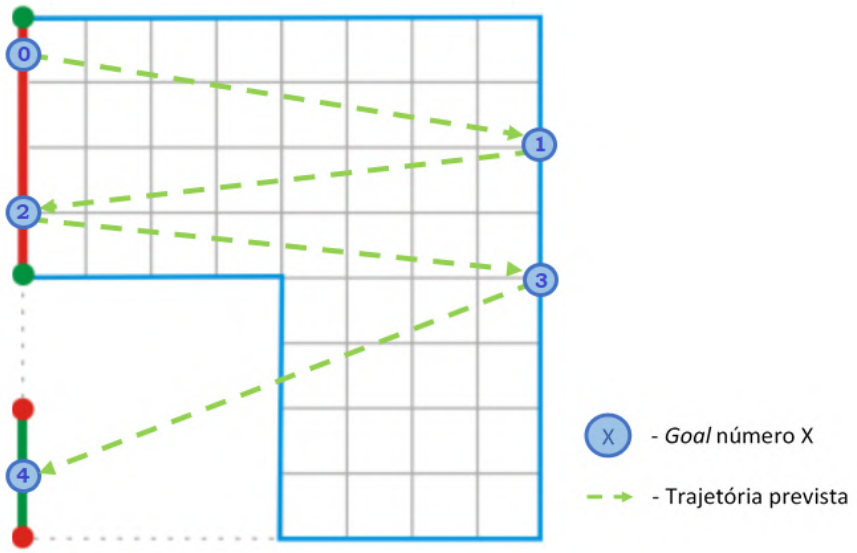

Figura 3.9: *Horizontal Middle Scanning*.

Previsões:

- Duração de prova: **12 a 17 minutos**;
- Pontuação: **30 pontos**;
- Distância a percorrer: **650 metros**;

3.4.2.2 *Vertical Middle Scanning*

Já o *Vertical Middle Scanning* não apresenta uma melhoria tão significativa de pontuação em relação ao *Vertical Fast Scanning*, comparativamente ao método equivalente da navegação Horizontal. Isto deve-se ao facto de o troço acrescentado ao trajeto ser relativamente curto, cobrindo apenas uma área que incrementa a pontuação em 3 pontos.

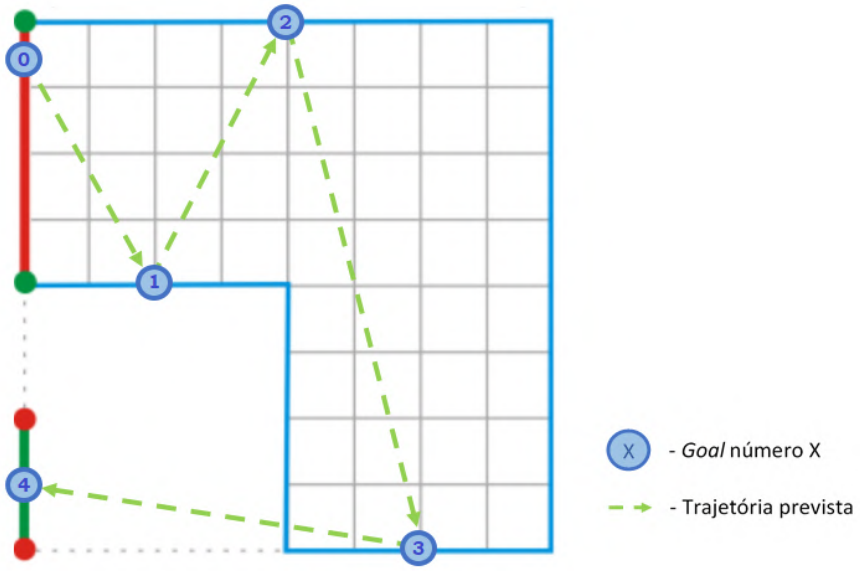

Figura 3.10: *Vertical Middle Scanning*.

Previsões:

- Duração de prova: **8 a 13 minutos**;
- Pontuação: **20 pontos**;
- Distância a percorrer: **450 metros**;

3.4.3 *Deep Scanning* (Varrimento Profundo)

Este é o método de varrimento profundo. Permite obter uma pontuação bastante elevada, conseguindo cobrir quase toda a área da prova. No entanto, é mais extenso no que toca à duração de prova e apresenta uma maior complexidade de trajeto, sendo assim mais propício a erros. Posto isto, é ideal para situações em que as condições externas são favoráveis (vento moderado e constante, correntes fracas, etc.).

Vantagens:

- Pontuação prevista: alta;

Desvantagens:

- Susceptibilidade a erros: alta;
- Duração de prova prevista: alta;

3.4.3.1 *Horizontal Deep Scanning*

Apresenta um incremento de 7 pontos em relação ao *Horizontal Middle Scanning*, percorrendo 88% da área total da prova. Por outro lado, a distância do seu trajeto aumenta em cerca de 165 metros, assim como a sua complexidade e por consequência o tempo despendido.

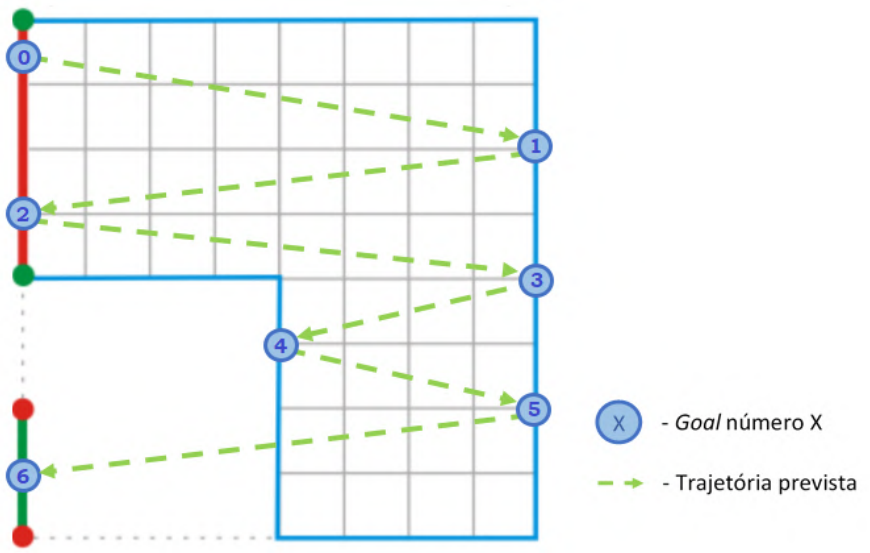

Figura 3.11: *Horizontal Deep Scanning*.

Previsões:

- Duração de prova: **15 e 20 minutos**;
- Pontuação: **37 pontos**;
- Distância a percorrer: **815 metros**;

3.4.3.2 *Vertical Deep Scanning*

No que toca à pontuação, este método apresenta um incremento bastante significativo em relação ao *Vertical Middle Scanning* (14 pontos de diferença), devido à adição de mais um troço que percorre ambas as extremidades da área da prova, originando um aumento de cerca 29% da área da prova. Esta diferença de previsões entre ambos os métodos é também refletida na duração da prova e na distância do trajeto (que aumentou para quase o dobro).

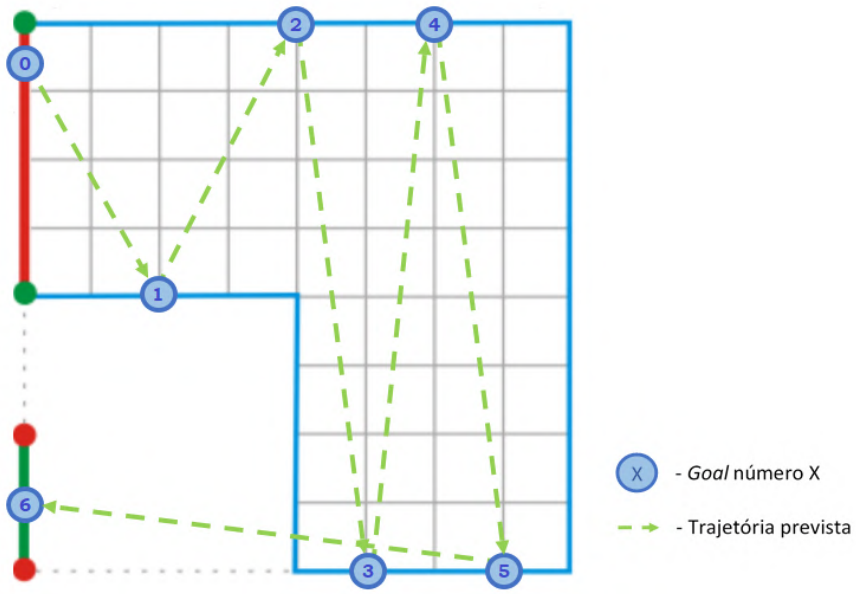

Figura 3.12: *Vertical Deep Scanning*.

Previsões:

- Duração de prova: **15 a 20 minutos**;
- Pontuação: **34 pontos**;
- Distância a percorrer: **800 metros**;

3.4.4 *Full Scanning* (Varrimento Completo)

O objetivo deste método é cobrir completamente a área da prova, ou seja, as 48 quadrículas. Apesar de "perfeito" no que toca à pontuação, possui um trajeto extenso e uma duração bastante longa. Assim sendo, é aconselhável apenas em situações em que se verifiquem condições de navegação praticamente perfeitas, pois o risco de o veleiro não conseguir efetuar a prova dentro do tempo limite é elevado.

Vantagens:

- Pontuação prevista: máxima;

Desvantagens:

- Susceptibilidade a erros: muito alta;
- Duração de prova prevista: muito alta;

3.4.4.1 *Horizontal Full Scanning*

Sendo o método da navegação horizontal com maior pontuação prevista, é também de todos os métodos aquele que apresenta maior número de *goals* e distância a percorrer, tornando-se assim o que apresenta maior complexidade espacial e temporal.

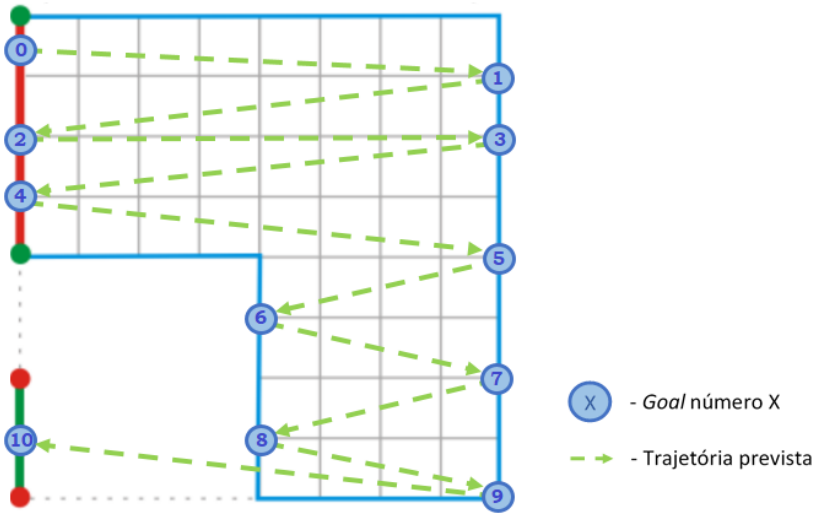

Figura 3.13: *Horizontal Full Scanning*.

Previsões:

- Duração de prova: **25 a 30 minutos**;
- Pontuação: **48 pontos**;
- Distância a percorrer: **1280 metros**;

3.4.4.2 *Vertical Full Scanning*

Apresenta uma maior eficiência no que toca à distância a percorrer em comparação com o *Horizontal Full Scanning* (cerca de 110 metros de diferença). É, no entanto, também bastante extenso e complexo, tendo por isso associado um elevado risco de ocorrência de erros.

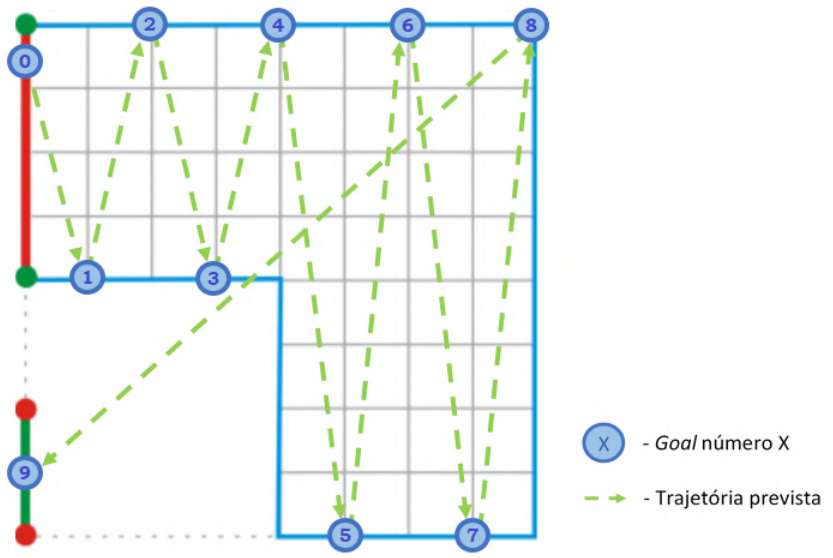

Figura 3.14: *Vertical Full Scanning*.

Previsões:

- Duração de prova: **23 a 28 minutos**;
- Pontuação: **48 pontos**;
- Distância a percorrer: **1170 metros**;

3.4.5 Comparação de características

Nas tabelas 3.2 e 3.3 encontram-se resumidas as características previstas de cada método, quer para a navegação Horizontal quer para a Vertical.

Tabela 3.2: Resumo dos métodos de varrimento da navegação Horizontal

	Pontuação	**Duração (minutos)**	**Distância (metros)**
Fast	16	5 a 10	340
Middle	30	12 a 17	650
Deep	37	15 a 20	815
Full	48	25 a 30	1280

Tabela 3.3: Resumo dos métodos de varrimento da navegação Vertical

	Pontuação	**Duração (minutos)**	**Distância (metros)**
Fast	17	7 a 12	420
Middle	20	8 a 13	450
Deep	34	15 a 20	800
Full	48	23 a 28	1170

Os valores das tabelas acima foram convertidos para os gráficos 3.15, 3.16 e 3.17, onde é possível comparar as características entre os diferentes tipos de varrimento e entre as ambas as estratégias de navegação.

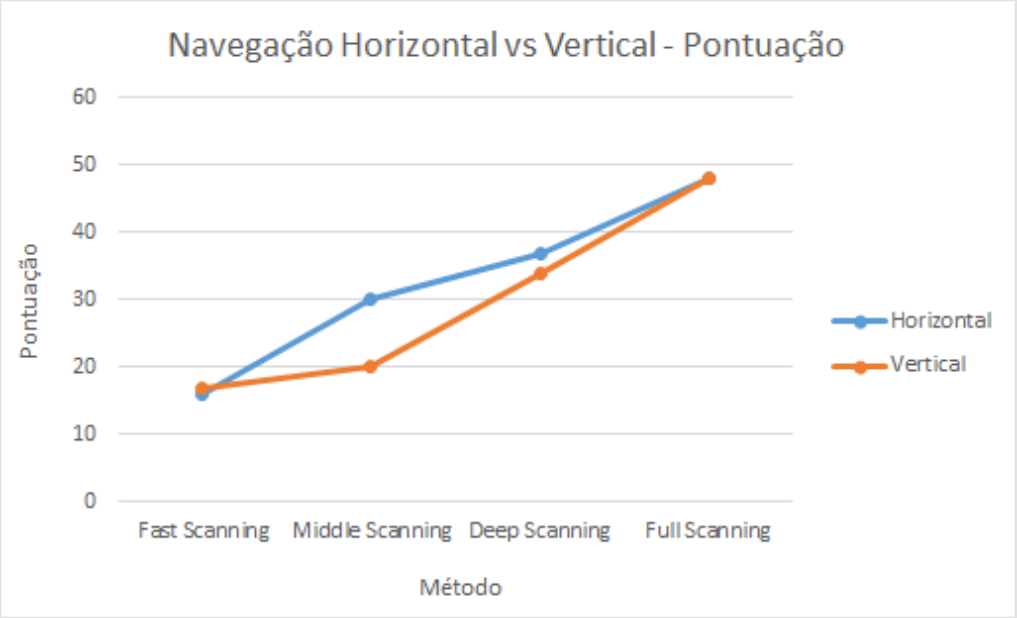

Figura 3.15: Comparação da previsão de pontuação.

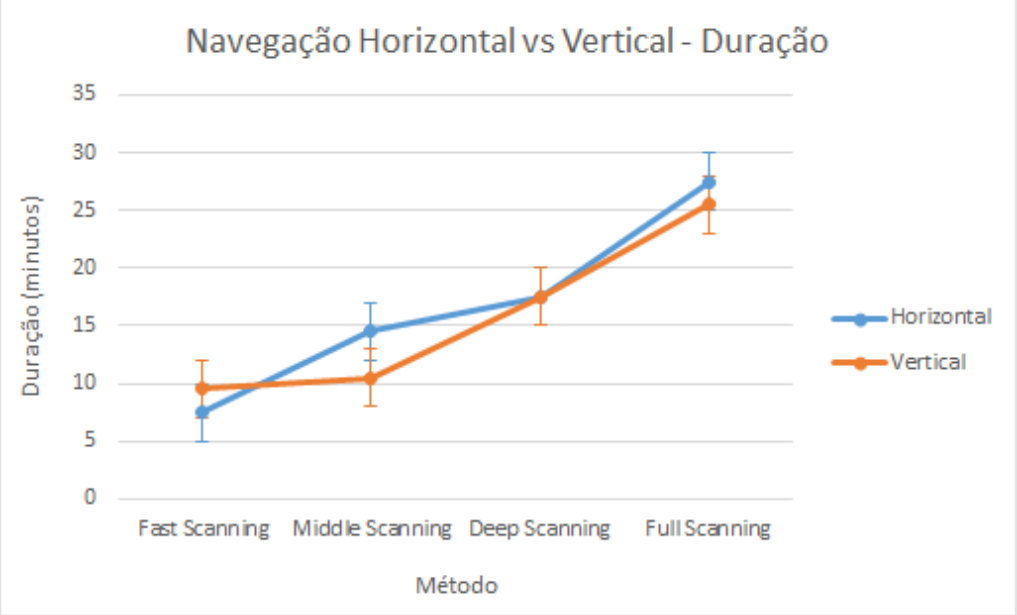

Figura 3.16: Comparação da previsão da duração de prova.

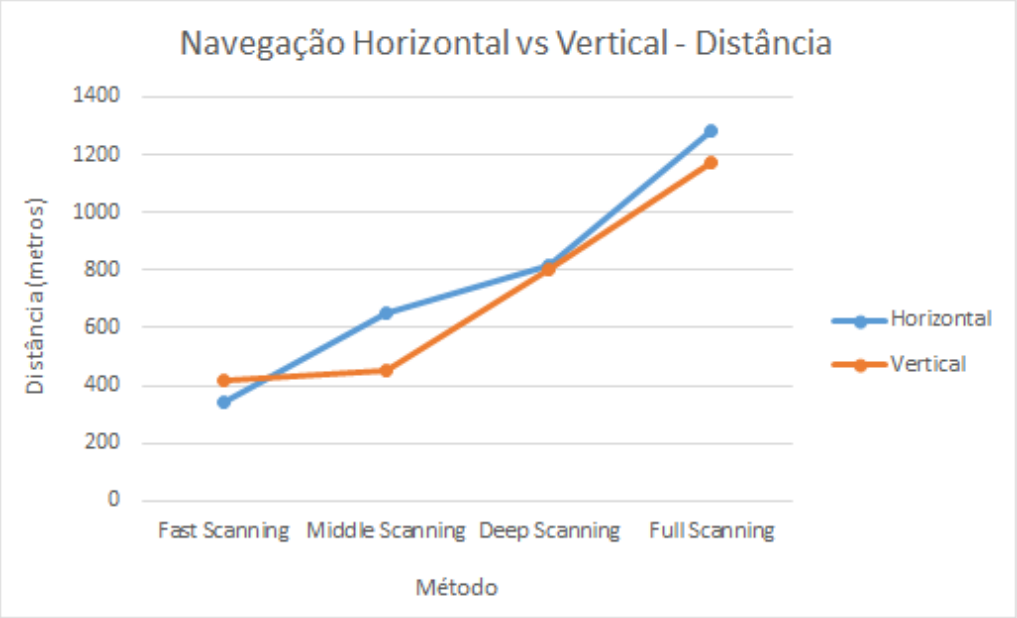

Figura 3.17: Comparação da previsão da distância a percorrer.

3.4.6 *Smart Scanning* (Varrimento Inteligente)

Como foi visto no capítulo anterior, o sistema tem à sua disposição um leque de vários tipos de varrimento (4 por cada estratégia de navegação), e portanto, é necessário escolher qual se irá aplicar.

Inicialmente, a ideia passou por tornar o sistema capaz de decidir qual o tipo de varrimento mais adequado às condições externas (ex: se se verificarem ventos muito fortes, a decisão passaria pelo *Horizontal/Vertical Fast Scanning*), através da medição da velocidade de deslocação do veleiro e/ou da velocidade do vento.

No entanto, os sensores utilizados pelo veleiro desta dissertação não reúnem as condições necessárias para obter valores credíveis em relação às características mencionadas anteriormente (o cata-vento apenas lê a direção do vento, não a velocidade do mesmo; o sensor GPS possui um parâmetro "velocidade de deslocação", mas os valores associados a este são totalmente irrealistas, e portanto não aplicáveis).

Posto isto, decidiu-se adotar um mecanismo "*Smart Scanning*". É atribuído ao sistema um método de varrimento predefinido (transversal a ambas as estratégias de navegação), sendo esse método aquele que será aplicado inicialmente na prova.

No caso de o método predefinido ser o *Fast Scanning* ou o *Middle Scanning*, este é implementado até ao fim da prova, ou seja, os *goals* são calculados inicialmente e mantém-se até o veleiro finalizar (ou não) a prova.

Caso seja o *Deep Scanning* ou o *Full Scanning*, a abordagem é diferente. Inicialmente os *goals* associados a estes métodos são gerados e o trajeto fica definido. No entanto, 2 destes *goals* gerados são especiais, pois também representam *checkpoints*. Os *checkpoints* são representações de "metas virtuais", os quais o veleiro tem de atravessar dentro de

um tempo limite (*deadline*). Caso o veleiro não consiga atravessar estes *checkpoints* no intervalo de tempo estabelecido, o sistema deve recalcular os *goals* de forma a estabelecer uma rota mais curta e direta em relação à meta.

Esta abordagem tem como objetivo permitir a utilização dos métodos de varrimento mais eficientes relativos à pontuação (*Deep Scanning e Full Scanning*) na maior parte dos cenários e de uma forma mais segura. Desta maneira, mesmo com condições atmosféricas adversas e com a possibilidade de ocorrência de erros ou dificuldade no cumprimento do trajeto, este é recalculado de forma a permitir ao veleiro cumprir pelo menos os requisitos mínimos da prova.

Nas figuras abaixo são ilustrados os *checkpoints* associados à abordagem *Smart Scanning* para os dois tipos de varrimento mais complexos das navegações horizontal e vertical:

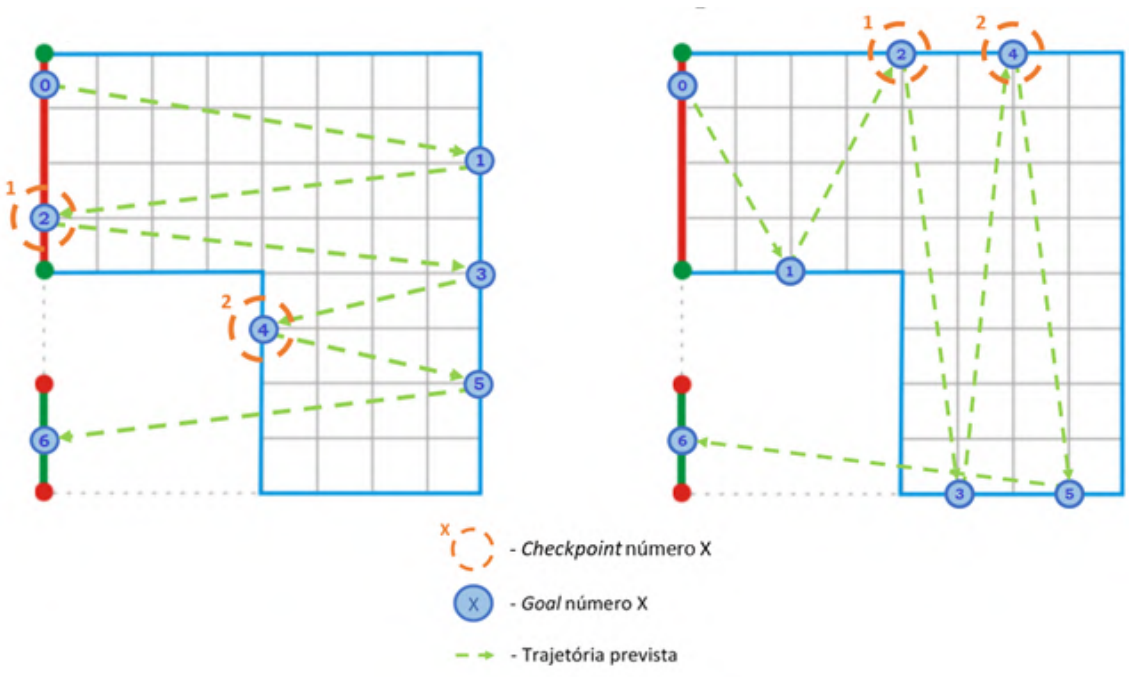

Figura 3.18: *Checkpoints* do *Smart Horizontal Deep Scanning* e *Smart Vertical Deep Scanning*.

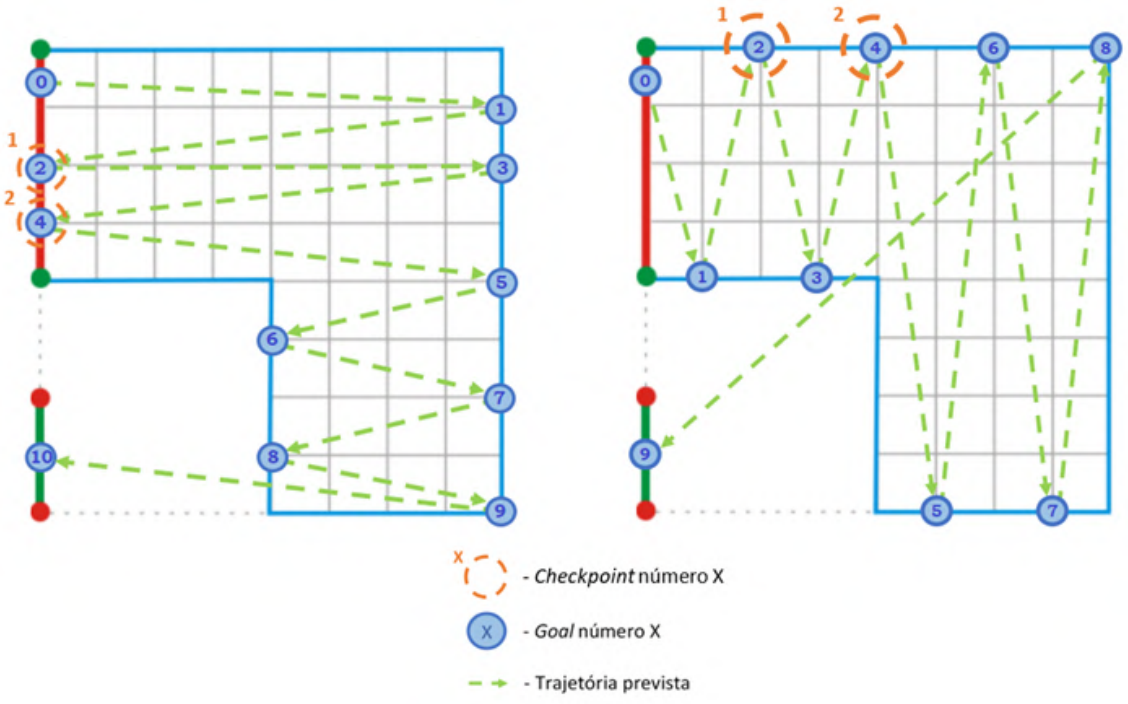

Figura 3.19: *Checkpoints* do *Smart Horizontal Full Scanning* e *Smart Vertical Full Scanning*.

De notar que, tanto para o varrimento profundo como para o varrimento total, os *checkpoints* são associados aos *goals* 2 e 4. A razão por detrás desta escolha prende-se com o facto de a fase "crítica" da prova ser o período inicial:

- Se as condições externas forem adversas, o veleiro terá dificuldade em navegar logo na fase inicial da prova, sendo portanto ideal recalcular um trajeto mais curto logo neste período de forma a não perder tempo a tentar atingir *goals* desnecessários;
- Caso as condições climatéricas sejam favoráveis e o veleiro conseguir atravessar os *checkpoints* dentro do tempo estabelecido, a probabilidade de concluir o varrimento é muito grande, pois à partida as condições de navegação serão idênticas até esta parte.

A colocação destes *checkpoints* nos *goals* 2 e 4 é, portanto, um mecanismo para o sistema perceber se conseguirá realizar a prova com o varrimento predefinido ou se necessita de adotar um varrimento mais curto e direto.

Para clarificar a utilização do *Smart Scanning*, é explicado um exemplo prático:

Na figura 3.20 encontra-se um cenário de uma prova de varrimento, na qual o método predefinido é o *Horizontal Deep Scanning*. Depois de iniciar a prova, o valeiro passou o *goal* 0 e percorrer a rota calculada inicialmente.

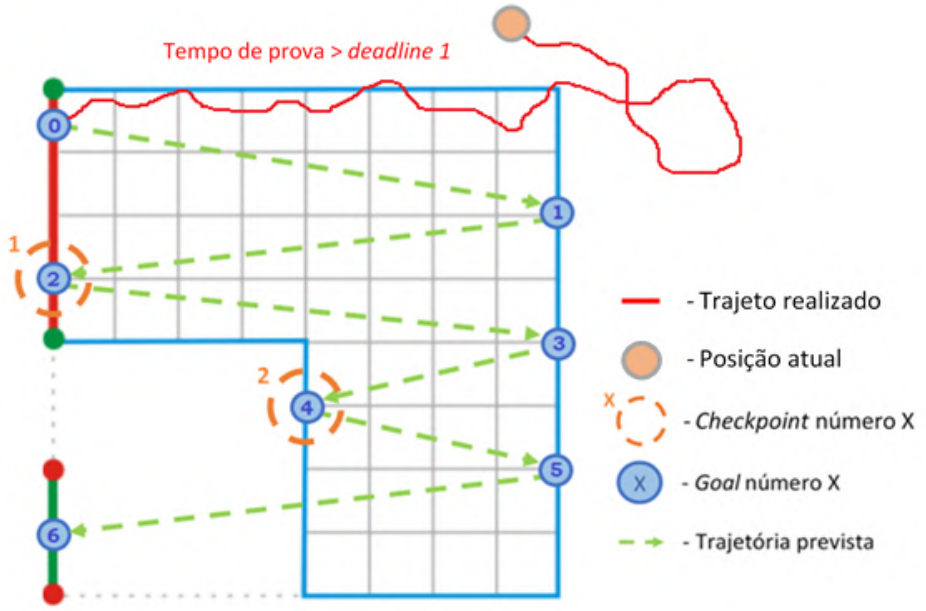

Figura 3.20: Exemplo 1 de utilização do *Smart Horizontal Deep Scanning*.

No entanto, devido a dificuldades de navegação, o tempo de prova excedeu a *deadline 1* sem que o veleiro tenha conseguido atravessar o *checkpoint 1*. Posto isto, a rota terá de ser recalculada, para evitar a desqualificação. Para tal (no caso de o método predefinido ser o Horizontal/Vertical Deep Scanning), o sistema assume o *goal* 5 como "próximo destino", ignorando todos os anteriores (figura 3.21). Para o caso do Horizontal/Vertical Full Scanning, o sistema assume o *goal* 6 como "próximo destino".

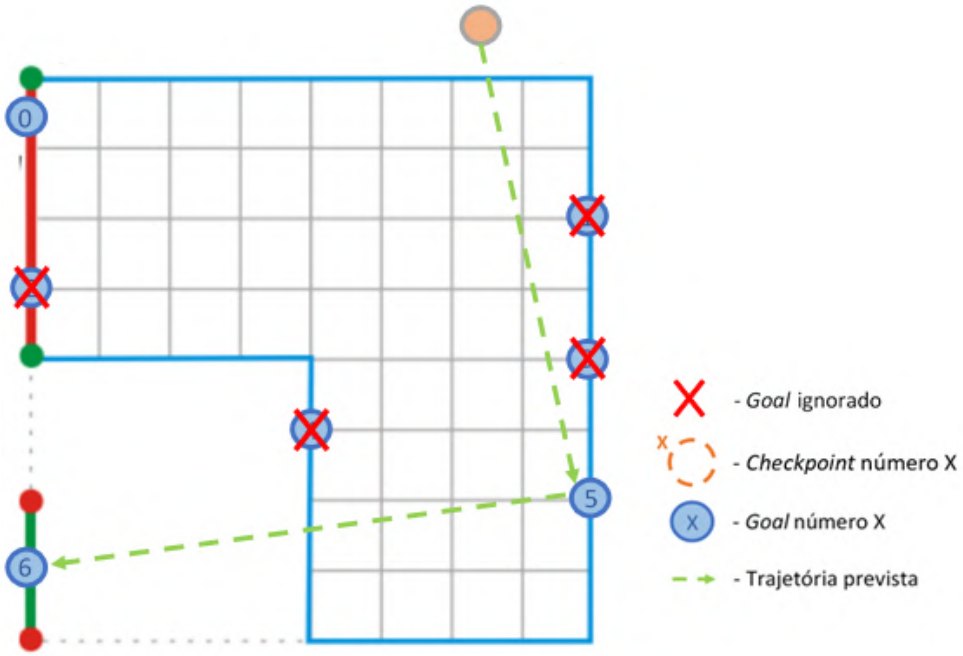

Figura 3.21: Exemplo 2 de utilização do *Smart Horizontal Deep Scanning*.

3.5 Diagrama da solução

Resumindo, ao iniciar a prova, o sistema avalia que tipo de navegação (Horizontal ou Vertical) irá adotar, consoante o valor da direção do vento. De seguida, é decidido qual o tipo de varrimento, dentro da estratégia de navegação adotada: um mecanismo predefinido normal, ou um mecanismo com a abordagem *Smart Scanning*. O diagrama abaixo ilustra de uma forma mais intuitiva e sintetizada todo o processo explicado anteriormente neste capítulo.

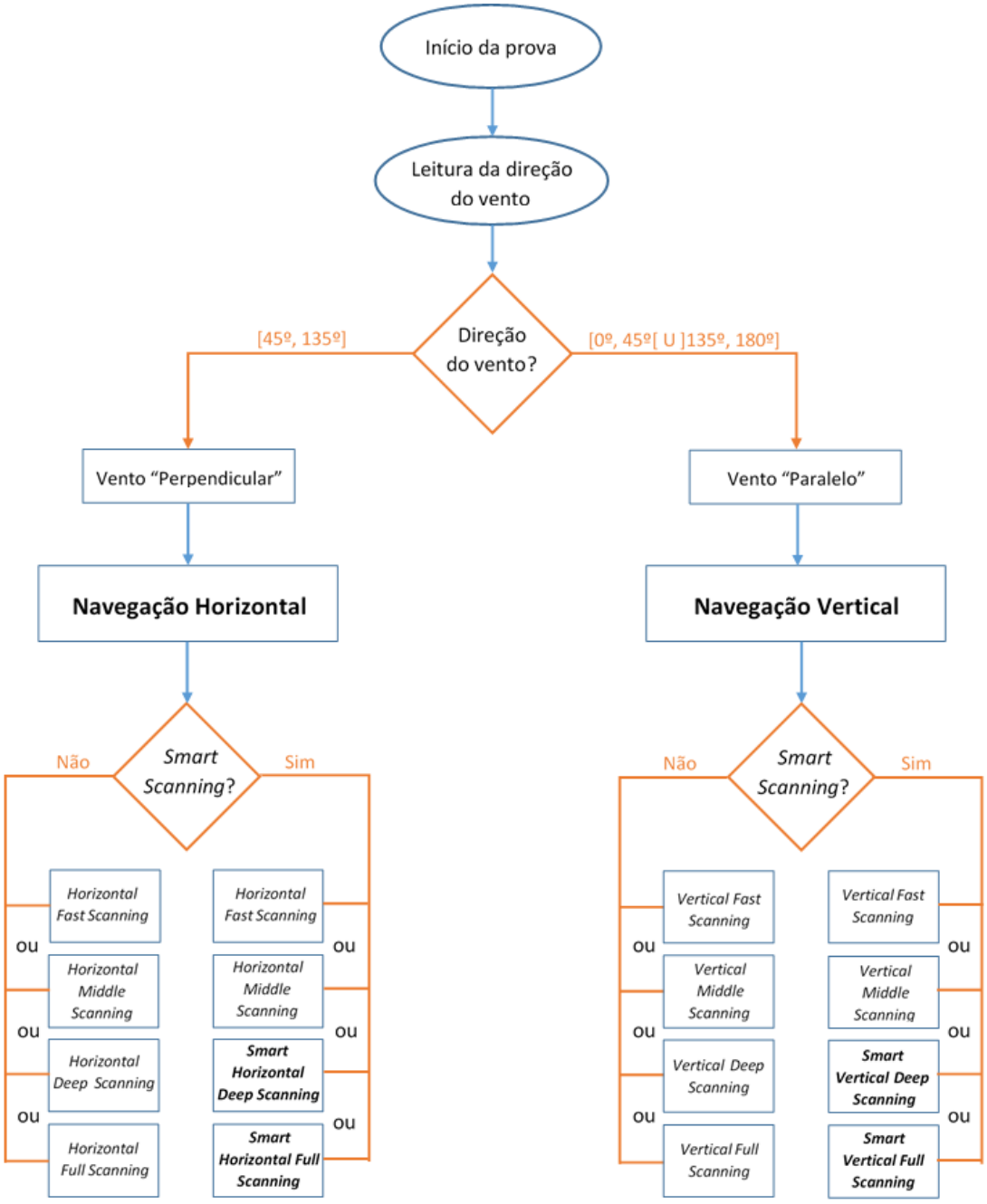

Figura 3.22: Diagrama do "*flow*" da solução a implementar.

Capítulo 4

Implementação

Após definida a abordagem e a solução para o problema, procedeu-se à implementação e construção da mesma.

Olhando para o diagrama da figura 3.22 do capítulo anterior, começou-se por implementar a solução "a partir do fim", ou seja, pelos métodos/tipos de varrimento. Isto porque estes métodos são a definição direta da abordagem à prova, pelo que a ideia foi começar a implementação por esta parte e, progressivamente, ir aumentando o grau de complexidade da solução, acrescentando posteriormente o mecanismo *Smart Scanning* e as estratégias de navegação Horizontal e Vertical.

Para a implementação desta solução e de cada um dos seus módulos, foram usados como ferramentas o Visual Studio 2017 e o Arduino IDE.

4.1 Tipos de varrimento

Para modelar os tipos/métodos de varrimento, é primeiramente necessário conhecer melhor a área da prova, para perceber que tipo de informações se podem retirar relativamente a esta.

Como já foi mostrado anteriormente, a área da prova em questão assume a forma de uma "peça de tetris", dividida por quadrículas de 20x20 metros. Ora esta área é composta por 8 bóias, que representam os vértices da mesma (figura 4.1).

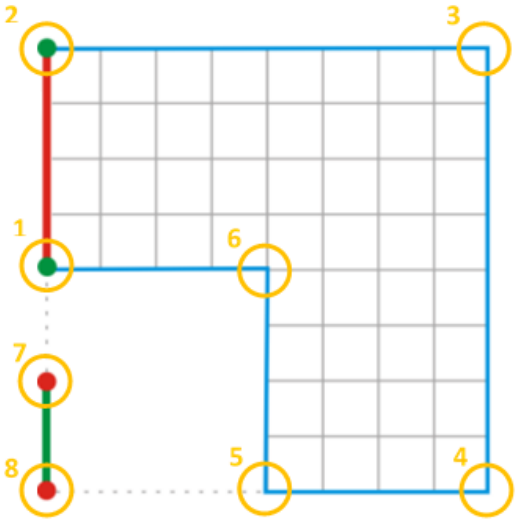

Figura 4.1: Representação das bóias na área da prova.

Como é visível na figura acima, as bóias foram numeradas (a representação das bóias na imagem não está à escala), de forma a distinguirem-se umas das outras. A cada uma das bóias está associada uma localização (latitude, longitude) e são a única informação, fornecida pela organização da prova, que o sistema recebe acerca da área, sendo portanto consideradas como *"input"* do sistema.

A partir desta informação, é possível definir outros conceitos importantes acerca da área. Como as retas exteriores que unem as bóias, ou seja, as retas que definem o perímetro desta "peça de tetris", juntamente com a linha de chegada (figura 4.2).

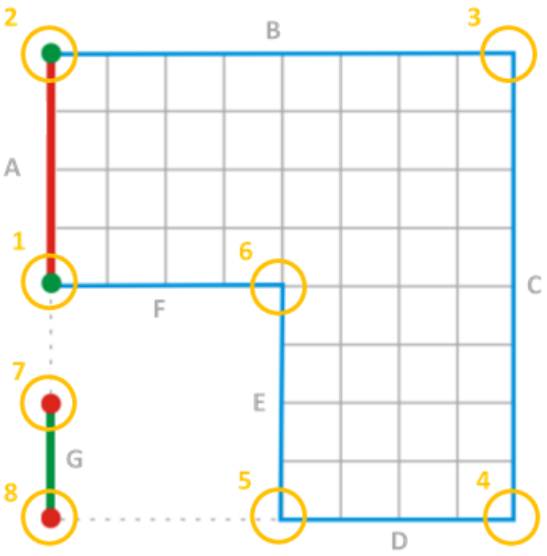

Figura 4.2: Representação das retas na área da prova.

A identificação destas retas é importante para entender o conceito que está por trás dos cálculos dos *goals* associados a cada tipo de varrimento, pois estes serão sempre pontos

contidos nestas retas, como se irá ver mais à frente.

Ao todo, existem 7 retas (A a G), que se relacionam com as bóias da seguinte forma:

- Reta A: liga as bóias 1 e 2;
- Reta B: liga as bóias 2 e 3;
- Reta C: liga as bóias 3 e 4;
- Reta D: liga as bóias 4 e 5;
- Reta E: liga as bóias 5 e 6;
- Reta F: liga as bóias 6 e 1;
- Reta G: liga as bóias 7 e 8;

Sendo que as bóias possuem cada uma um par de coordenadas GPS associado, que são fornecidos pela organização da competição, antes da prova:

- Bóia 1 = (buoy_1_lat, buoy_1_long);
- Bóia 2 = (buoy_2_lat, buoy_2_long);
- Bóia 3 = (buoy_3_lat, buoy_3_long);
- Bóia 4 = (buoy_4_lat, buoy_4_long);
- Bóia 5 = (buoy_5_lat, buoy_5_long);
- Bóia 6 = (buoy_6_lat, buoy_6_long);
- Bóia 7 = (buoy_7_lat, buoy_7_long);
- Bóia 8 = (buoy_8_lat, buoy_8_long);

Desta forma, é possível prosseguir para o cálculo dos *goals* que irão definir a trajetória do veleiro.

4.1.1 Representação geométrica dos *goals*

Cada *goal* gerado tem por trás um conceito matemático que assenta na determinação do **ponto médio entre 2 pontos**.

Imagine-se a situação da figura 4.3, onde existe um segmento de reta que une dois pontos, P_1 e P_2, com coordenadas (X_1, Y_1) e (X_2, Y_2), respetivamente.

Figura 4.3: Segmento de reta que une os pontos P1 e P2.

Para determinar o ponto médio entre os pontos P_1 e P_2, basta calcular a média de cada componente X e Y, ou seja, somar as coordenadas correspondentes (X_1+X_2 e Y_1+Y_2) e dividir por 2:

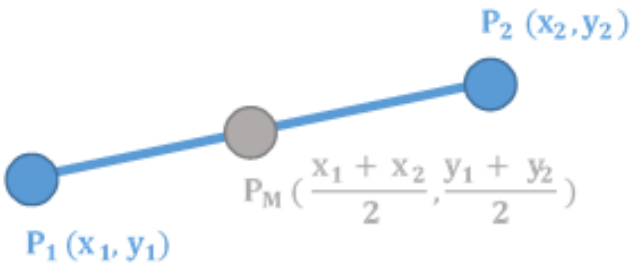

Figura 4.4: Ponto médio entre os pontos P1 e P2.

Todos os *goals* de cada tipo de varrimento são gerados através de cálculos simples ou cálculos em cadeia de pontos médios.

Para enquadrar de uma forma mais intuitiva este conceito no contexto do problema, tome-se como exemplo o cenário da figura 4.5. Pretende-se calcular as coordenadas do *goal* X - ponto médio do segmento de reta A, que une as bóias 1 (buoy_1_lat, buoy_1_long) e 2 (buoy_2_lat, buoy_2_long) - e o *goal* Y, que é o ponto médio do segmento de reta que une a bóia 1 e o *goal* X, ou seja, a bóia 1 e este *goal* representam os extremos de **1/4 da reta A**.

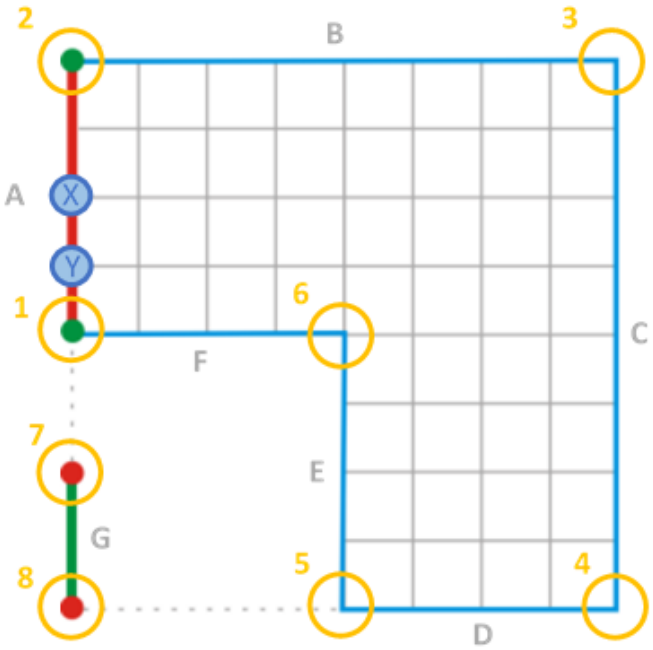

Figura 4.5: Exemplo de cálculo de um goal.

Para calcular o *goal X* procede-se da seguinte forma:

$$Goal\_X = (\frac{buoy\_1\_lat + buoy\_2\_lat}{2}, \frac{buoy\_1\_long + buoy\_2\_long}{2}) \tag{4.1}$$

E para o *goal* Y:

$$Goal\_Y = (\frac{buoy\_1\_lat + goal\_X\_lat}{2}, \frac{buoy\_1\_long + goal\_X\_long}{2}) \tag{4.2}$$

Sendo,

$$goal\_X\_lat = \frac{buoy\_1\_lat + buoy\_2\_lat}{2} \tag{4.3}$$

$$goal\_X\_long = \frac{buoy\_1\_long + buoy\_2\_long}{2} \tag{4.4}$$

Obtém-se assim as coordenadas geográficas dos *goals* X e Y, sendo este exemplo aplicável aos cálculos de cada um dos métodos de varrimento.

Os *goals* podem ser caracterizados consoante o seu **"nível de cálculo"** e o **segmento de reta que origina**. O "nível de cálculo" diz respeito ao número de cálculos de pontos médios que são necessários fazer para obter as coordenadas desejadas. No exemplo anterior, o *goal* X possuí **nível 1**, pois foi apenas necessário fazer o cálculo do ponto médio 1 vez. Já o *goal* Y possuí **nível 2**, pois foi preciso 2 cálculos de pontos médios para obter as coordenadas respetivas. Pode dizer-se que o "nível de cálculo" traduz-se na profundidade da cadeia de cálculos do ponto médio.

O *goal* calculado origina também uma "partição" da reta em que este está contido. No caso do *goal* X, "dividiu" a reta A ao meio, originando dois segmentos equivalentes a **1/2** desta mesma reta. No caso do *goal* Y, originou um segmento de reta equivalente a **1/4** (ou 3/4, mas opta-se sempre por referir o menor segmento) da reta A.

4.1.2 *Horizontal Fast Scanning*

Este método apresenta o seguinte trajeto:

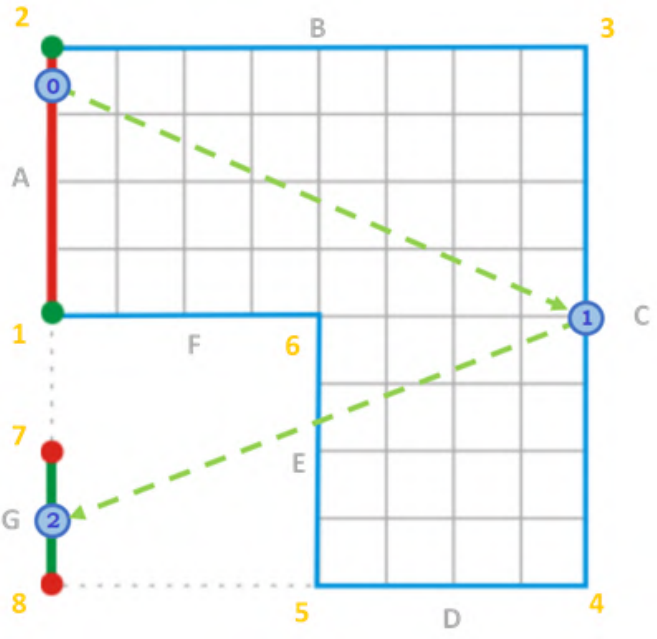

Figura 4.6: *Horizontal Fast Scanning - goals*.

Os *goals* associados a este trajeto podem ser caracterizados por (nível de cálculo; segmento de reta originado):

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 1; 1/2 da reta C;
- ***Goal 2*** - nível 1; 1/2 da reta G;

Sendo calculados desta forma:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2}+buoy\_2\_long}{2}) \tag{4.5}$$

$$Goal\_1 = (\frac{buoy\_3\_lat+buoy\_4\_lat}{2}, \frac{buoy\_3\_long+buoy\_4\_long}{2}) \tag{4.6}$$

$$Goal\_2 = (\frac{buoy\_7\_lat+buoy\_8\_lat}{2}, \frac{buoy\_7\_long+buoy\_8\_long}{2}) \tag{4.7}$$

4.1.3 *Vertical Fast Scanning*

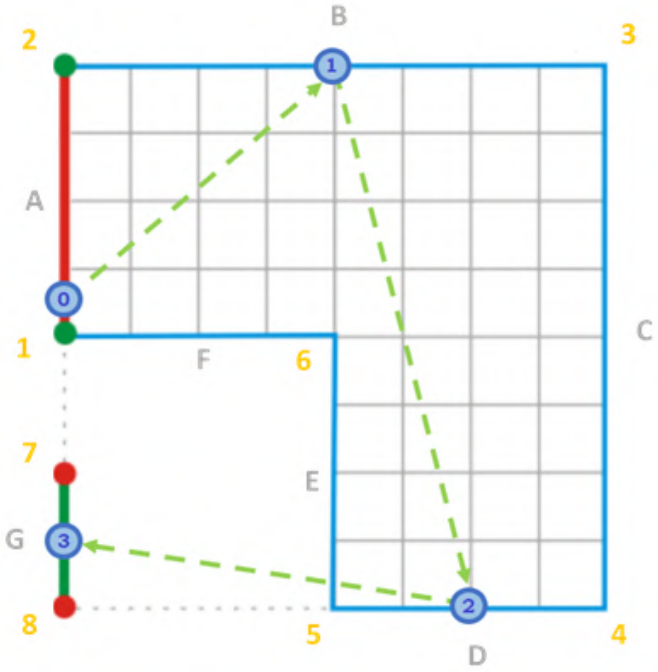

Figura 4.7: *Vertical Fast Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (inferior);
- ***Goal 1*** - nível 1; 1/2 da reta B;
- ***Goal 2*** - nível 1; 1/2 da reta D;
- ***Goal 3*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{buoy\_1\_lat + \frac{buoy\_1\_lat + \frac{buoy\_1\_lat + buoy\_2\_lat}{2}}{2}}{2}, \frac{buoy\_1\_long + \frac{buoy\_1\_long + \frac{buoy\_1\_long + buoy\_2\_long}{2}}{2}}{2}) \tag{4.8}$$

$$Goal\_1 = (\frac{buoy\_2\_lat + buoy\_3\_lat}{2}, \frac{buoy\_2\_long + buoy\_3\_long}{2}) \tag{4.9}$$

$$Goal\_2 = (\frac{buoy\_4\_lat + buoy\_5\_lat}{2}, \frac{buoy\_4\_long + buoy\_5\_long}{2}) \tag{4.10}$$

$$Goal\_3 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.11}$$

4.1.4 *Horizontal Middle Scanning*

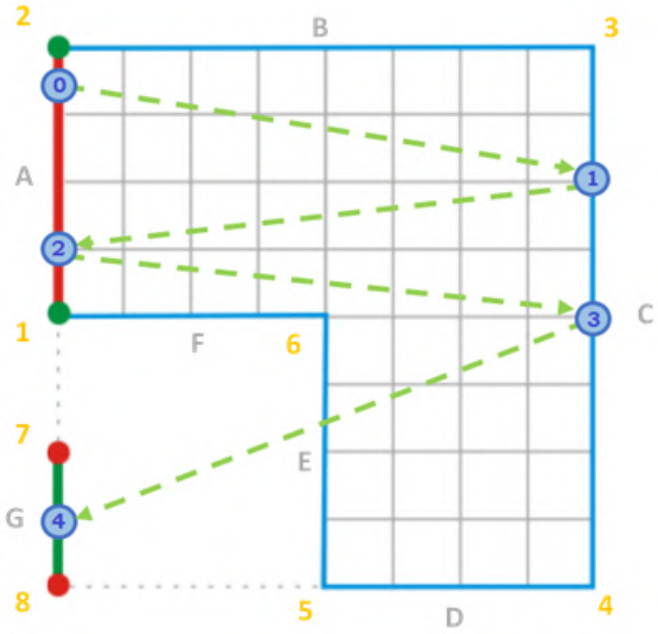

Figura 4.8: *Horizontal Middle Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 2; 1/4 da reta C (superior);
- ***Goal 2*** - nível 2; 1/4 da reta A (inferior);
- ***Goal 3*** - nível 1; 1/2 da reta C;
- ***Goal 4*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2}+buoy\_2\_long}{2}) \tag{4.12}$$

$$Goal\_1 = (\frac{buoy\_3\_lat+\frac{buoy\_3\_lat+buoy\_4\_lat}{2}}{2}, \frac{buoy\_3\_long+\frac{buoy\_3\_long+buoy\_4\_long}{2}}{2}) \tag{4.13}$$

$$Goal\_2 = (\frac{buoy\_1\_lat+\frac{buoy\_1\_lat+buoy\_2\_lat}{2}}{2}, \frac{buoy\_1\_long+\frac{buoy\_1\_long+buoy\_2\_long}{2}}{2}) \tag{4.14}$$

$$Goal\_3 = (\frac{buoy\_3\_lat + buoy\_4\_lat}{2}, \frac{buoy\_3\_long + buoy\_4\_long}{2}) \tag{4.15}$$

$$Goal\_4 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.16}$$

4.1.5 *Vertical Middle Scanning*

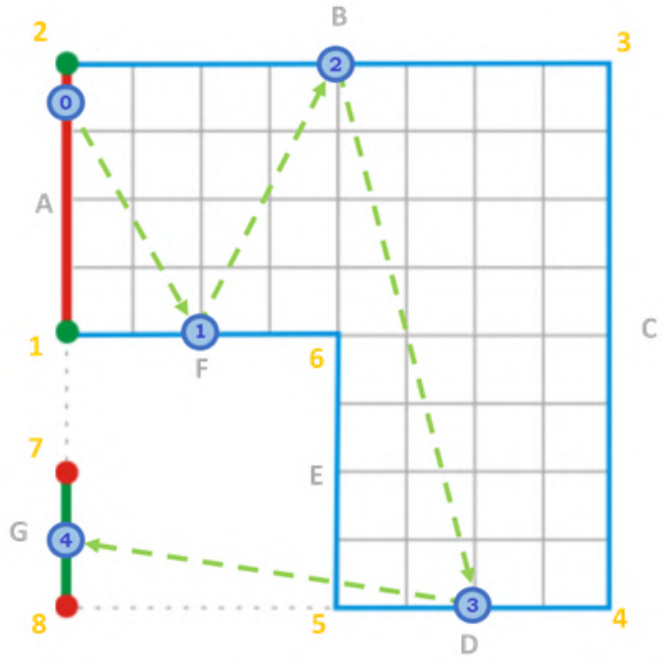

Figura 4.9: *Vertical Middle Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 1; 1/2 da reta F;
- ***Goal 2*** - nível 1; 1/2 da reta B;
- ***Goal 3*** - nível 1; 1/2 da reta D;
- ***Goal 4*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2} + buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2} + buoy\_2\_long}{2}) \tag{4.17}$$

$$Goal\_1 = (\frac{buoy\_6\_lat + buoy\_1\_lat}{2}, \frac{buoy\_6\_long + buoy\_1\_long}{2}) \tag{4.18}$$

$$Goal\_2 = (\frac{buoy\_2\_lat + buoy\_3\_lat}{2}, \frac{buoy\_2\_long + buoy\_3\_long}{2}) \tag{4.19}$$

$$Goal\_3 = (\frac{buoy\_4\_lat + buoy\_5\_lat}{2}, \frac{buoy\_4\_long + buoy\_5\_long}{2}) \tag{4.20}$$

$$Goal\_4 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.21}$$

4.1.6 *Horizontal Deep Scanning*

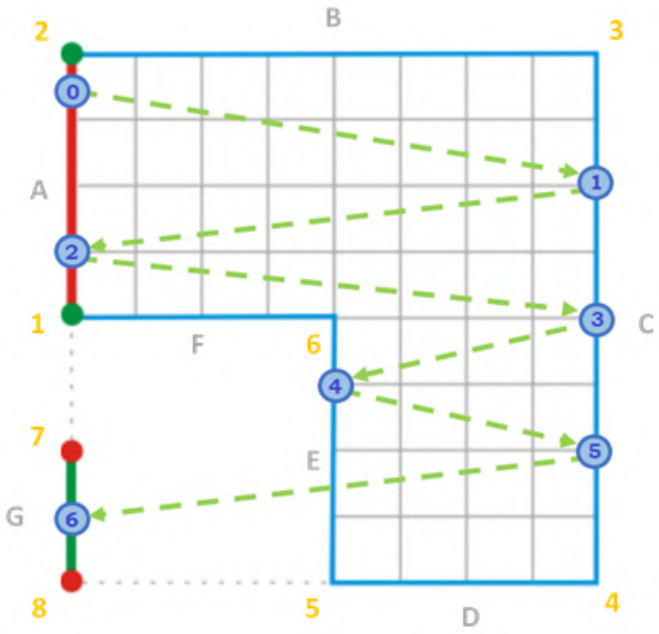

Figura 4.10: *Horizontal Deep Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 2; 1/4 da reta C (superior);
- ***Goal 2*** - nível 2; 1/4 da reta A (inferior);
- ***Goal 3*** - nível 1; 1/2 da reta C;
- ***Goal 4*** - nível 2; 1/4 da reta E (superior);
- ***Goal 5*** - nível 2; 1/4 da reta C (inferior);
- ***Goal 6*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2} + buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2} + buoy\_2\_long}{2}) \tag{4.22}$$

$$Goal\_1 = (\frac{buoy\_3\_lat + \frac{buoy\_3\_lat+buoy\_4\_lat}{2}}{2}, \frac{buoy\_3\_long + \frac{buoy\_3\_long+buoy\_4\_long}{2}}{2}) \tag{4.23}$$

$$Goal\_2 = (\frac{buoy\_1\_lat + \frac{buoy\_1\_lat+buoy\_2\_lat}{2}}{2}, \frac{buoy\_1\_long + \frac{buoy\_1\_long+buoy\_2\_long}{2}}{2}) \tag{4.24}$$

$$Goal\_3 = (\frac{buoy\_3\_lat + buoy\_4\_lat}{2}, \frac{buoy\_3\_long + buoy\_4\_long}{2}) \tag{4.25}$$

$$Goal\_4 = (\frac{\frac{buoy\_5\_lat+buoy\_6\_lat}{2} + buoy\_6\_lat}{2}, \frac{\frac{buoy\_5\_long+buoy\_6\_long}{2} + buoy\_6\_long}{2}) \tag{4.26}$$

$$Goal\_5 = (\frac{\frac{buoy\_3\_lat+buoy\_4\_lat}{2} + buoy\_4\_lat}{2}, \frac{\frac{buoy\_3\_long+buoy\_4\_long}{2} + buoy\_4\_long}{2}) \tag{4.27}$$

$$Goal\_6 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.28}$$

4.1.7 *Vertical Deep Scanning*

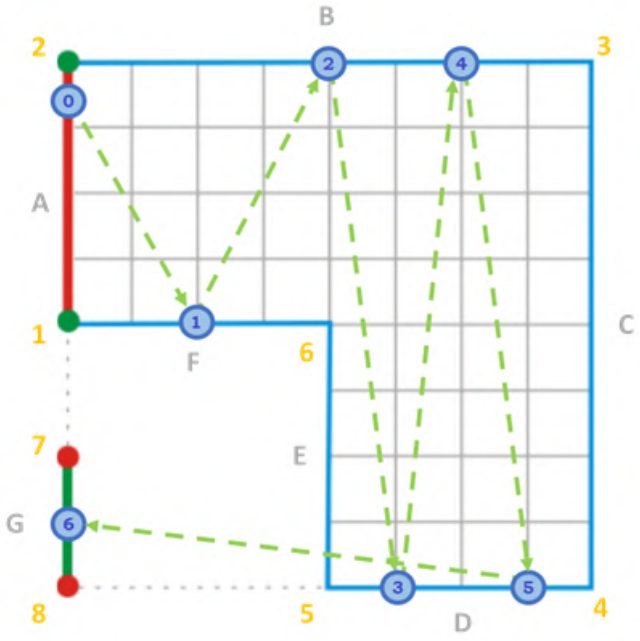

Figura 4.11: *Vertical Deep Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 1; 1/2 da reta F;
- ***Goal 2*** - nível 1; 1/2 da reta B;
- ***Goal 3*** - nível 2; 1/4 da reta D (esquerda);
- ***Goal 4*** - nível 2; 1/4 da reta B (direita);
- ***Goal 5*** - nível 2; 1/4 da reta D (direita);
- ***Goal 6*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}+buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2}+buoy\_2\_long}{2}) \tag{4.29}$$

$$Goal\_1 = (\frac{buoy\_6\_lat+buoy\_1\_lat}{2}, \frac{buoy\_6\_long+buoy\_1\_long}{2}) \tag{4.30}$$

$$Goal\_2 = (\frac{buoy\_2\_lat+buoy\_3\_lat}{2}, \frac{buoy\_2\_long+buoy\_3\_long}{2}) \tag{4.31}$$

$$Goal\_3 = (\frac{\frac{buoy\_4\_lat+buoy\_5\_lat}{2} + buoy\_5\_lat}{2}, \frac{\frac{buoy\_4\_long+buoy\_5\_long}{2} + buoy\_5\_long}{2}) \tag{4.32}$$

$$Goal\_4 = (\frac{\frac{buoy\_2\_lat+buoy\_3\_lat}{2} + buoy\_3\_lat}{2}, \frac{\frac{buoy\_2\_long+buoy\_3\_long}{2} + buoy\_3\_long}{2}) \tag{4.33}$$

$$Goal\_5 = (\frac{buoy\_4\_lat + \frac{buoy\_4\_lat+buoy\_5\_lat}{2}}{2}, \frac{buoy\_4\_long + \frac{buoy\_4\_long+buoy\_5\_long}{2}}{2}) \tag{4.34}$$

$$Goal\_6 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.35}$$

4.1.8 *Horizontal Full Scanning*

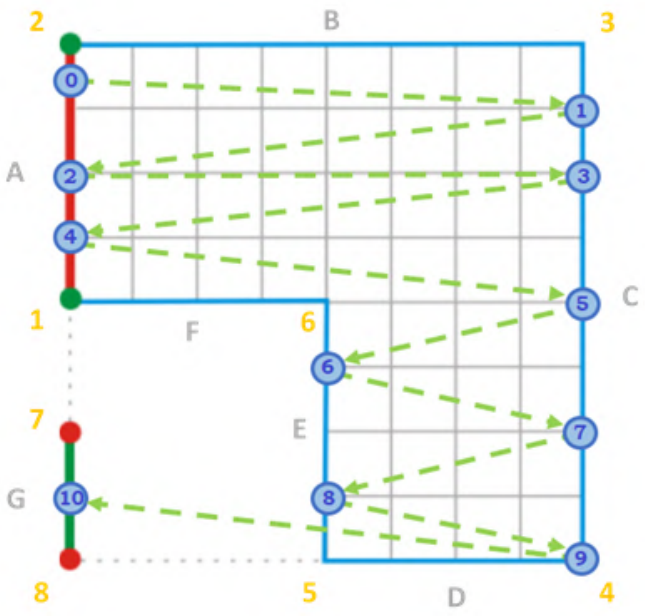

Figura 4.12: *Horizontal Full Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 3; 1/8 da reta C (superior);
- ***Goal 2*** - nível 1; 1/2 da reta A;
- ***Goal 3*** - nível 2; 1/4 da reta C (superior);

- ***Goal 4*** - nível 2; 1/4 da reta A (inferior);
- ***Goal 5*** - nível 1; 1/2 da reta C;
- ***Goal 6*** - nível 2; 1/4 da reta E (superior);
- ***Goal 7*** - nível 2; 1/4 da reta C (inferior);
- ***Goal 8*** - nível 2; 1/4 da reta E (inferior);
- ***Goal 9*** - nível 0; bóia 4;
- ***Goal 10*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2} + buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2} + buoy\_2\_long}{2}) \tag{4.36}$$

$$Goal\_1 = (\frac{buoy\_3\_lat + \frac{buoy\_3\_lat+\frac{buoy\_3\_lat+buoy\_4\_lat}{2}}{2}}{2}, \frac{buoy\_3\_long + \frac{buoy\_3\_long+\frac{buoy\_3\_long+buoy\_4\_long}{2}}{2}}{2}) \tag{4.37}$$

$$Goal\_2 = (\frac{buoy\_1\_lat + buoy\_2\_lat}{2}, \frac{buoy\_1\_long + buoy\_2\_long}{2}) \tag{4.38}$$

$$Goal\_3 = (\frac{buoy\_3\_lat + \frac{buoy\_3\_lat+buoy\_4\_lat}{2}}{2}, \frac{buoy\_3\_long + \frac{buoy\_3\_long+buoy\_4\_long}{2}}{2}) \tag{4.39}$$

$$Goal\_4 = (\frac{buoy\_1\_lat + \frac{buoy\_1\_lat+buoy\_2\_lat}{2}}{2}, \frac{buoy\_1\_long + \frac{buoy\_1\_long+buoy\_2\_long}{2}}{2}) \tag{4.40}$$

$$Goal\_5 = (\frac{buoy\_3\_lat + buoy\_4\_lat}{2}, \frac{buoy\_3\_long + buoy\_4\_long}{2}) \tag{4.41}$$

$$Goal\_6 = (\frac{\frac{buoy\_5\_lat+buoy\_6\_lat}{2} + buoy\_6\_lat}{2}, \frac{\frac{buoy\_5\_long+buoy\_6\_long}{2} + buoy\_6\_long}{2}) \tag{4.42}$$

$$Goal\_7 = (\frac{\frac{buoy\_3\_lat+buoy\_4\_lat}{2} + buoy\_4\_lat}{2}, \frac{\frac{buoy\_3\_long+buoy\_4\_long}{2} + buoy\_4\_long}{2}) \tag{4.43}$$

$$Goal\_8 = (\frac{buoy\_5\_lat + \frac{buoy\_5\_lat + buoy\_6\_lat}{2}}{2}, \frac{buoy\_5\_long + \frac{buoy\_5\_long + buoy\_6\_long}{2}}{2}) \tag{4.44}$$

$$Goal\_9 = (buoy\_4\_lat, buoy\_4\_long); \tag{4.45}$$

$$Goal\_10 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.46}$$

4.1.9 *Vertical Full Scanning*

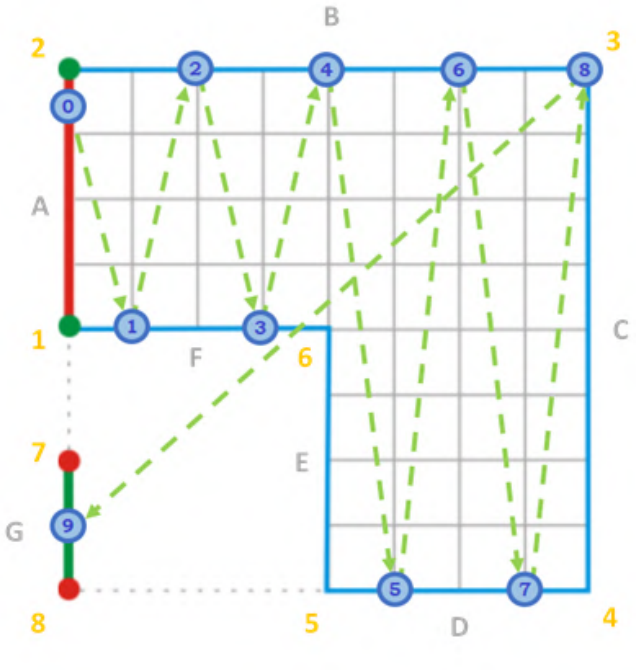

Figura 4.13: *Vertical Full Scanning - goals*.

Caracterização dos *goals*:

- ***Goal 0*** - nível 3; 1/8 da reta A (superior);
- ***Goal 1*** - nível 2; 1/4 da reta F (esquerda);
- ***Goal 2*** - nível 2; 1/4 da reta B (esquerda);
- ***Goal 3*** - nível 2; 1/4 da reta F (direita);
- ***Goal 4*** - nível 1; 1/2 da reta B;
- ***Goal 5*** - nível 2; 1/4 da reta D (esquerda);
- ***Goal 6*** - nível 2; 1/4 da reta B (direita);

- ***Goal 7*** - nível 2; 1/4 da reta D (direita);
- ***Goal 8*** - nível 0; bóia 3;
- ***Goal 9*** - nível 1; 1/2 da reta G;

Cálculos:

$$Goal\_0 = (\frac{\frac{\frac{buoy\_1\_lat+buoy\_2\_lat}{2}+buoy\_2\_lat}{2} + buoy\_2\_lat}{2}, \frac{\frac{\frac{buoy\_1\_long+buoy\_2\_long}{2}+buoy\_2\_long}{2} + buoy\_2\_long}{2}) \tag{4.47}$$

$$Goal\_1 = (\frac{\frac{buoy\_6\_lat+buoy\_1\_lat}{2} + buoy\_1\_lat}{2}, \frac{\frac{buoy\_6\_long+buoy\_1\_long}{2} + buoy\_1\_long}{2}) \tag{4.48}$$

$$Goal\_2 = (\frac{buoy\_2\_lat + \frac{buoy\_2\_lat+buoy\_3\_lat}{2}}{2}, \frac{buoy\_2\_long + \frac{buoy\_2\_long+buoy\_3\_long}{2}}{2}) \tag{4.49}$$

$$Goal\_3 = (\frac{buoy\_6\_lat + \frac{buoy\_6\_lat+buoy\_1\_lat}{2}}{2}, \frac{buoy\_6\_long + \frac{buoy\_6\_long+buoy\_1\_long}{2}}{2}) \tag{4.50}$$

$$Goal\_4 = (\frac{buoy\_2\_lat + buoy\_3\_lat}{2}, \frac{buoy\_2\_long + buoy\_3\_long}{2}) \tag{4.51}$$

$$Goal\_5 = (\frac{\frac{buoy\_4\_lat+buoy\_5\_lat}{2} + buoy\_5\_lat}{2}, \frac{\frac{buoy\_4\_long+buoy\_5\_long}{2} + buoy\_5\_long}{2}) \tag{4.52}$$

$$Goal\_6 = (\frac{\frac{buoy\_2\_lat+buoy\_3\_lat}{2} + buoy\_3\_lat}{2}, \frac{\frac{buoy\_2\_long+buoy\_3\_long}{2} + buoy\_3\_long}{2}) \tag{4.53}$$

$$Goal\_7 = (\frac{buoy\_4\_lat + \frac{buoy\_4\_lat+buoy\_5\_lat}{2}}{2}, \frac{buoy\_4\_long + \frac{buoy\_4\_long+buoy\_5\_long}{2}}{2}) \tag{4.54}$$

$$Goal\_8 = (buoy\_3\_lat, buoy\_3\_long) \tag{4.55}$$

$$Goal\_9 = (\frac{buoy\_7\_lat + buoy\_8\_lat}{2}, \frac{buoy\_7\_long + buoy\_8\_long}{2}) \tag{4.56}$$

4.2 *Smart Scanning*

Após implementados os diversos métodos de varrimento, o passo seguinte foi lidar com o mecanismo *Smart Scanning*. Para uma melhor compreensão da implementação desta parte, é necessário ter em conta o significado das seguintes variáveis:

- ***smart_scanning*** - variável *booleana* que deve ser inicializada com o valor lógico *true* caso se pretenda usar este mecanismo, ou *false* caso contrário;
- ***scanning_depth*** - inteiro (definido pelo utilizador) que define qual a profundidade de varrimento predefinida (1 - *fast*, 2 - *middle*, 3 - *deep*, 4 - *full*);
- ***checkpoint_1_reached*** - variável *booleana* que representa a deteção da passagem pelo *checkpoint* 1. É inicializada com o valor lógico *false*, permanecendo com este valor enquanto o veleiro não atravessar o *checkpoint* 1 e passando a *true* após este momento;
- ***checkpoint_2_reached*** - semelhante à anterior, mas relacionada com o *checkpoint* 2;
- ***end_of_navigation*** - variável inicializada a *false*, usada para detetar o fim da navegação (quando o veleiro atravessa o último *goal*);
- ***current_time*** - usada para guardar o tempo atual (hh:mm) convertido para minutos;
- ***start_time*** - variável usada para guardar o momento de início de prova (no *setup()*), em minutos;
- ***CHECK_POINT_1_DEADLINE*** - constante que representa o tempo - *deadline* 1 - (em minutos) que o veleiro tem para atravessar o *checkpoint* 1 (pode ser alterada no ficheiro ConstSailboat.h);
- ***CHECK_POINT_2_DEADLINE*** - semelhante à constante anterior, mas relativa ao *checkpoint* 2;

No anexo III encontra-se representado o pedaço de código que ilustra a implementação deste mecanismo (integrado no *loop()* do ficheiro WRSC_NOVALANTIA.ino, substituindo a anterior deteção de chegada a um *goal*, explicada na subsecção 2.3.2.1).

A ordem de execução desta implementação é a seguinte:

1. O sistema verifica se o utilizador pretendeu usar a abordagem *Smart Scanning* e se a profundidade de varrimento é *deep* ou *full*:
   a) Em caso afirmativo, são executadas as instruções referidas no ponto 2;
   b) Em caso negativo, é usado o mecanismo de deteção de chegada aos *goals* tradicional (linhas 55 a 66 da imagem do anexo III);
2. Estando o mecanismo *Smart Scanning* ativo, é verificado se o veleiro já atravessou o *goal*, através da função *bowline.arrival_destination()*:

a) Se retornar *"true"*, o sistema deteta se o *goal* em questão é um dos *checkpoints* (caso seja, afeta a variável respetiva) e inicializa o próximo (linhas 21 e 22) - se o *goal* atingido for o último, a variável *end_of_navigation* passa a *true* (linhas 24 a 26);

b) Se retornar *"false"*, é calculado o tempo de prova (tempo atual - tempo inicial):

   i. Se este exceder a *deadline* 1 e o veleiro não tiver passado ainda pelo primeiro *checkpoint*, o atual trajeto é ignorado e o sistema assume como próximo objetivo o *goal* 5 - depois disto o mecanismo *Smart Scanning* é desativado.

   ii. Se este exceder a *deadline* 2 e o veleiro não tiver passado pelo segundo *checkpoint* (mas já pelo primeiro), o atual trajeto é ignorado e é inicializado o *goal* 6 como próximo destino - depois disto o mecanismo *Smart Scanning* é desativado.

A figura abaixo exemplifica como é afetado o vetor *route_race[]* quando um dos *checkpoints* não é atravessado a tempo (neste caso, o *checkpoint* 1), numa situação em que o método de varrimento é o *Horizontal/Vertical Deep Scanning*.

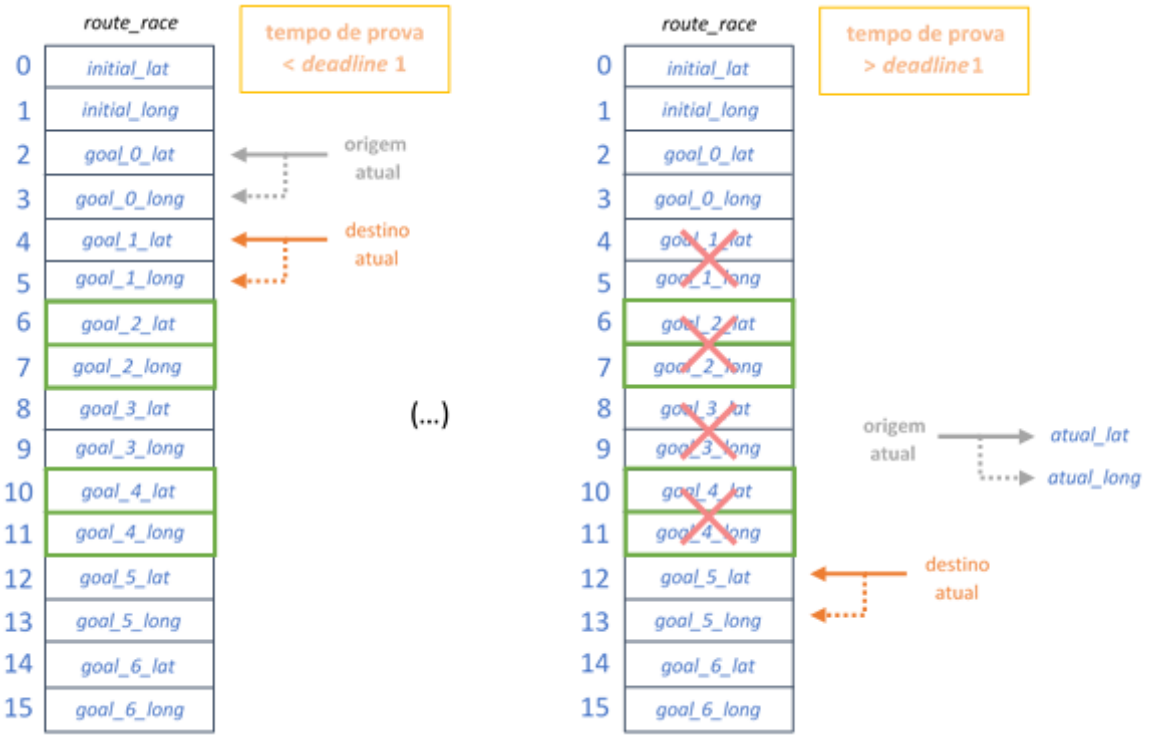

Figura 4.14: Ilustração da atribuição de um novo trajeto no vetor *route_race[]*.

4.3 Decisão da estratégia de navegação

No que toca à solução proposta, falta explicar a implementação da ”ferramenta” de decisão da estratégia de navegação (horizontal ou vertical).

Como foi visto anteriormente, a decisão da estratégia de navegação deriva (caso o utilizador pretenda) da leitura do vento - para um ”vento perpendicular” é escolhida a navegação horizontal, e para um ”vento paralelo” é escolhida a vertical. ´ Antes de se prosseguir com a explicação da implementação propriamente dita, é necessário ter em conta o significado das seguintes variáveis:

- ***wind_factor*** - variável *booleana* que deve ser inicializada com o valor *true* caso o utilizador pretenda que a decisão da estratégia de navegação seja influenciada pela direção do vento. Por outro lado, deve ser inicializada a *false* caso se pretenda usar uma estratégia e um varrimento predefinido;
- ***windAlignment_iterations*** - inteiro que representa as iterações necessárias para calcular a média dos valores da direção do vento (inicializada com o valor 1);
- ***wind_al_avg*** - média dos valores da direção do vento;
- ***scanning_method_decision*** - variável do tipo inteiro cujo valor corresponde ao método de varrimento a implementar: 1 - *Horizontal Fast*, 2 - *Horizontal Middle*, 3 - *Horizontal Deep*, 4 - *Horizontal Full*, 5 - *Vertical Fast*, 6 - *Vertical Middle*, 7 - *Vertical Deep*, 8 - *Vertical Full*;
- ***BUOY_1_LAT*** (...) - coordenada latitude da bóia 1 (equivalente para as outras bóias);
- ***BUOY_1_LONG*** (...) - coordenada longitude da bóia 1 (equivalente para as outras bóias);
- ***goals_number*** - inteiro que representa o número de *goals* associados ao método de varrimento que será implementado.

Nota: A implementação referente a este subcapítulo foi colocada no *setup()* do ficheiro principal do projeto, pois necessita de ser executada logo no início da navegação e apenas uma vez.

Caso não se pretenda usar este mecanismo de escolha da estratégia de navegação, mas sim escolher um método de varrimento de forma predefinida, a variável *wind_factor* deve ser inicializada com o valor lógico *false*.

Caso contrário, e o comportamento sistema é o seguinte:

O primeiro passo é calcular a média dos valores provenientes do cata-vento (*wind_al_avg*).

Para tal é lido um determinado número de amostras de dados, definido pela variável *windAlignment_iterations*, provenientes do cata-vento. É, portanto, com este conjunto de amostras que é calculada a média.

Depois de calculada a média, é verificado a que intervalo de ângulos esta média corresponde, definindo a estratégia de navegação consoante o intervalo.

No final, é conjugado o tipo de profundidade de varrimento definido com a estratégia escolhida (através de um *switch-case*) resultando no método/tipo de varrimento a aplicar na prova.

A figura 4.15 representa o diagrama dos passos anteriormente descritos desta implementação.

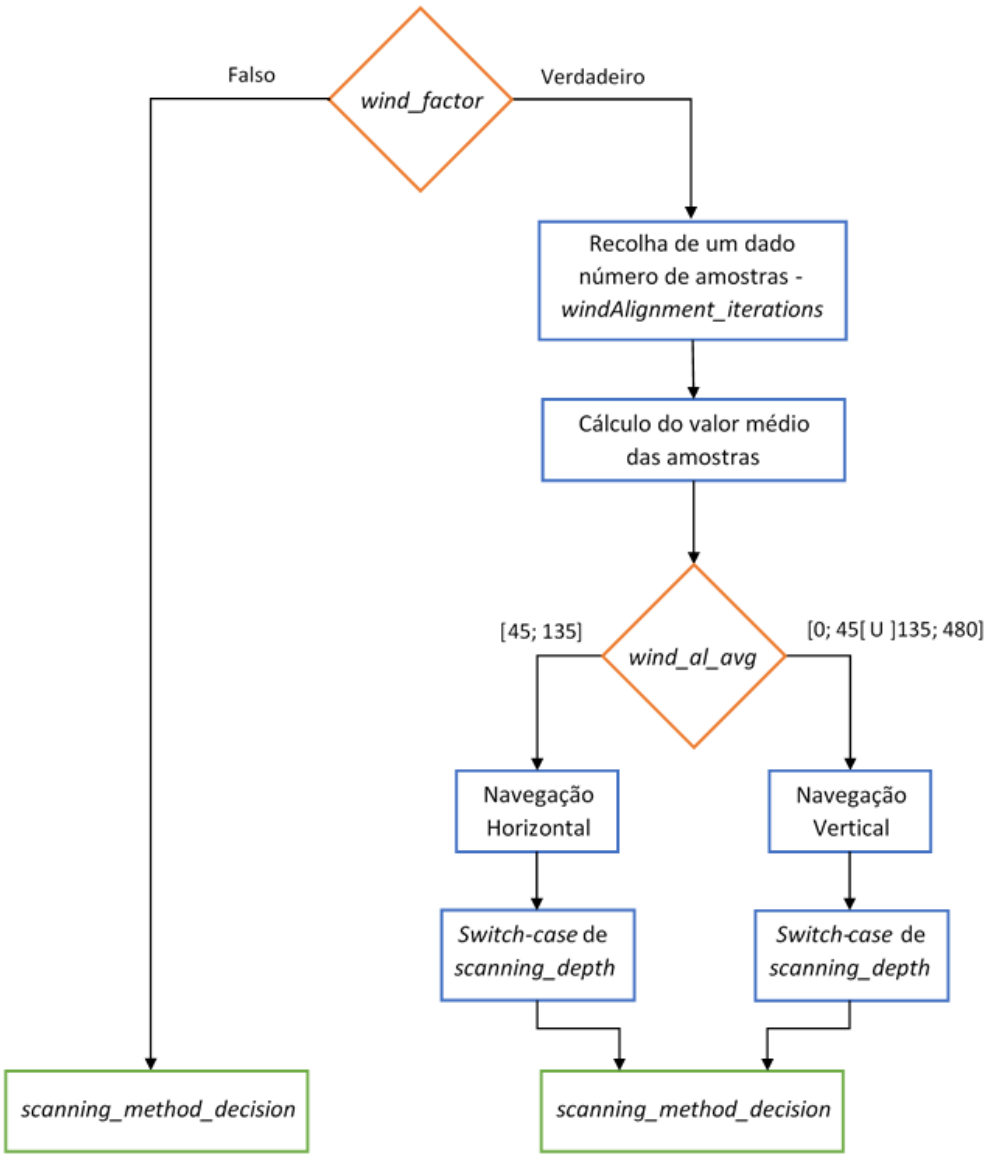

Figura 4.15: Implementação da decisão da estratégia e do método de navegação.

Nota: na ocorrência de erros no cálculo da média da direção do vento (ex: leituras incorretas por parte do cata-vento) o método de varrimento por defeito é o *Horizontal Fast Scanning*, por ser um dos mais seguros e rápidos.

4.4 Outros ajustes

Foram também realizados alguns ajustes no código base fornecido, de forma a enquadrar toda a implementação do *Area Scanning* no projeto e também por necessidade/correção de alguns aspetos.

ConstSailboat.h

```
const unsigned int ROTARY_ENC_MIN_PULSE_WIDTH = 700;
const unsigned int ROTARY_ENC_MAX_PULSE_WIDTH = 3100;
```

Figura 4.16: Valores mínimo e máximo lidos pelo cata-vento (*screenshot* do Arduino IDE).

As constantes definidas na figura 4.16 dizem respeito aos valores PWM máximo e mínimo (respetivamente) lidos pelo sensor do cata-vento. Estas constantes devem ser alteradas sempre que o sensor do cata-vento é trocado, pois diferentes sensores possuem ligeiras diferenças na gama de valores que recolhem.

```
const unsigned int SAIL_MIN_PULSE_WIDTH = 1900;
const unsigned int SAIL_MAX_PULSE_WIDTH = 2030;
const unsigned int RUDDER_MIN_PULSE_WIDTH = 900;
const unsigned int RUDDER_MAX_PULSE_WIDTH = 1800;
```

Figura 4.17: Valores mínimos e máximos suportados pelos servos das velas e do leme, respetivamente.

As constantes *SAIL_MIN_PULSE_WIDTH* e *SAIL_MAX_PULSE_WIDTH* representam os valores mínimo e máximo aplicáveis ao servo das velas (quanto menor o valor de *input* no servo, menor a tensão nas velas e mais folgadas ficam, e vice-versa). Apesar de o servo ter capacidade para suportar valores mais baixos que 1900 e maiores que 2030, as constantes foram definidas desta forma para evitar que o fio preso ao servo e que caça/folga as velas fique emaranhado (algo que acontece sistematicamente com uma gama de valores mais alargada e que torna quase impossível a navegação do veleiro).

O valore mínimo associado ao servo do leme também foi ligeiramente diminuído, depois de algumas verificações e testes nesse sentido.

Além disso, foram adicionadas constantes específicas da implementação do *Area Scanning* (figura 4.18), nomeadamente as definições das *deadlines* e das coordenadas das bóias (vértices) que caracterizam a área da prova.

```
//Area Scanning
const unsigned int CHECK_POINT_1_DEADLINE = 15;
const unsigned int CHECK_POINT_2_DEADLINE = 20;
const double BUOY_1_LAT = 38.659209;
const double BUOY_1_LONG = -9.205917;
const double BUOY_2_LAT = 38.659444;
const double BUOY_2_LONG = -9.205919;
const double BUOY_3_LAT = 38.659413;
const double BUOY_3_LONG = -9.205409;
const double BUOY_4_LAT = 38.658874;
const double BUOY_4_LONG = -9.205386;
const double BUOY_5_LAT = 38.658873;
const double BUOY_5_LONG = -9.205639;
const double BUOY_6_LAT = 38.659211;
const double BUOY_6_LONG = -9.205641;
const double BUOY_7_LAT = 38.659080;
const double BUOY_7_LONG = -9.205895;
const double BUOY_8_LAT = 38.658889;
const double BUOY_8_LONG = -9.205876;
```

Figura 4.18: Definição das *deadlines* e das coordenadas das bóias.

Data_acquisition.cpp

A função ***windVanePosition()*** sofreu pequenas modificações. O valor subtraído na segunda instrução da função passou de -35 para -80. Este valor representa um *offset* e é subtraído para o ângulo 0º com a proa da embarcação, devendo ser verificado/alterado sempre que o sensor do cata-vento é trocado. Foi também necessário um passo adicional para converter os ângulos negativos para positivos.2

remote.cpp

A função ***sendPlannedTrack()*** também foi modificada, pois estava implementada para lidar com um número fixo de *goals* (definidos pela constante *NUMBER_BUOYS* e usada maioritariamente para a *regatta*), e para o caso da prova *Area Scanning* o número de *goals* varia consoante o método de varrimento. Desta forma, o número de *goals* (*goals_number*) passa a ser passado como argumento desta função.

Capítulo 5

Validação e Resultados

Terminada a implementação, prosseguiu-se com a fase de testes e validação (e posterior análise de resultados).

Primeiramente foram testados individualmente - testes unitários - os módulos implementados (tipos de varrimento, cálculo da média da direção do vento, etc), passando depois à validação do sistema com a integração de todos os módulos desenvolvidos. Por fim, foram feitos testes reais com o veleiro a navegar, no evento REX (explicado mais a frente) organizado pela Marinha Portuguesa.

5.1 Testes unitários

A primeira fase de testes serviu para avaliar se cada módulo desenvolvido estava em condições de ser integrado no projeto.

Nota: Não foram realizados testes unitários para o módulo *Smart Scanning*, pois este necessita da integração dos outros módulos para ser testado. A sua validação será feita portanto no subcapítulo 5.2.

5.1.1 Tipos de varrimento

A ordem de validação foi idêntica à ordem de implementação, pelo que se começou por testar cada tipo/método de varrimento.

Para tal foi desenvolvida uma pequena ferramenta - Area_Scanning.cpp - (constituída essencialmente pelas funções associadas a cada método, apenas com a adição de código para *debug*) no *Visual Studio 2017*, para uma percepção mais visual dos testes de cada método.

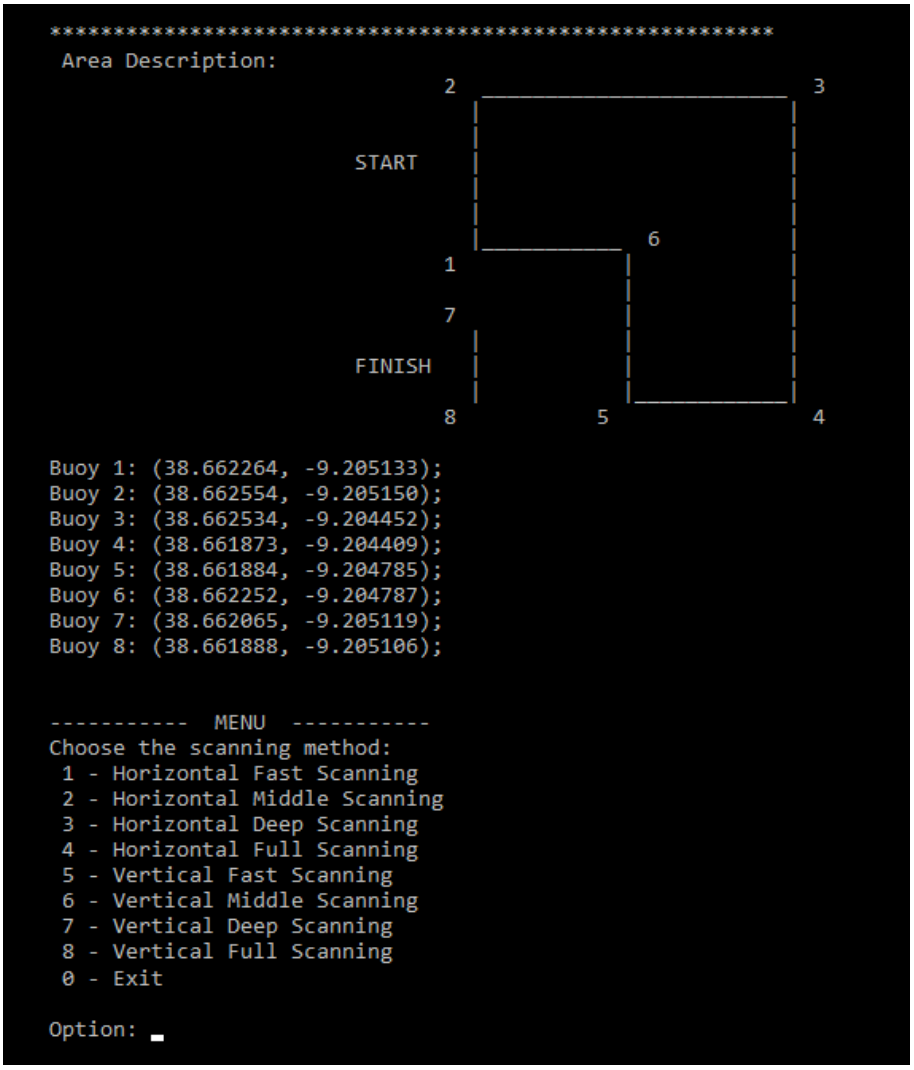

Figura 5.1: *Interface* inicial da ferramenta de *debug* dos tipos de varrimento.

Ao "correr" esta ferramenta, é apresentada na consola o conteúdo ilustrado na figura 5.1, constituído pela representação da área da prova, as coordenadas das bóias (que podem ser modificadas no ficheiro Area_Scanning.h) e o menu de interação com o utilizador, que permite escolher o método de varrimento a implementar.

Com a ajuda do *Google Maps*, é possível ter uma representação das coordenadas das bóias no mapa, na qual foram traçadas as retas que definem a área da prova:

Figura 5.2: Representação das bóias (ícones amarelos).

Horizontal Fast Scanning

Selecionando a opção 1, são calculados e apresentados os *goals* associados ao *Horizontal Fast Scanning*, assim como ilustrações da trajetória e da pontuação previstas.

Nota: A trajetória prevista apresentada não é uma representação exata, estando limitada no que toca à definição da rota.

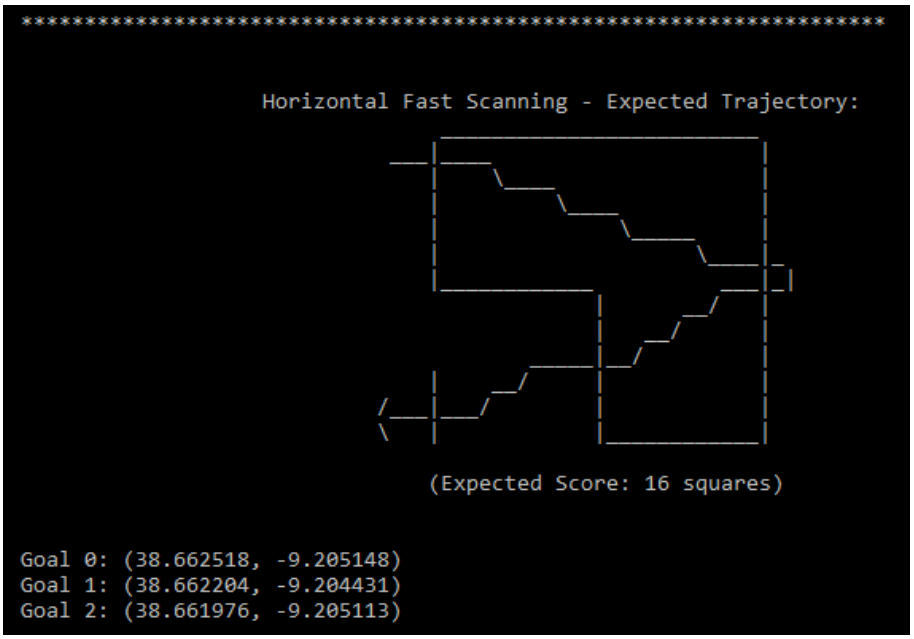

Figura 5.3: *Screenshot* da interface associada ao *Horizontal Fast Scanning*.

A figura 5.4 contém a representação geográfica dos *goals* gerados, sobrepostos à imagem 5.2.

Figura 5.4: Representação dos *goals* gerados associados ao *Horizontal Fast Scanning*.

Vertical Fast Scanning

Para se verificar o método *Vertical Fast Scanning* seleciona-se a opção 5:

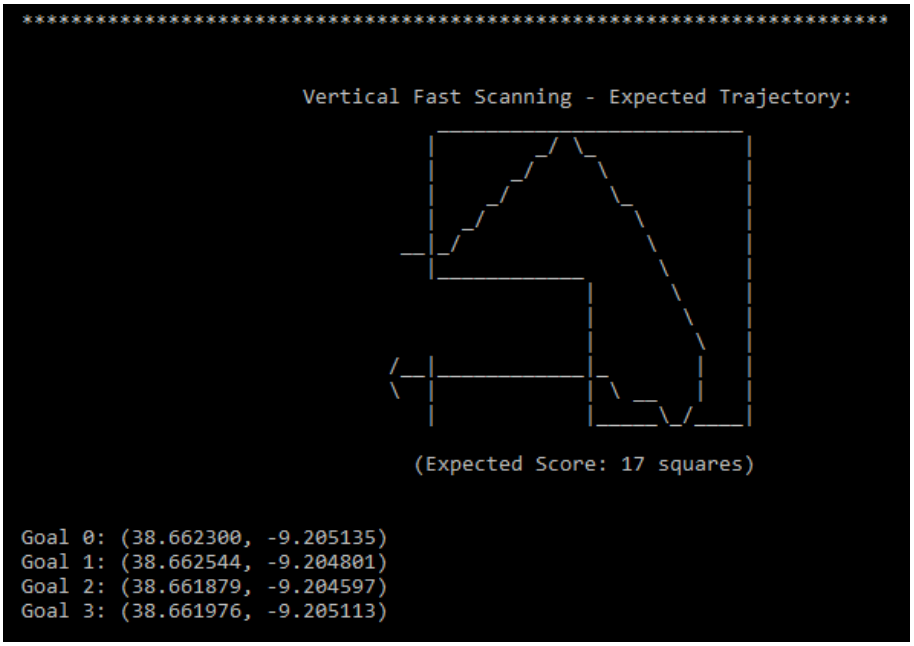

Figura 5.5: *Screenshot* da interface associada ao *Vertical Fast Scanning*.

Representação dos *goals* gerados:

Figura 5.6: Representação dos *goals* gerados associados ao *Vertical Fast Scanning*.

Horizontal Middle Scanning

Goals calculados para o método *Horizontal Middle Scanning*:

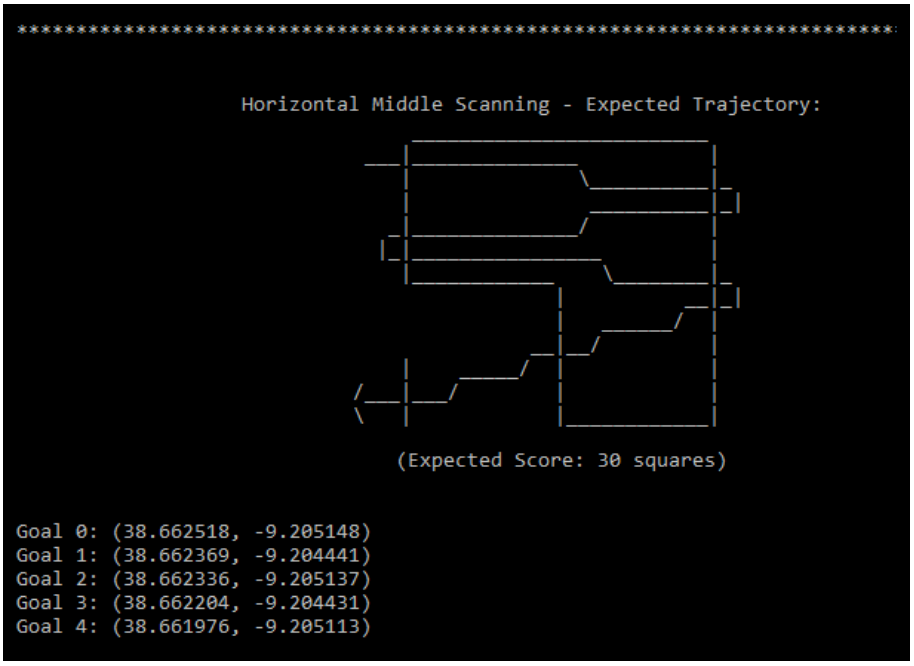

Figura 5.7: *Screenshot* da interface associada ao *Horizontal Middle Scanning*.

Representação dos *goals* gerados:

Figura 5.8: Representação dos *goals* gerados associados ao *Horizontal Middle Scanning*.

Vertical Middle Scanning

Goals calculados para o método *Vertical Middle Scanning*:

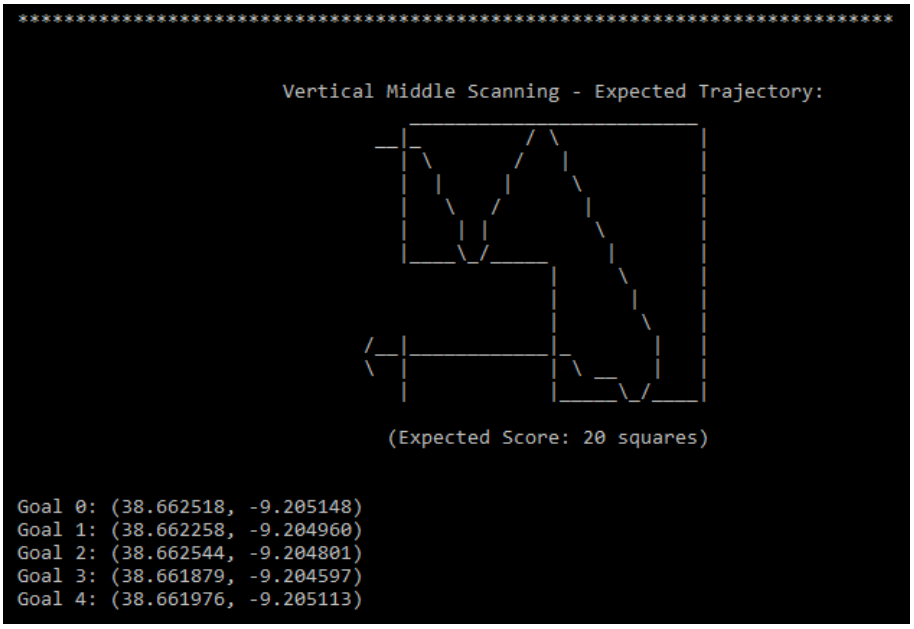

Figura 5.9: *Screenshot* da interface associada ao *Vertical Middle Scanning*.

Representação dos *goals* gerados:

Figura 5.10: Representação dos *goals* gerados associados ao *Vertical Middle Scanning*.

Horizontal Deep Scanning

Goals gerados associados ao método *Horizontal Deep Scanning*:

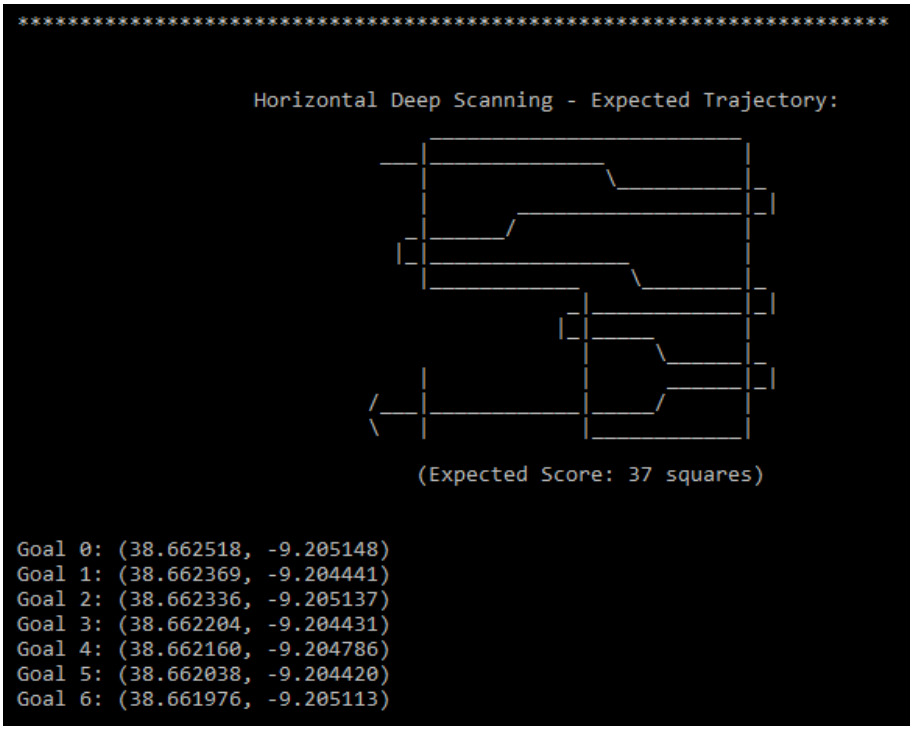

Figura 5.11: *Screenshot* da interface associada ao *Horizontal Deep Scanning*.

Representação dos *goals* gerados:

Figura 5.12: Representação dos *goals* gerados associados ao *Horizontal Deep Scanning*.

Vertical Deep Scanning

Conjunto de coordenadas dos *goals* gerados para o método *Vertical Deep Scanning*:

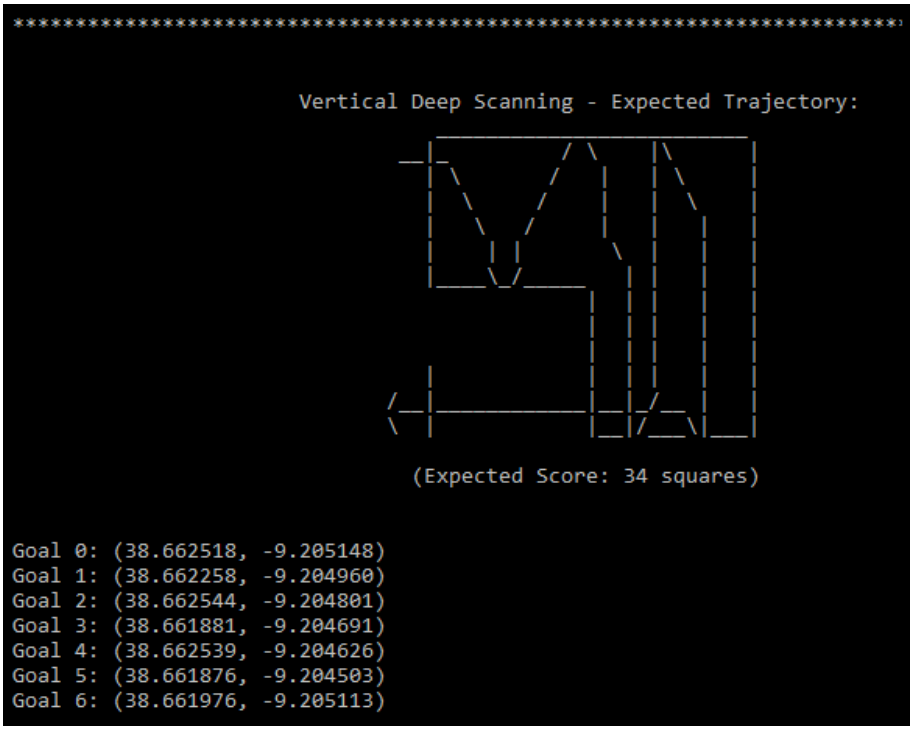

Figura 5.13: *Screenshot* da interface associada ao *Vertical Deep Scanning*.

Representação dos *goals* gerados:

Figura 5.14: Representação dos *goals* gerados associados ao *Vertical Deep Scanning*.

Horizontal Full Scanning

Goals calculados:

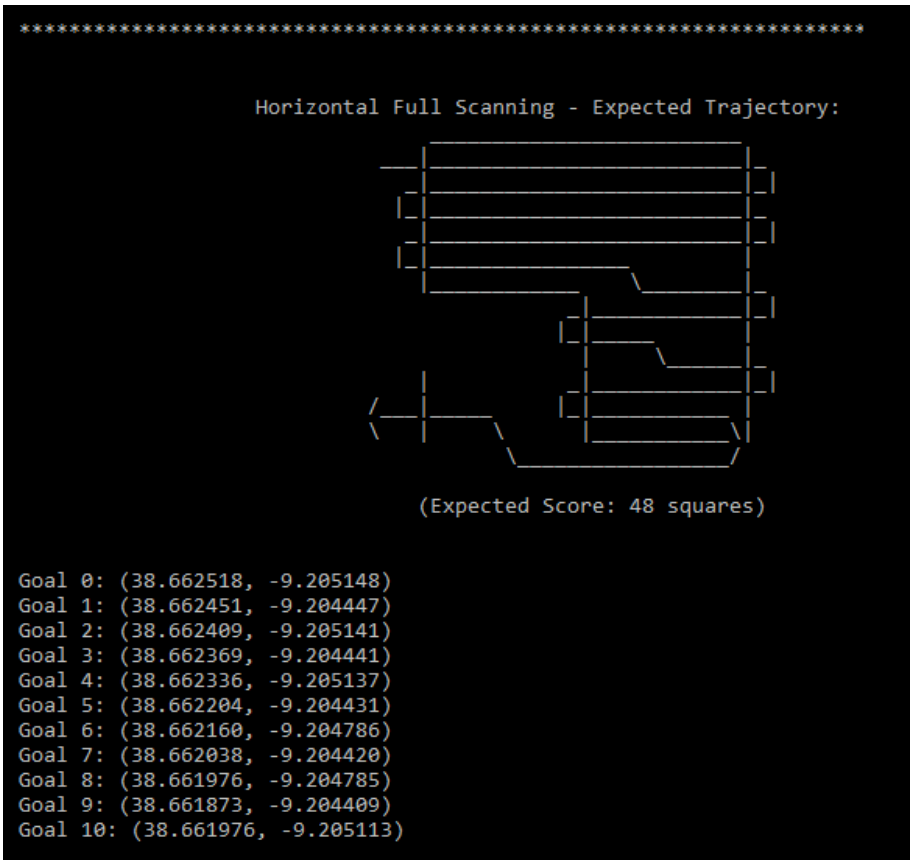

Figura 5.15: *Screenshot* da interface associada ao *Horizontal Full Scanning*.

Representação dos *goals* gerados:

Figura 5.16: Representação dos *goals* gerados associados ao *Horizontal Full Scanning*.

Vertical Full Scanning

Goals calculados:

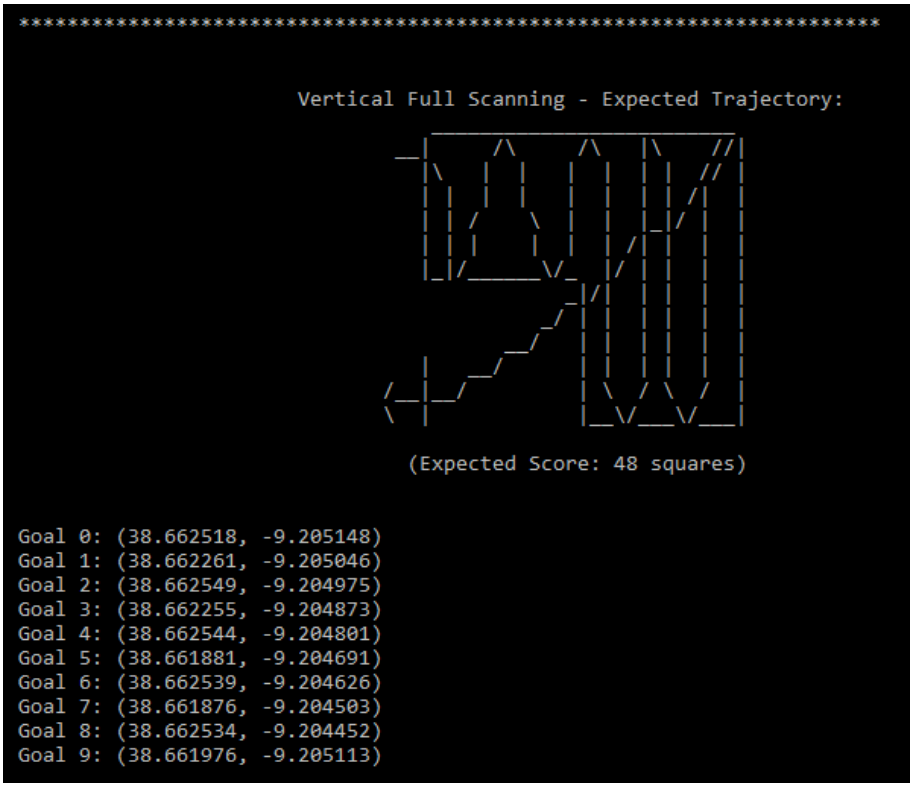

Figura 5.17: *Screenshot* da interface associada ao *Vertical Full Scanning*.

Representação dos *goals* gerados:

Figura 5.18: Representação dos *goals* gerados associados ao *Vertical Full Scanning*.

5.1.1.1 Análise aos resultados anteriores

Através da observação da localização dos *goals* gerados no mapa para cada método de varrimento, pode afirmar-se que os resultados foram os esperados, coincidindo com os *goals* que tinham sido definidos no capítulo da solução proposta.

De notar também que, mesmo que a definição da área da prova não seja perfeita ou à escala (como foi o caso nestes testes, onde as retas definidas pelas bóias são ligeiramente desproporcionais e desalinhadas), o sistema tem capacidade de gerar os *goals* esperados, pois estes têm por base o cálculo de valores médios entre as bóias.

Desta forma é possível concluir que estes módulos podem ser integrados no projeto.

5.1.2 Cálculo do valor médio da direção do vento

Para testar a leitura e cálculo do valor médio da direção do vento, foi necessário expor o conteúdo das variáveis em questão (os valores lidos pelo sensor e o posterior valor médio), através do monitor série do Arduino IDE. Para tal, foram efetuados 4 experimentos, cada um com posições diferentes do cata-vento.

Experimento 1: cata-vento alinhado com a proa

A figura 5.19 representa um teste em que o cata-vento se encontra aproximadamente alinhado com a proa do veleiro (neste caso é fictícia pois foi usado o kit do *hardware* em separado, que será explicado posteriormente), pelo que o valores lidos devem rondar os 0º.

Nota: Em todos os quatro testes deste módulo foram efetuadas ligeiras oscilações ao cata-vento aquando a sua leitura, para que os valores recolhidos não fossem sempre os mesmos. Desta forma as experiências aproximaram-se mais da realidade.

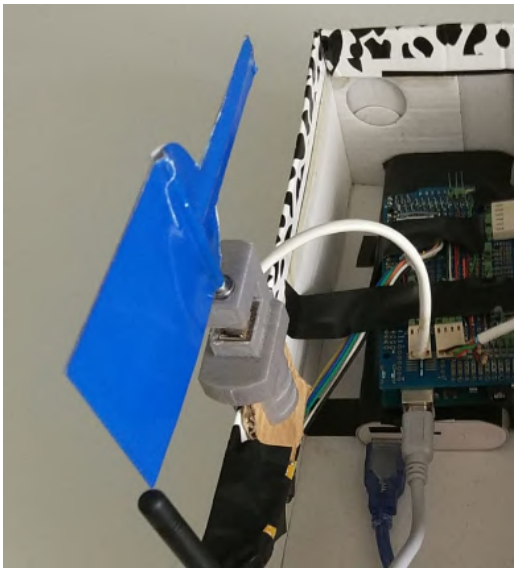

Figura 5.19: Ilustração do cata-vento aproximadamente orientado para a proa.

O sistema recolheu 10 amostras provenientes do sensor do cata-vento (representadas na figura 5.20 pela expressão *"Wind Alignment"*), calculando de seguida a média destes valores (*Wind Alignment Avarage*):

```
Wind Alignment: 8
Wind Alignment: 8
Wind Alignment: 8
Wind Alignment: 8
Wind Alignment: 9
Wind Alignment: 10
Wind Alignment: 9
Wind Alignment: 11
Wind Alignment: 4
Wind Alignment: 1
Wind Alignment Avarage: 7.60
```

Figura 5.20: Amostras da direção do vento e respetiva média (cata-vento aproximadamente orientado para a proa).

Experimento 2: cata-vento alinhado a bombordo

O processo do primeiro teste repetiu-se para o segundo, mas desta feita com o cata-vento orientado para bombordo (onde o ângulo deve rondar os 90º), ilustrado na figura abaixo. As amostras recolhidas e a respetiva média calculada são apresentadas na figura 5.22.

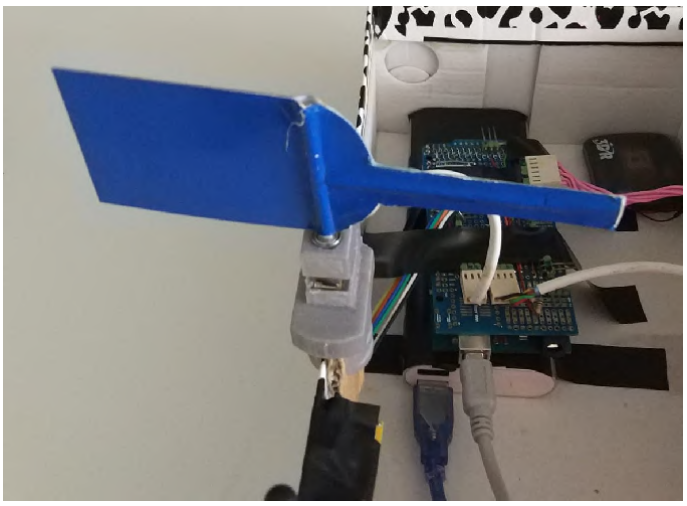

Figura 5.21: Ilustração do cata-vento aproximadamente orientado para bombordo.

```
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 90
Wind Alignment: 88
Wind Alignment: 86
Wind Alignment: 85
Wind Alignment: 83
Wind Alignment: 82
Wind Alignment: 82
Wind Alignment Avarage: 87.20
```

Figura 5.22: Amostras da direção do vento e respetiva média (cata-vento aproximadamente orientado para bombordo).

Experimento 3: cata-vento alinhado com a popa

O terceiro teste consistiu na validação deste módulo para a situação em que o cata-vento está praticamente alinhado com a popa, sendo de esperar um ângulo a rondar os 180º.

Figura 5.23: Ilustração do cata-vento aproximadamente orientado para a popa.

```
Wind Alignment: 172
Wind Alignment: 173
Wind Alignment: 178
Wind Alignment: 178
Wind Alignment: 178
Wind Alignment: 180
Wind Alignment: 180
Wind Alignment: 179
Wind Alignment: 179
Wind Alignment: 179
Wind Alignment Avarage: 177.60
```

Figura 5.24: Amostras da direção do vento e respetiva média (cata-vento aproximadamente orientado para a popa).

Experimento 4 : cata-vento alinhado a estibordo

No quarto e último teste, a orientação do cata-vento foi definida para estibordo. Relembrar que os valores trabalhados pelo sistema que foram lidos pelo sensor de *Hall* e posteriormente convertidos são limitados de 0º a 180º, pelo que neste caso seja de esperar um ângulo aproximadamente de 90º (novamente).

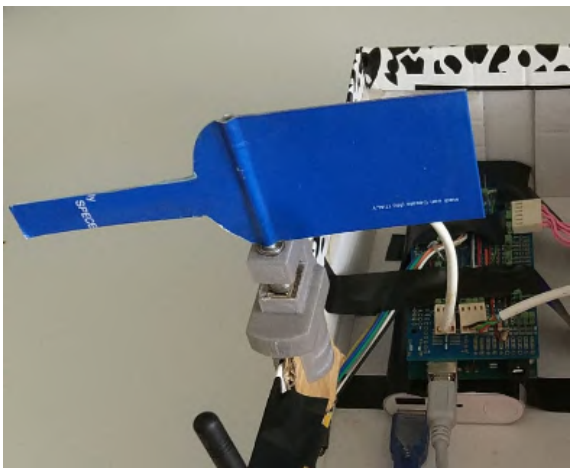

Figura 5.25: Ilustração do cata-vento aproximadamente orientado para estibordo.

```
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 93
Wind Alignment: 94
Wind Alignment: 94
Wind Alignment: 95
Wind Alignment: 96
Wind Alignment: 97
Wind Alignment: 97
Wind Alignment Avarage: 94.20
```

Figura 5.26: Amostras da direção do vento e respetiva média (cata-vento aproximadamente orientado para estibordo).

5.1.2.1 Análise aos resultados anteriores

Tendo em conta que, no primeiro experimento, o cata-vento se encontrava aproximadamente alinhado com a proa, o esperado seria obter amostras a rondar os 0º, com uma média também próxima deste valor. Os valores obtidos pertencem a um intervalo de 1º e 8º, resultando numa média de 7.60º. É possível afirmar, portanto, que o resultado coincidiu com a expectativa, pois 7.60º é um valor bastante próximo de 0º, considerando a escala do problema: [0º, 180º] x 2 (notar também que a posição do cata-vento foi definida "a olho", com ligeiras oscilações, pelo que dificilmente seriam obtidas amostras mais próximas de 0º).

No experimento 2 seriam de esperar valores próximos de 90º (cata-vento orientado a bombordo), e foram obtidas amostras entre 82º e 92º, com uma média de 87.20º.

Na terceira experiência (cata-vento alinhado com a popa, sendo expectável um ângulo $\approx$ 180º) foi obtida uma média de 177.60º para uma gama de amostras de 172º a 180º.

No último experimento, seriam de esperar novamente valores próximos de 90º, apesar de neste caso o cata-vento se encontrar alinhado para estibordo. Os resultados obtidos foram: amostras entre 92º e 97º, e um valor médio de 94.20.

Desta forma, as conclusões acerca dos experimentos 2, 3 e 4 são semelhantes às conclusões tiradas no experimento 1: os resultados obtidos corresponderam as expectativas.

5.1.3 Decisão da estratégia de navegação

Depois de testado o módulo de cálculo da média da direção do vento, foi feita a validação da parte que decide a estratégia de navegação a utilizar.

A figura 5.27 representa três testes, onde foram usadas amostras de ângulos diferentes para cada situação, tendo o sistema decidido usar as estratégias destacadas a azul:

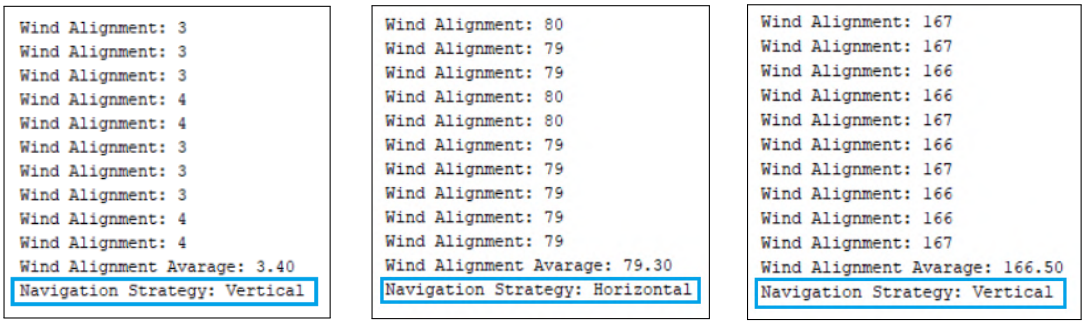

Figura 5.27: Três situações ilustrativas de decisões da estratégia de navegação.

De recordar que a decisão da estratégia deve recair sobre a navegação vertical quando o ângulo de incidência do vento pertence ao intervalo [0º, 45º[U]135º, 180º]. Por outro lado, deve ser escolhida a navegação horizontal quando este ângulo pertence ao intervalo [45º, 135º].

Na primeira situação, o sistema recebeu o conjunto de dados provenientes do cata-vento e calculou a sua média, com o valor de 3.40º. Ora a estratégia decidida perante este valor foi a vertical, que vai de encontro ao esperado (3.40º $\in$ [0º, 45º[).

No segundo teste (imagem central) a estratégia escolhida foi a horizontal, para uma média angular de 79.30º. Este valor pertence ao intervalo [45º, 135º], pelo que a decisão também foi acertada.

No terceiro e último teste a média das amostras foi 166.50º. Tendo em conta que este valor pertence a]135º, 180º], o sistema comportou-se da forma esperada ao decidir pela estratégia vertical.

É portanto possível concluir que, tendo em conta os três testes realizados, este módulo tem o comportamento esperado, para qualquer conjunto de amostras.

5.2 Testes de integração

Os testes de integração, como o próprio nome indica, têm o propósito de integrar os módulos implementados (e testados individualmente) no projeto base, de forma a validar o mesmo e perceber como se comporta o sistema como um todo.

Foi usado um kit de *hardware* numa caixa, composto pelos sensores, pela placa Arduino, pela antena de rádio e por uma *powerbank* (para alimentar a placa Arduino). A utilização deste kit permite validar e testar o código do projeto de uma forma mais prática, sem ser necessário montar e transportar o veleiro para efetuar estes teses.

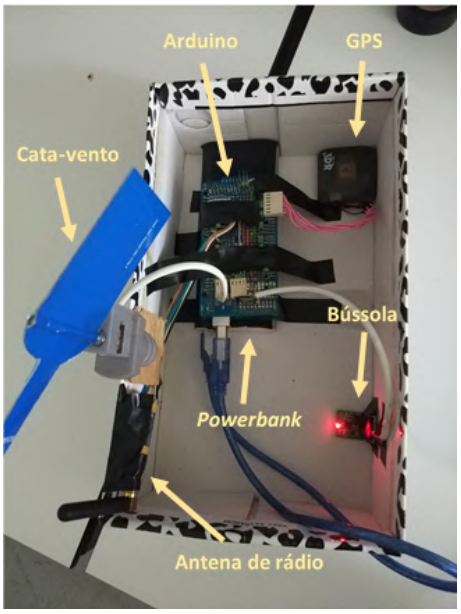

Figura 5.28: Kit de teste.

Para auxiliar a compreensão e percepção do comportamento do sistema durante a execução do código e de que forma estão as variáveis a ser afetadas (*debug*) foi utilizada uma ferramenta na dissertação [29] (além do monitor série do Arduino IDE).

Esta ferramenta tem por base a comunicação via rádio com o Arduino, permitindo observar os dados do sistema em tempo real e fazer a sua recolha.

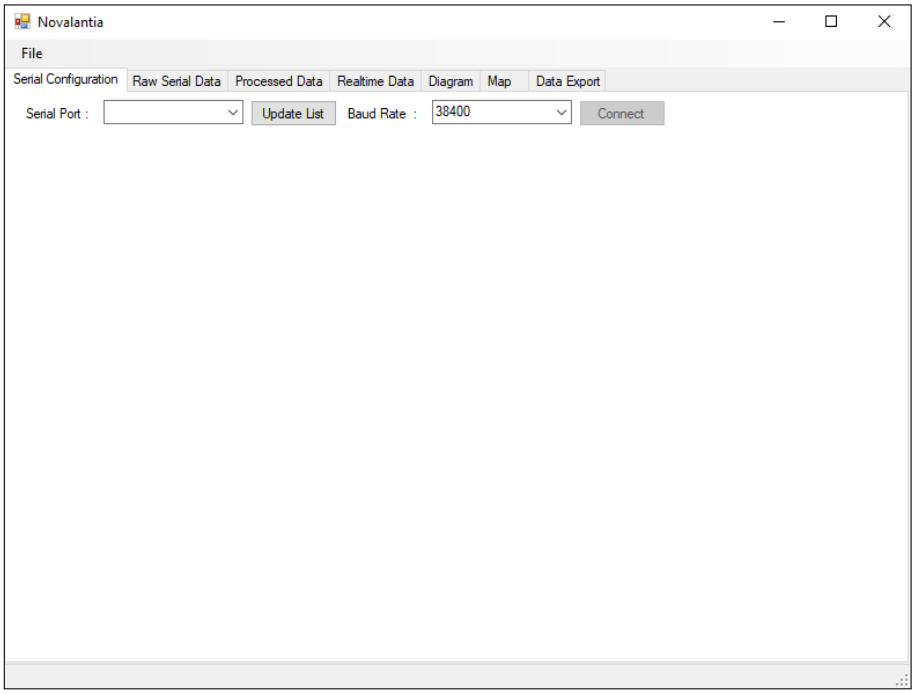

Figura 5.29: Ferramenta desenvolvida em [29].

Esta ferramenta possuí várias funcionalidades:

- Emparelhamento com a antena de rádio conectada ao Arduino;
- Apresentação direta da *string* enviada com os dados do sistema;
- Apresentação mais intuitiva de todos os detalhes acerca dos dados recolhidos pelos sensores, assim como outras informações importantes, como as coordenadas do próximo *goal* ou a hora do sistema do GPS;
- Exibição de um mapa com a localização dos *goals* gerados/inseridos;
- Possibilidade de exportar todos os dados recolhidos para um ficheiro *.json*.

Para este caso, as funcionalidades de maior interesse são a da apresentação de todos os detalhes acerca dos dados recolhidos e a exibição dos *goals* no mapa.

As figuras 5.30 e 5.31 representam as *interfaces* com o utilizador associadas a estas duas funcionalidades, respetivamente.

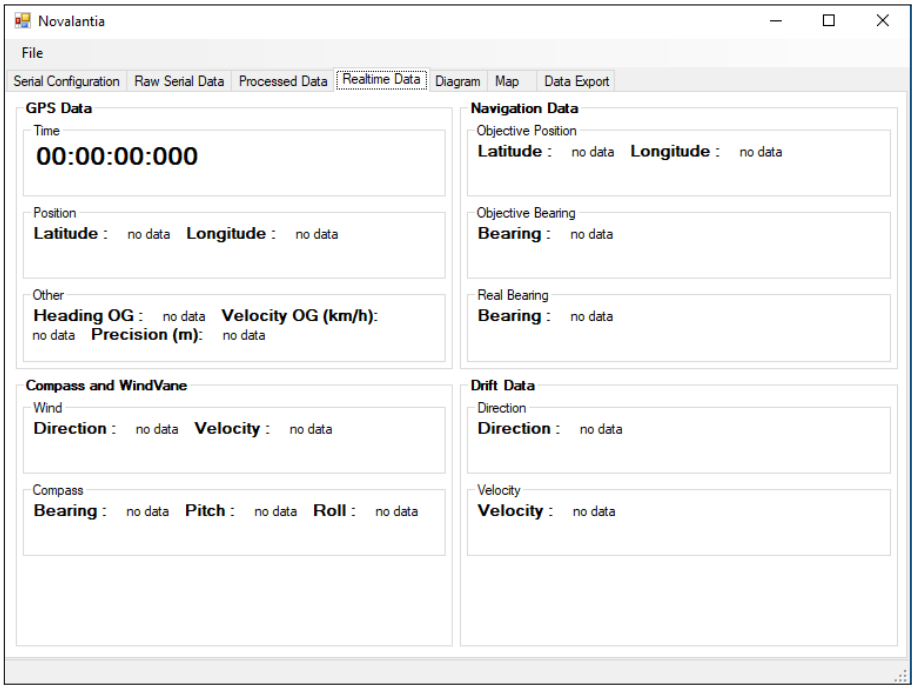

Figura 5.30: Apresentação dos dados em tempo real.

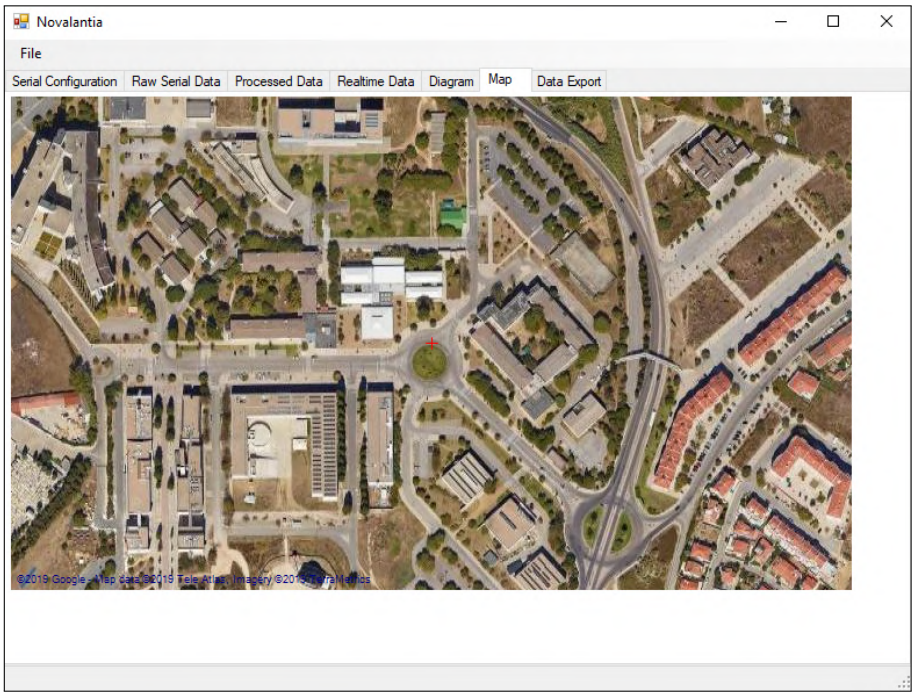

Figura 5.31: Exibição do mapa.

5.2.1 Escolha do método de varrimento e atribuição dos *goals*

Os primeiros testes de integração realizados consistiram em validar se o sistema é capaz de gerar os *goals* corretamente, associados ao tipo de varrimento predefinido e com

a adição do módulo da decisão da estratégia de navegação, consoante a direção do vento.

Estes testes foram efetuados na área da figura 5.32, definida pelas bóias com as seguintes coordenadas (latitude, longitude):

- Bóia 1: (38.659209, -9.205917);
- Bóia 2: (38.659444, -9.205919);
- Bóia 3: (38.659413, -9.205409);
- Bóia 4: (38.658874, -9.205386);
- Bóia 5: (38.658873, -9.205639);
- Bóia 6: (38.659211, -9.205641);
- Bóia 7: (38.659080, -9.205895);
- Bóia 8: (38.658889, -9.205876);

Figura 5.32: Bóias e área de testes.

Nota: As linhas a vermelho foram introduzidas manualmente, para dar uma melhor percepção da área da prova.

5.2.1.1 Profundidade de varrimento: *Fast*

Na primeira experiência foi definida uma profundidade de varrimento *fast*.

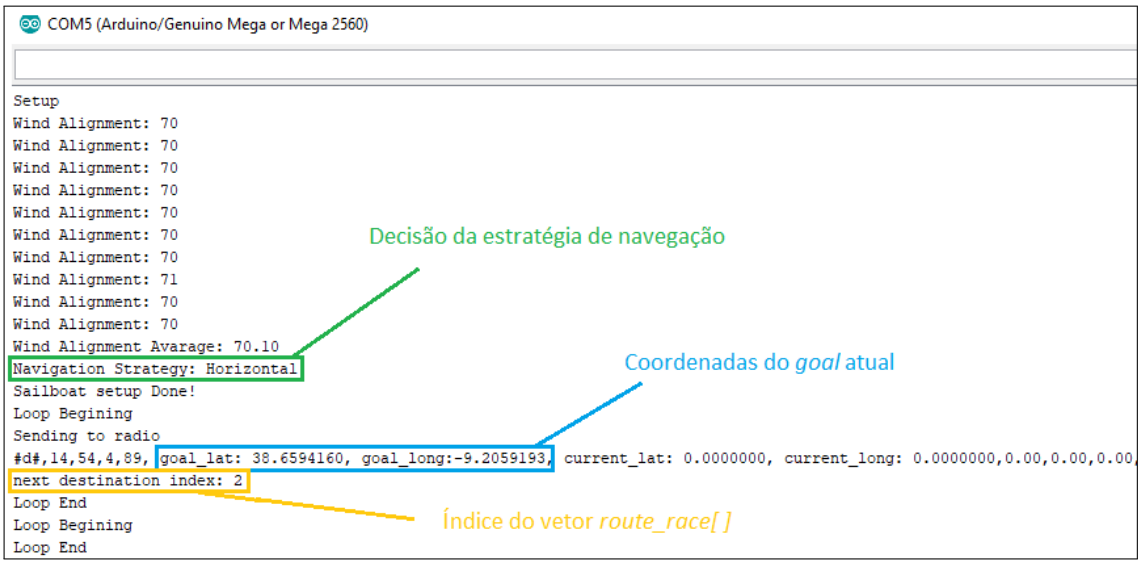

Figura 5.33: Dados do sistema no inicio da navegação. Profundidade *fast* e 70.10º de direção do vento.

Na figura 5.33 estão representadas algumas informações (em tempo real) acerca do comportamento do sistema, numa fase inicial da navegação.

A primeira parte destacada diz respeito ao cálculo da média da direção do vento e posterior decisão da estratégia de navegação.

Antes de terminar o *setup()* (na imagem representado pelo *print* "Sailboat setup Done!") o sistema calcula os *goals* associados ao método, definido a partir da conjugação da profundidade de varrimento e da estratégia de navegação. Neste caso, a profundidade definida (*fast*) e a estratégia decidida (horizontal) deram origem à escolha do método *Horizontal Fast Scanning*), juntamente com o cálculo dos *goals* associados.

Na figura 5.33, marcado pelo rectângulo azul, encontra-se a definição do *goal* atual, que, estando no inicio da navegação, é o *goal 0*.

Destacado a laranja está o índice do vetor que contém as coordenadas dos *goals*, *route_race[]*. Esta informação é útil para perceber qual o número do *goal* que o veleiro está a tentar alcançar no momento (índice = 2 -> *goal 0*, índice = 4 -> *goal 1*, índice = 6 -> *goal 2*, etc.).

Com a ajuda da ferramenta anteriormente mencionada, é possível ter uma visualização das coordenadas dos *goals* (representados pelos símbolos amarelos) gerados no mapa, como mostra a figura 5.34.

Figura 5.34: *Goals* gerados associados ao *Horizontal Fast Scanning*.

Num segundo experimento, foi rodado o cata-vento de forma a resultar numa média angular pertencente ao intervalo [0º, 45º[U]135º, 180º]. Neste caso, o comportamento do sistema foi o seguinte:

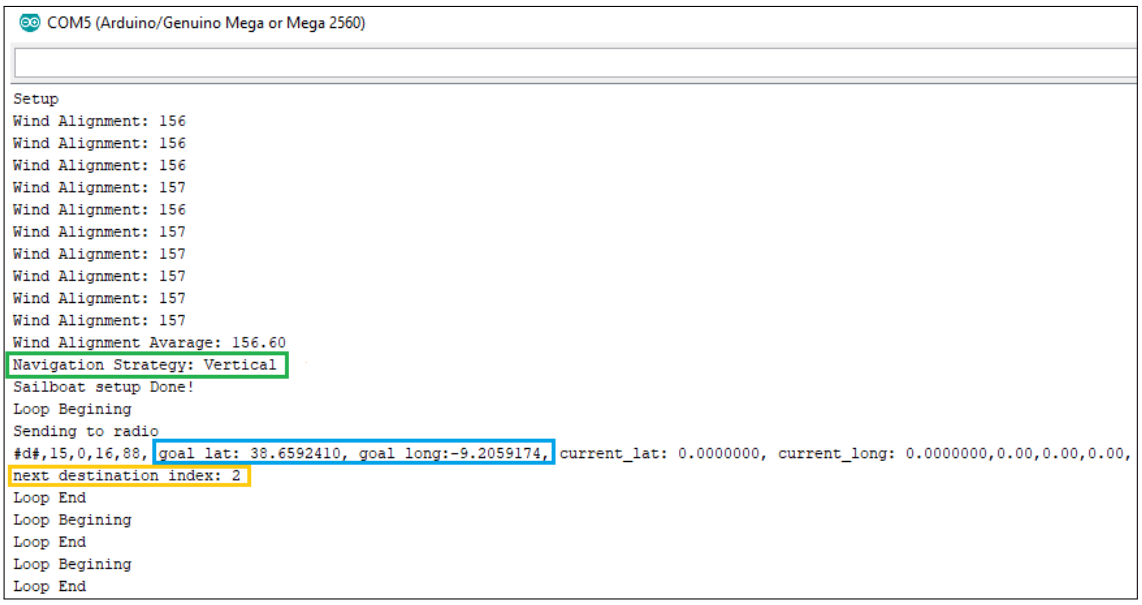

Figura 5.35: Dados do sistema no inicio da navegação. Profundidade *fast* e direção do vento de 156.60º.

Como se pode ver destacado a verde, a estratégia de navegação escolhida foi vertical, dando origem ao método *Vertical Fast Scanning*, cujos *goals* gerados encontram-se representados na figura 5.36.

As coordenadas do *goal* atual (representadas pela área azul) correspondem às coordenadas do *goal 0*, que seria o expectável, assim como o índice com o valor 2.

Figura 5.36: *Goals* gerados associados ao *Vertical Fast Scanning*.

5.2.1.2 Profundidade de varrimento: *Middle*

Todo o processo anterior foi repetido para uma profundidade de varrimento *middle*, sendo que para o primeiro experimento foi definido o cata-vento orientado para bombordo:

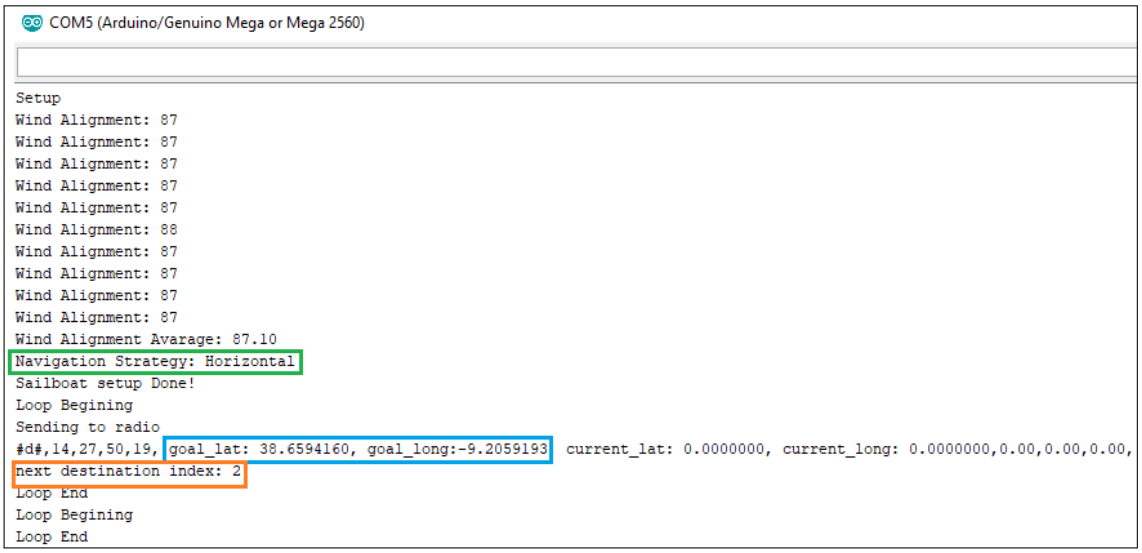

Figura 5.37: Dados do sistema no inicio da navegação. Profundidade *middle* e direção do vento de 87.10º.

Goals gerados:

Figura 5.38: *Goals* gerados associados ao *Horizontal Middle Scanning*.

Na segunda experiência, rodou-se o cata-vento de forma a este ficar orientado com a proa:

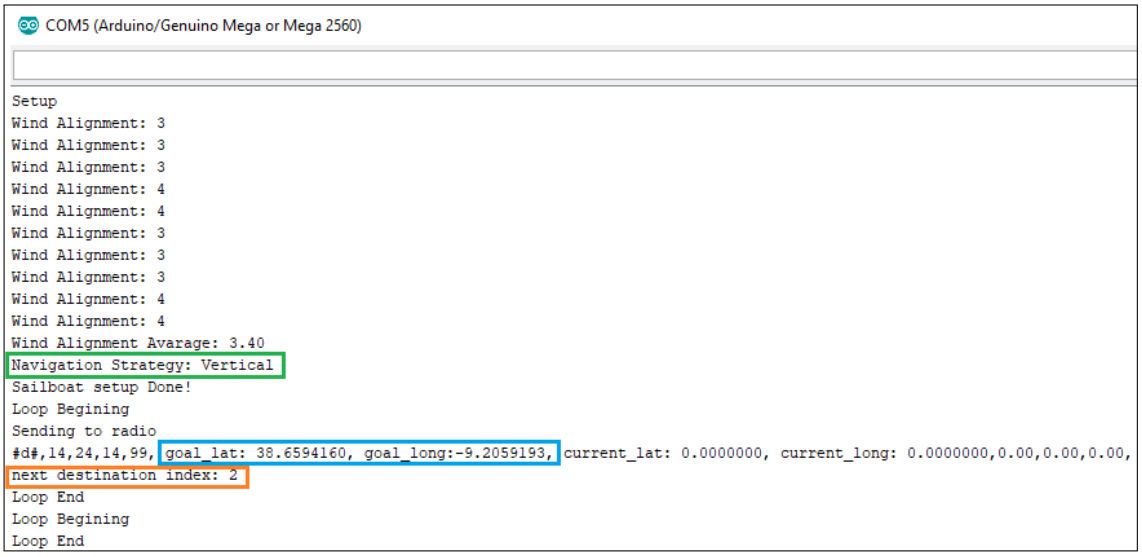

Figura 5.39: Dados do sistema no inicio da navegação. Profundidade de navegação *middle* e direção do vento de 3.40º.

Goals gerados:

Figura 5.40: *Goals* gerados associados ao *Vertical Middle Scanning*.

5.2.1.3 Profundidade de varrimento: *Deep*

Alterou-se a profundidade de varrimento para *deep* e repetiu-se o mesmo esquema de testes. Com o cata-vento orientado para bombordo, o comportamento do sistema foi o seguinte:

```
COM5 (Arduino/Genuino Mega or Mega 2560)

Setup
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 92
Wind Alignment: 90
Wind Alignment: 88
Wind Alignment: 86
Wind Alignment: 85
Wind Alignment: 83
Wind Alignment: 82
Wind Alignment: 82
Wind Alignment Avarage: 87.20
Navigation Strategy: Horizontal
Sailboat setup Done!
Loop Begining
Sending to radio
#d#,14,34,44,31, goal_lat: 38.6594160, goal_long:-9.2059193, current_lat: 0.0000000, current_long: 0.0000000,0.00,0.00,0.00,
next destination index: 2
Loop End
Loop Begining
Loop End
```

Figura 5.41: Dados do sistema no inicio da navegação. Profundidade de navegação *deep* e direção do vento de 87.20º.

Goals gerados:

Figura 5.42: *Goals* gerados associados ao *Horizontal Deep Scanning*.

Realizando a segunda experiência, com o cata-vento aproximadamente alinhado com a popa, os resultados foram:

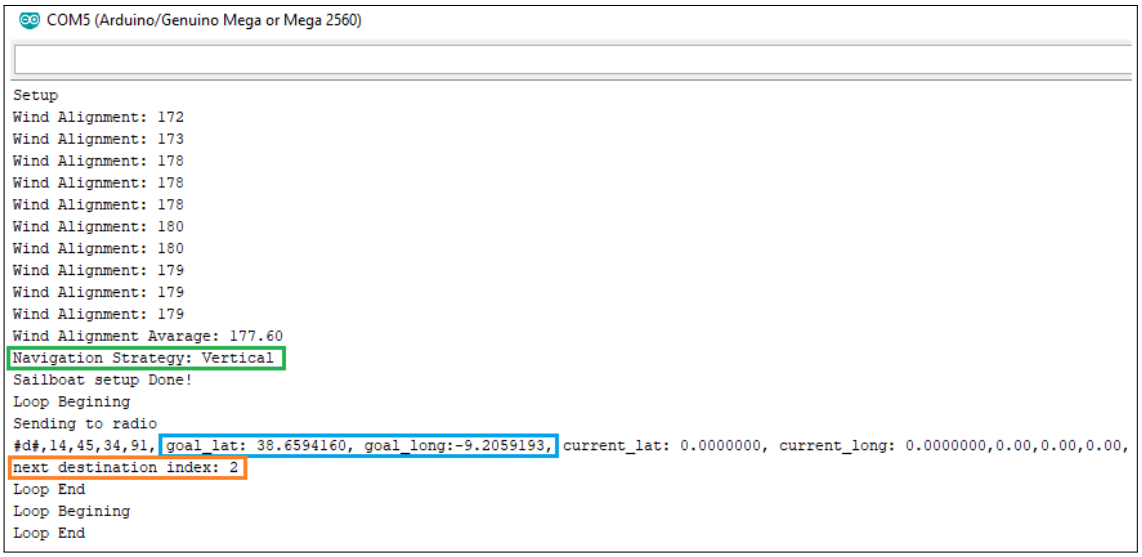

Figura 5.43: Dados do sistema no inicio da navegação. Profundidade de navegação *deep* e direção do vento de 177.60º.

Goals gerados:

Figura 5.44: *Goals* gerados associados ao *Vertical Deep Scanning*.

5.2.1.4 Profundidade de varrimento: *Full*

Por último, foram feitos dois testes para uma profundidade de varrimento *full*, obtendo os seguintes resultados no primeiro:

```
Setup
Wind Alignment: 80
Wind Alignment: 79
Wind Alignment: 79
Wind Alignment: 80
Wind Alignment: 80
Wind Alignment: 79
Wind Alignment: 79
Wind Alignment: 79
Wind Alignment: 79
Wind Alignment: 79
Wind Alignment Avarage: 79.30
Navigation Strategy: Horizontal
Sailboat setup Done!
Loop Begining
Sending to radio
#d#,14,32,55,79, goal_lat: 38.6594160, goal_long:-9.2059193, current_lat: 0.0000000, current_long: 0.0000000,0.00,0.00,0.00,
next destination index: 2
Loop End
Loop Begining
Loop End
```

Figura 5.45: Dados do sistema no inicio da navegação. Profundidade *full* e direção do vento de 79.30º.

Goals gerados:

Figura 5.46: *Goals* gerados associados ao *Horizontal Full Scanning*.

As figuras abaixo dizem respeito aos resultados obtidos relativamente à segunda experiência:

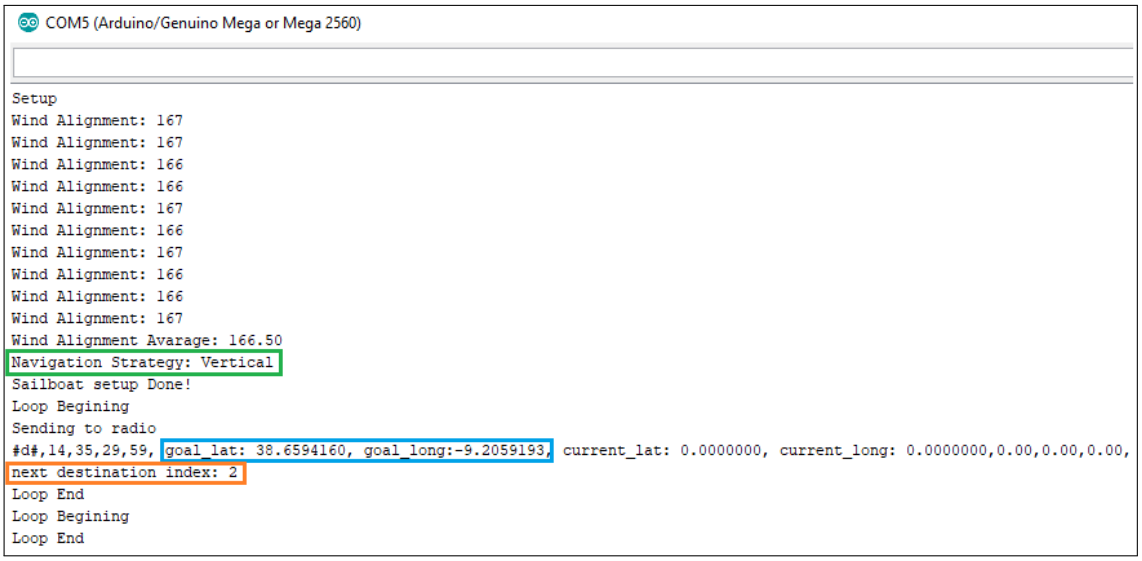

Figura 5.47: Dados do sistema no inicio da navegação. Profundidade de navegação *full* e direção do vento de 166.50º.

Goals gerados:

Figura 5.48: *Goals* gerados associados ao *Vertical Full Scanning*.

5.2.1.5 Análise aos resultados anteriores

Foram realizados vários testes, para os quatro tipos de profundidade de varrimento e para diferentes ângulos de experiência para experiência. Em todos os exemplos, o sistema calculou gerou corretamente os *goals* para cada método e para cada *input* externo (direção do vento).

É possível então concluir-se que os módulos dos tipos de varrimento e da decisão da estratégia de navegação foram integrados com sucesso no projeto.

5.2.2 Simulação de varrimentos

O passo seguinte foi testar a integração anterior com a deteção dos *goals*, simulando cenários de varrimento. Para tal, foram testados todos os métodos de varrimento, utilizando o kit de *hardware* para percorrer os *goals* gerados em cada situação, reproduzindo assim o trajeto que o veleiro deveria fazer numa situação real de prova.

Para auxílio dos testes, foi usada novamente a ferramenta desenvolvida em [29], mais especificamente as suas funcionalidades de apresentação em tempo real do comportamento do sistema e a exibição no mapa dos *goals* e da localização atual.

5.2.2.1 *Horizontal Fast Scanning*

O primeiro método simulado foi o *Horizontal Fast Scanning*. O sistema foi iniciado e realizou todos os cálculos necessários para começar o varrimento, tendo gerado os seguintes *goals*:

- Goal 0: (38.659415, -9.205919);

- Goal 1: (38.659143, -9.205398);
- Goal 2: (38.658985, -9.205886);

A figura 5.49 representa os dados do sistema no início da do varrimento:

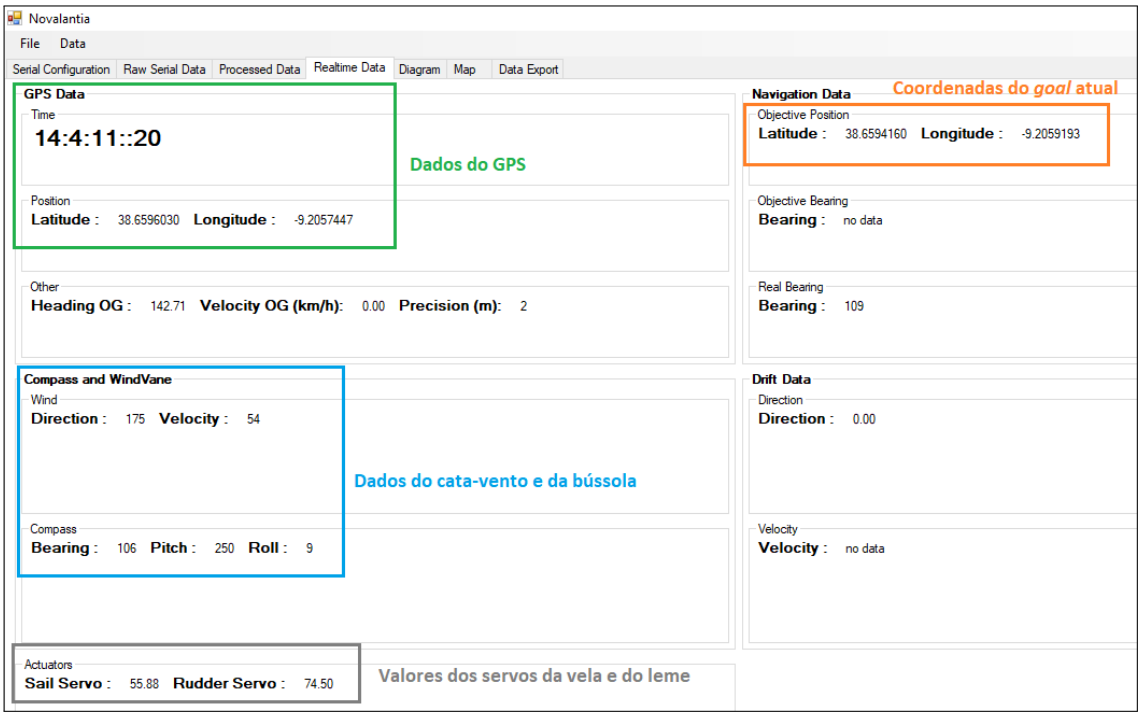

Figura 5.49: *Goal 0* do método *Horizontal Fast Scanning* e respetivos dados do sistema.

Como se encontra legendado na figura anterior, são os dados relativos ao GPS, ao cata-vento, à bússola, aos servos das velas e do leme, ao destino (*goal* atual), etc.

Nota: Dado o contexto do problema, os dados de maior interesse são os relativos ao GPS e as coordenadas do *goal*, pelo que daqui para a frente apenas esses parâmetros serão exibidos nas figuras respetivas.

O inicio do varrimento ocorreu às 14h04m (a ferramenta não apresenta o caractere ”0” antes da unidade dos minutos quando o valor é menor que 10), sendo que o primeiro destino (representado pelas coordenadas destacadas a laranja) corresponde às coordenadas do *goal 0*. De notar que as últimas casas decimais das coordenadas apresentadas na ferramenta são aparentemente diferentes das dos *goals* apresentados na lista anterior. Isto deve-se a simplesmente a fatores de arredondamentos efetuados pela ferramenta, ou seja, os *goals* apresentados são exatamente os mesmos que os gerados, mas a ferramenta foi formatada para exibir estes com mais uma casa decimal do que o suposto (ao invés de apenas acrescentar um ”0” na última casa decimal, a ferramenta atribui valores aleatórios a esta, desconhecendo-se a razão deste comportamento).

Após as configurações iniciais, o kit foi levado em direção às coordenadas do *goal 0*. O sistema, entretanto, detectou a sua chegada. A localização do momento de deteção do *goal 0* está representada na figura abaixo:

Figura 5.50: Deteção do *Goal 0* do método *Horizontal Fast Scanning*.

Foi possível ter percepção de que o sistema detetou a chegada ao *goal* devido à atualização das coordenadas de destino, com mostra a figura 5.51 (quando é detetada a chegada a um destino, as coordenadas são imediatamente atualizadas com o próximo *goal*, neste caso para o *goal 1*).

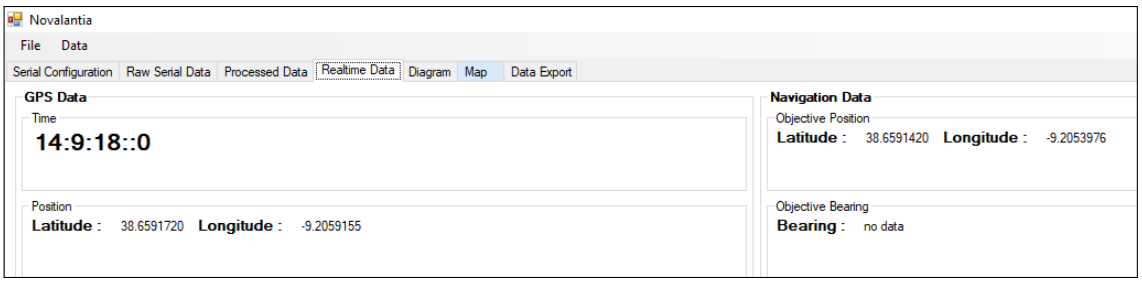

Figura 5.51: Dados do sistema no momento de chegada ao *goal 0* do método *Horizontal Fast Scanning*, com a inicialização das coordenadas do *goal 1*.

De seguida, o kit foi transportado para o *goal 1*, tendo sido detetada a sua chegada na localização abaixo ilustrada:

Figura 5.52: Deteção do *Goal 1* do método *Horizontal Fast Scanning*.

O instante de tempo em que o "veleiro" chegou ao *goal 1* e inicializou o *goal 2* encontra-se representado na imagem:

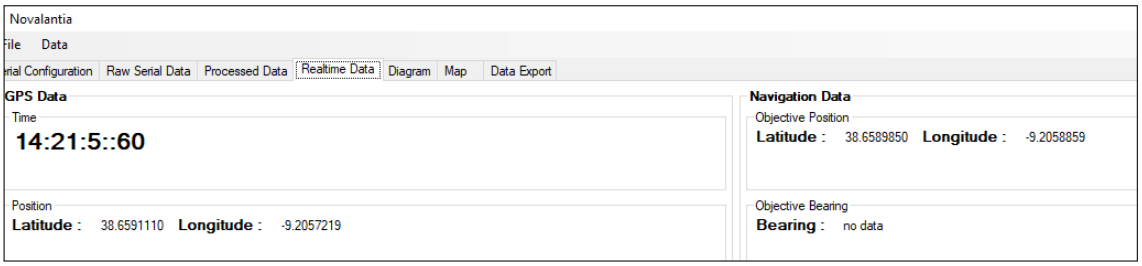

Figura 5.53: Dados do sistema no momento de chegada ao *goal 1* do método *Horizontal Fast Scanning*, com a inicialização das coordenadas do *goal 1*.

Por fim, a deteção do *goal 2* e final deu-se na seguinte localização:

Figura 5.54: Deteção do *Goal 2* do método *Horizontal Fast Scanning*.

No momento em que o sistema detetou a chegada ao último *goal*, o parâmetro de exibição do próximo destino foi atualizado com ”zeros”, representando o final do varrimento.

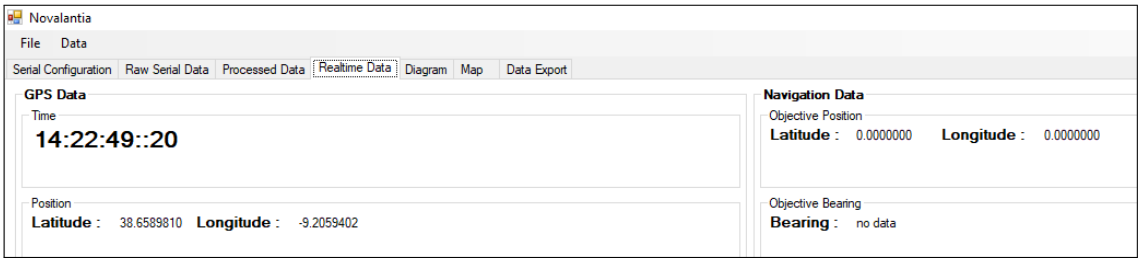

Figura 5.55: Dados do sistema no momento de chegada ao *goal 2* e respetivo fim do método *Horizontal Fast Scanning*.

Quando o varrimento terminou, o relógio do GPS apontava 14h22m e 49s, resultando num total de 18 minutos e 38 segundos de prova.

5.2.2.2 *Vertical Fast Scanning*

O segundo método simulado foi o *Vertical Fast Scanning*. As coordenadas dos *goals* gerados foram as seguintes (latitude, longitude):

- Goal 0: (38.659238, -9.205917);
- Goal 1: (38.659429, -9.205664);
- Goal 2: (38.658873, -9.205513);

- Goal 3: (38.658985, -9.205884);

O procedimento para esta simulação foi idêntico ao anterior: foram percorridos os *goals* por ordem e foram tiradas capturas de ecrã aos dados do sistema e à localização geográfica do "veleiro" nos momentos de deteção dos mesmos.

No inicio do varrimento os dados eram os seguintes:

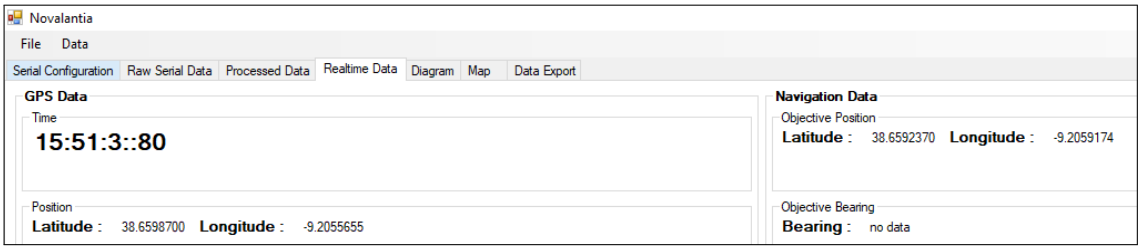

Figura 5.56: *Goal 0* do método *Vertical Fast Scanning* e respetivos dados do sistema.

No momento de deteção do *goal 0* a localização do sistema é a representada (pelo ícon azul) na figura 5.57, tendo sido inicializado o *goal 1* (figura 5.58):

Figura 5.57: Deteção do *Goal 0* do método *Vertical Fast Scanning*.

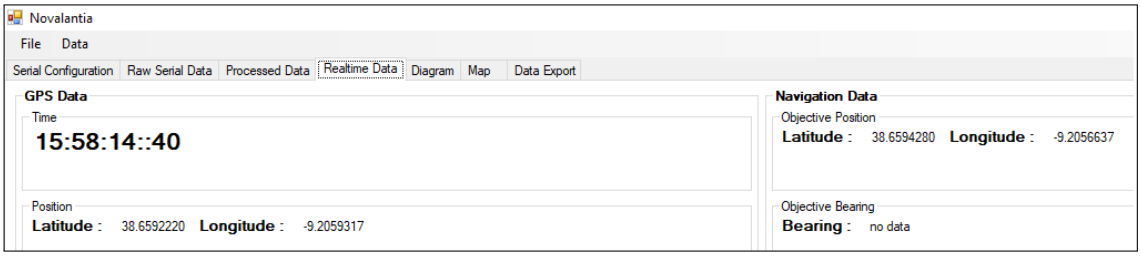

Figura 5.58: Dados do sistema no momento de chegada ao *goal 0* do método *Vertical Fast Scanning*, com a inicialização das coordenadas do *goal 1*.

Representação do momento de deteção do *goal 1* (figura 5.59) e inicialização do *goal 2* (figura 5.60):

Figura 5.59: Deteção do *Goal 1* do método *Vertical Fast Scanning*.

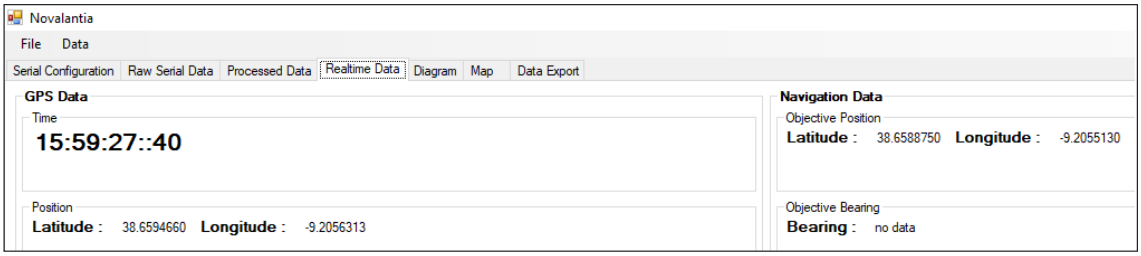

Figura 5.60: Dados do sistema no momento de chegada ao *goal 1* do método *Vertical Fast Scanning*, com a inicialização das coordenadas do *goal 2*.

Representação do momento de deteção do *goal 2* (figura 5.61) e inicialização do *goal 3* (figura 5.64):

Figura 5.61: Localização do momento de chegada ao *Goal 2* do método *Vertical Fast Scanning*.

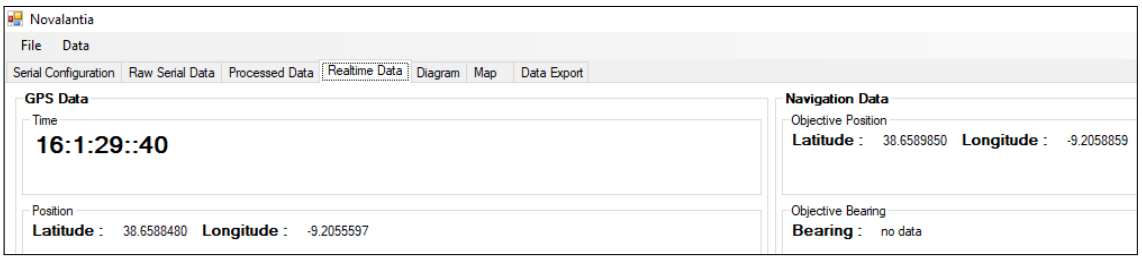

Figura 5.62: Dados do sistema no momento de chegada ao *goal 2* do método *Vertical Fast Scanning*, com a inicialização das coordenadas do *goal 3*.

Representação do momento de deteção do *goal 3* (último destino):

Figura 5.63: Deteção do *Goal 3* do método *Vertical Fast Scanning*.

No momento de chegada ao *goal 3*, o sistema assumiu o fim do varrimento:

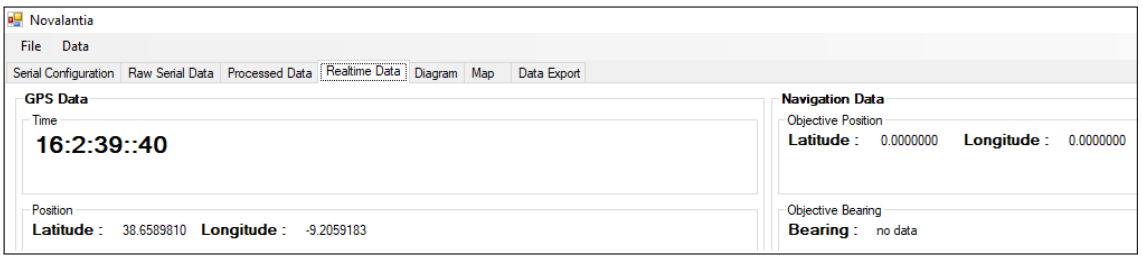

Figura 5.64: Dados do sistema no momento de chegada ao *goal 3* e respetivo fim do método *Vertical Fast Scanning*.

O tempo de prova total associado a este método foi de 11 minutos e 36 segundos.

5.2.2.3 *Horizontal Middle Scanning*

O terceiro teste consistiu na simulação do método *Horizontal Middle Scanning*. tendo sido geradas as seguintes coordenadas dos destinos:

- Goal 0: (38.659415, -9.205919);
- Goal 1: (38.659278, -9.205403);
- Goal 2: (38.659268, -9.205917);
- Goal 3: (38.659143, -9.205398);

- Goal 4: (38.658985, -9.205886);

Nota: Para evitar sobrecarregar o documento com um maior número de imagens (quanto maior a profundidade do varrimento, maior o número de *goals* gerados e portanto maior o número de validações), os *screenshots* da deteção/inicialização dos *goals* e da representação geográfica desses mesmos momentos foram agrupados.

A figura 5.65 representa a inicialização dos *goals* associados a este método:

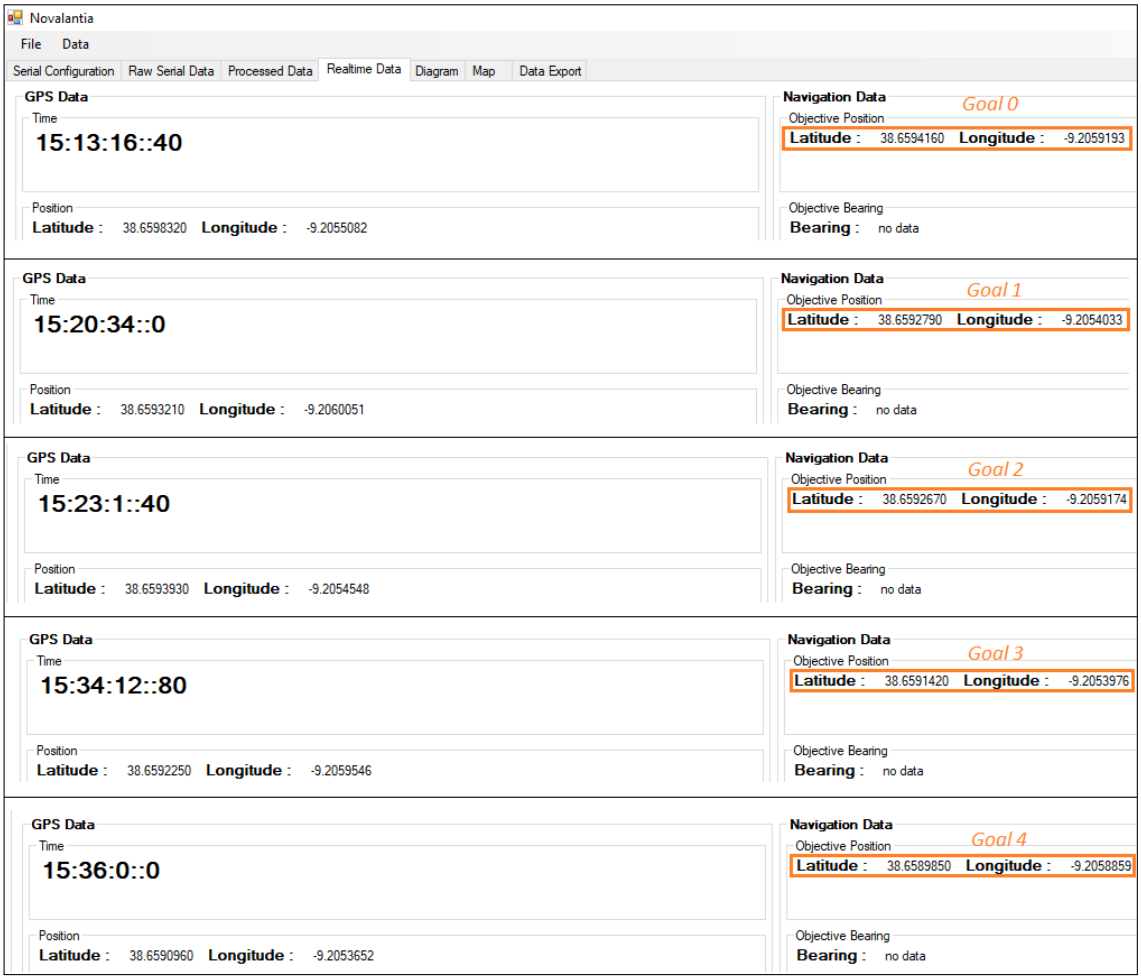

Figura 5.65: Inicialização/Deteção dos *goals* associados ao método *Horizontal Middle Scanning*.

No momento de cada inicialização/deteção, foi feita uma captura de ecrã de forma a ser possível a visualização da localização do "veleiro" nesse dado instante. Estes *screenshots* foram agrupados e apresentados na figura abaixo:

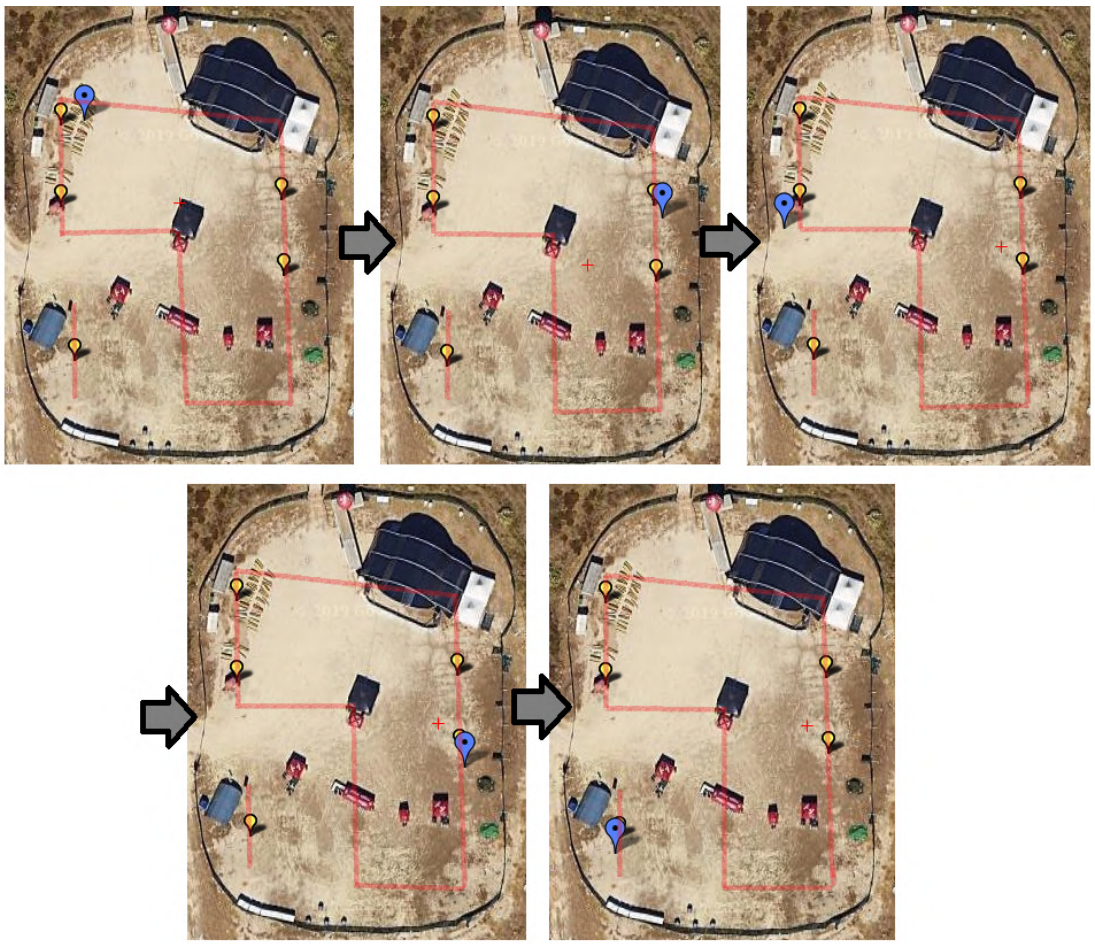

Figura 5.66: Momentos de chegada aos *goals* (0 a 4) associados ao método *Horizontal Middle Scanning*.

À chegada ao último *goal*, foi detetado o fim do varrimento:

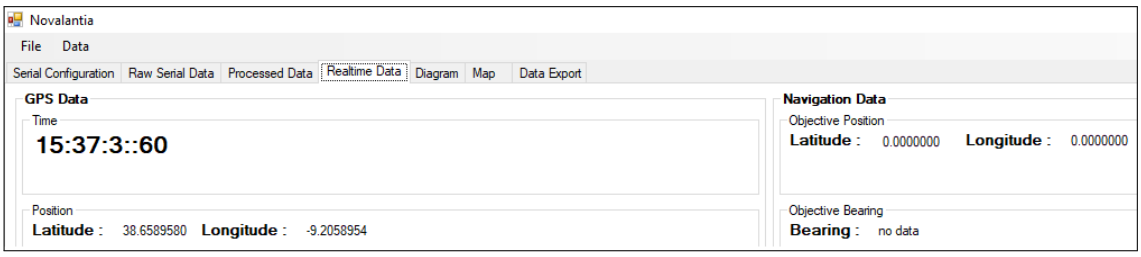

Figura 5.67: Dados do sistema no momento de chegada ao *goal 4* e respetivo fim do método *Horizontal Middle Scanning*.

O inicio da simulação deu-se às 15h13m e 16s e o fim às 15h37m e 03s, totalizando 23 minutos e 47 segundos de tempo de prova.

5.2.2.4 *Vertical Middle Scanning*

Prosseguiram-se os testes, desta feita com o *Vertical Middle Scanning*.

Os *goals* calculados foram os seguintes:

- Goal 0: (38.659415, -9.205919);
- Goal 1: (38.659210, -9.205779);
- Goal 2: (38.659429, -9.205664);
- Goal 3: (38.658873, -9.205513);
- Goal 4: (38.658985, -9.205886);

Inicialização dos *goals*:

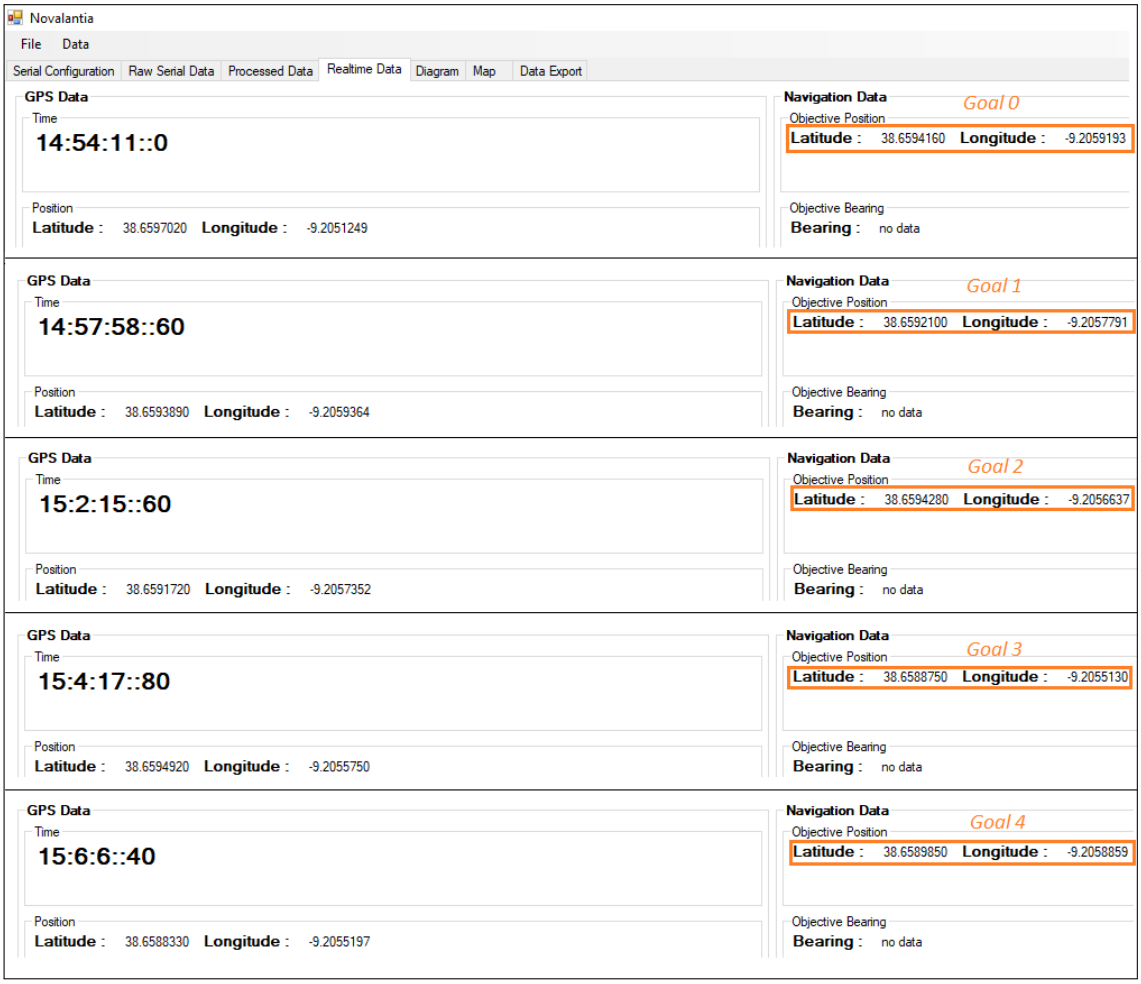

Figura 5.68: Inicialização/Deteção dos *goals* associados ao método *Vertical Middle Scanning*.

Localização do sistema no momento de deteção de chegada aos destinos:

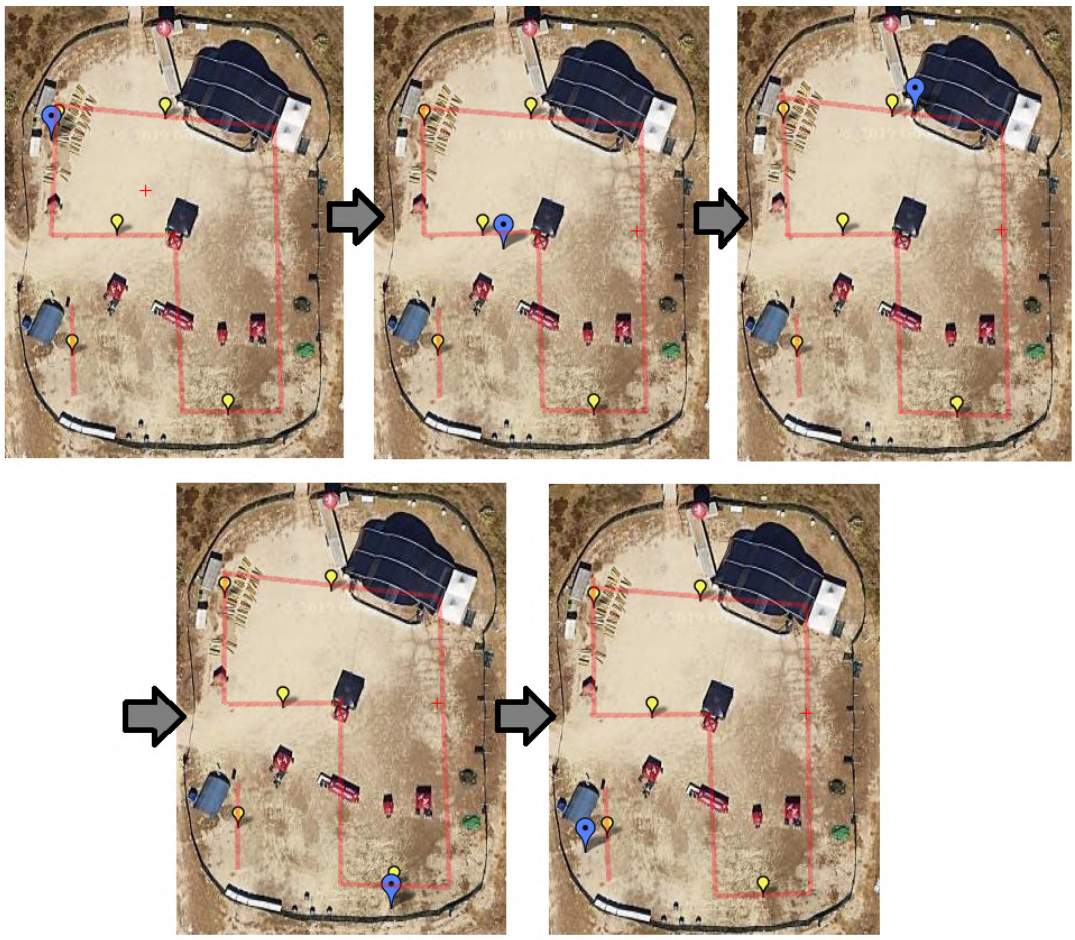

Figura 5.69: Momentos de chegada aos *goals* (0 a 4) associados ao método *Vertical Middle Scanning*.

Fim do varrimento:

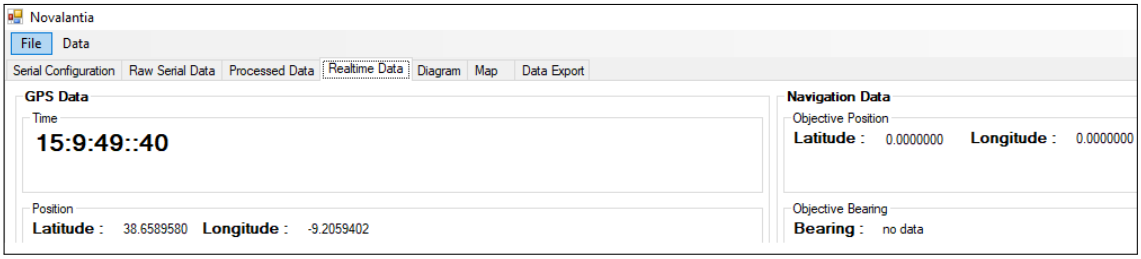

Figura 5.70: Dados do sistema no momento de chegada ao *goal 4* e respetivo fim do método *Vertical Middle Scanning*.

Tempo total de simulação: 15 minutos e 38 segundos.

5.2.2.5 *Horizontal Deep Scanning*

Para simular o varrimento do método *Horizontal Deep Scanning* foram usadas diferentes coordenadas de bóias, em relação aos testes anteriores (variando assim mais um fator

de teste, de forma a verificar se o sistema se comporta de forma idêntica). As coordenadas introduzidas foram as seguintes:

- Bóia 1: (38.661143, -9.206117);
- Bóia 2: (38.660810, -9.206107);
- Bóia 3: (38.660802, -9.207096);
- Bóia 4: (38.661535, -9.207094);
- Bóia 5: (38.661553, -9.206578);
- Bóia 6: (38.661149, -9.206569);
- Bóia 7: (38.661393, -9.206167);
- Bóia 8: (38.661548, -9.206156);

Na figura 5.71 encontram-se representadas no mapa as coordenadas referidas acima.

Figura 5.71: Representação geográfica das coordenadas das bóias e respetiva área de varrimento.

Os *goals* gerados associados a este método têm as seguintes coordenadas:

- Goal 0: (38.660852, -9.206108);
- Goal 1: (38.660985, -9.207095);
- Goal 2: (38.661060, -9.206115);
- Goal 3: (38.661169, -9.207095);
- Goal 4: (38.661250, -9.206571);
- Goal 5: (38.661352, -9.207095);
- Goal 6: (38.661471, -9.206162);

Inicialização/Deteção dos *goals*:

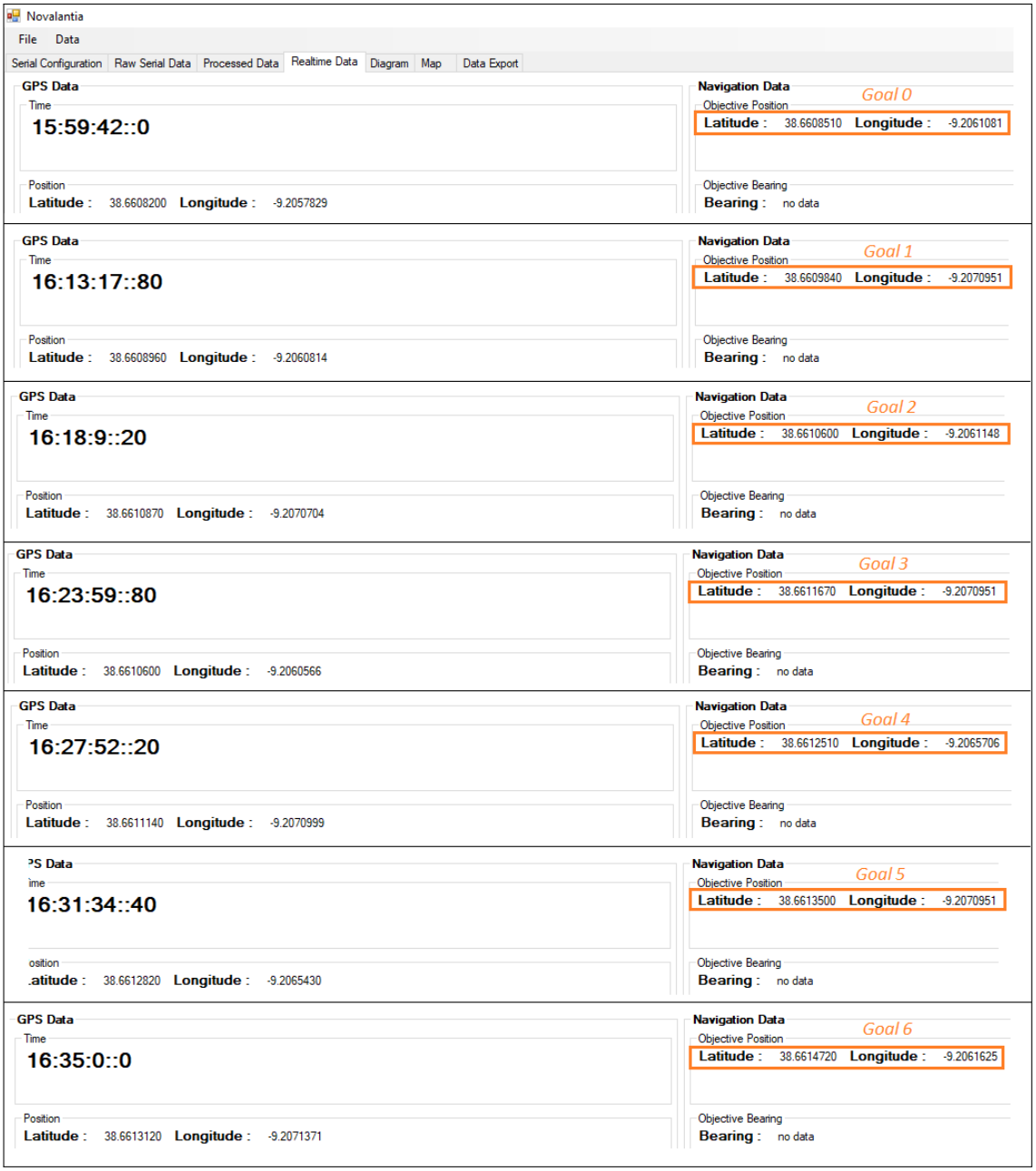

Figura 5.72: Inicialização/Deteção dos *goals* associados ao método *Horizontal Deep Scanning*.

Nas subimagens abaixo encontram-se as representações geográficas (ícones azuis) nos instantes de deteção dos *goals* (ícones amarelos):

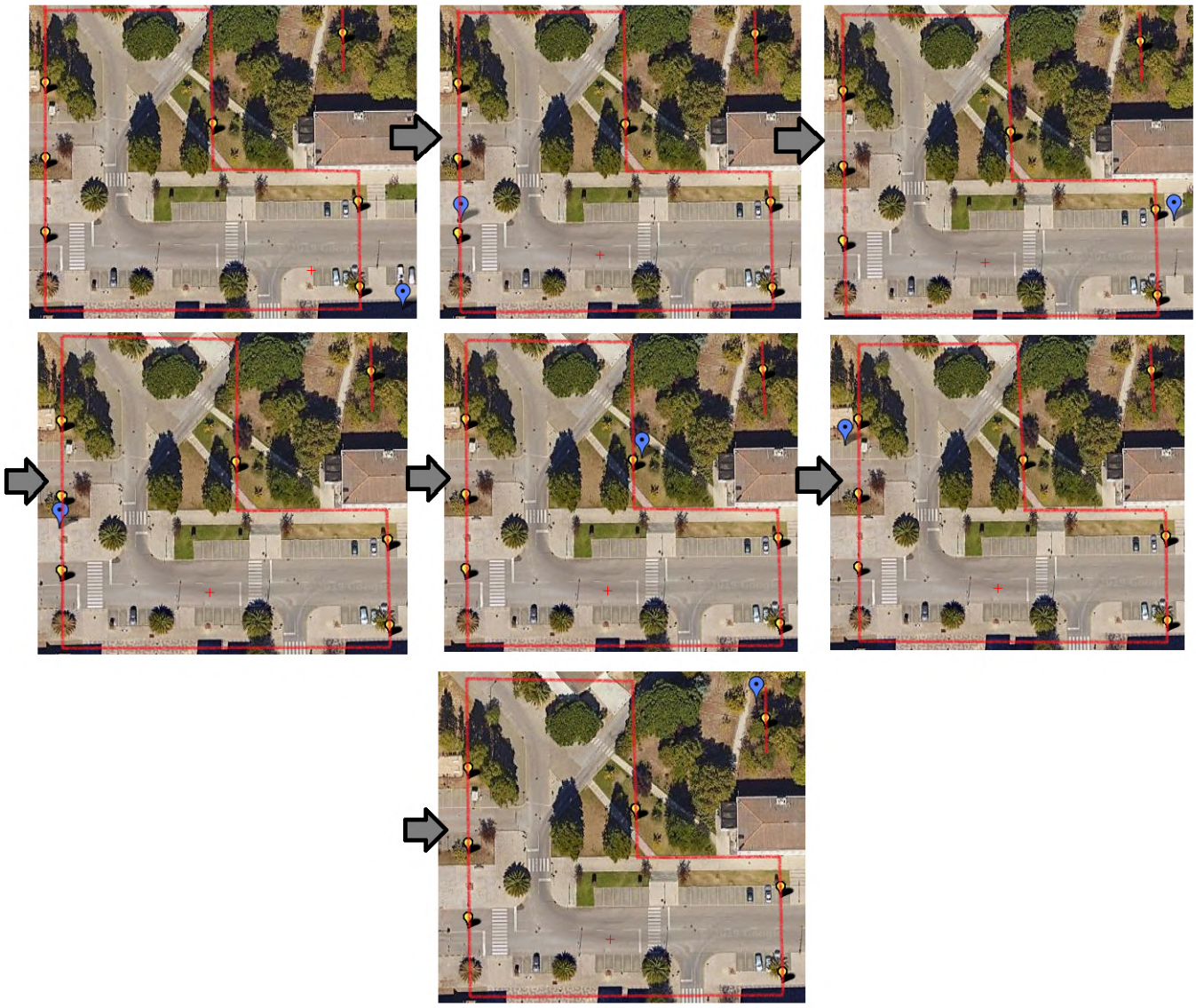

Figura 5.73: Localização dos momentos de chegada aos *goals* (0 a 6) associados ao método *Horizontal Deep Scanning*.

Deteção do fim do varrimento:

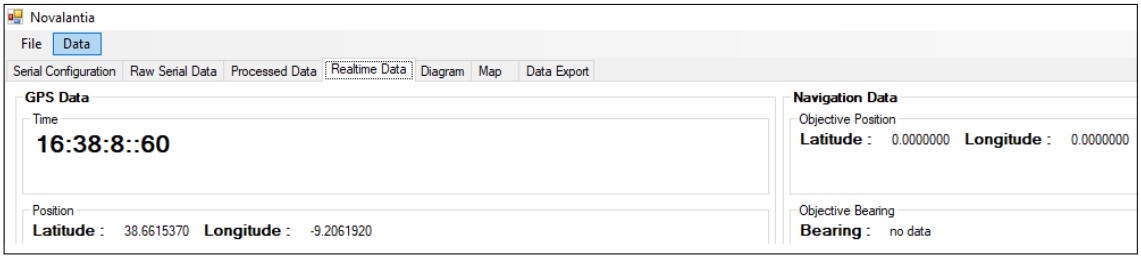

Figura 5.74: Dados do sistema no momento de chegada ao *goal 6* e respetivo fim do método *Horizontal Deep Scanning*.

Tempo total de varrimento: 38 minutos e 26 segundos.

5.2.2.6 *Vertical Deep Scanning*

As coordenadas das bóias definidas para este teste foram as mesmas usadas para as simulações dos métodos *Horizontal/Vertical Fast Scanning* e *Horizontal/Vertical Middle Scanning* (ou seja, apenas para a simulação do método *Horizontal Deep Scanning* a definição das coordenadas das bóias foi diferente).

A figura abaixo representa as capturas de ecrã dos dados do sistema, nos momentos de deteção de chegada a cada destino:

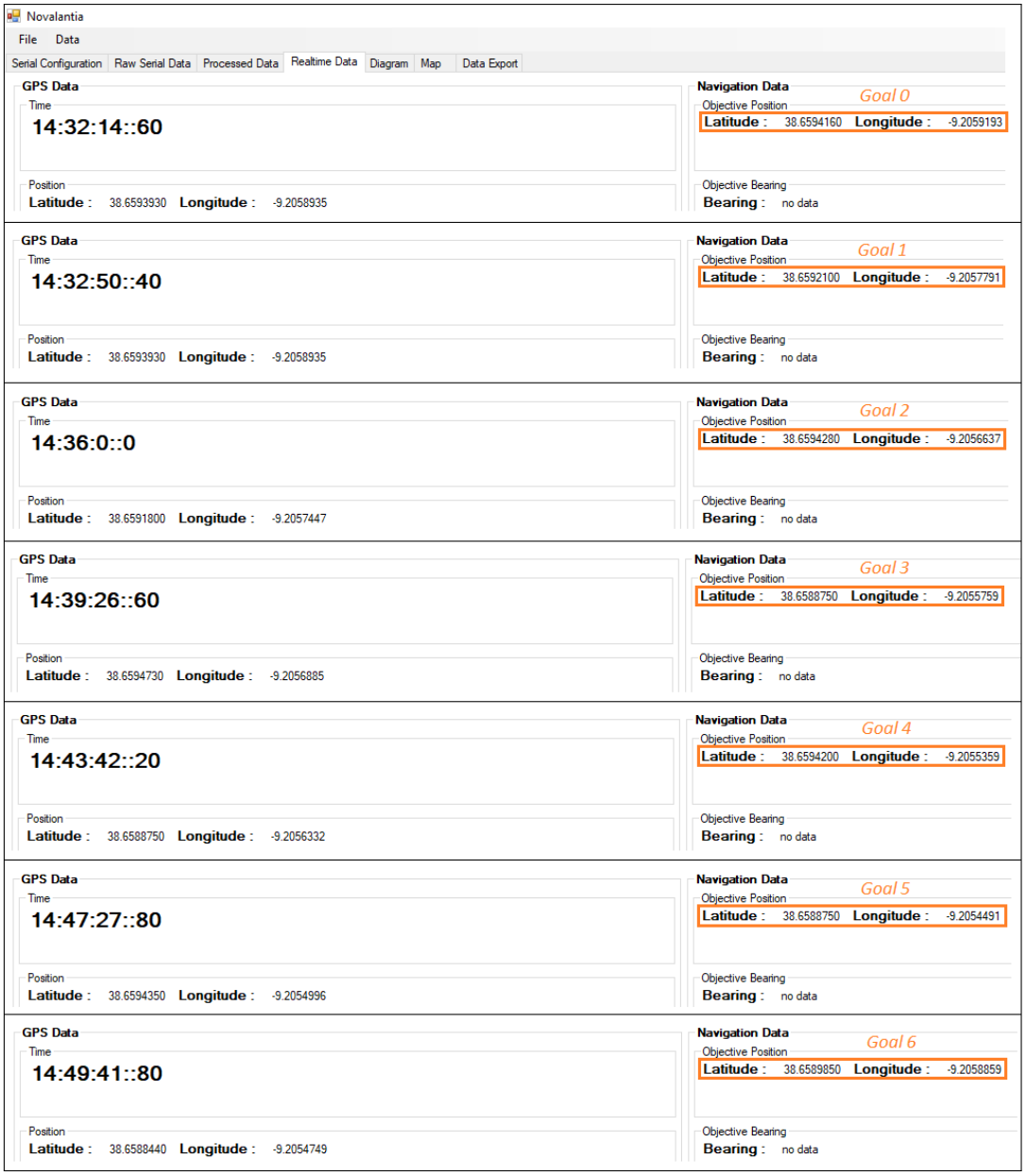

Figura 5.75: Inicialização/Deteção dos *goals* associados ao método *Vertical Deep Scanning*.

Para cada *goal* detetado na figura anterior, foi também tirado um *screenshot* da localização do "veleiro" no instante em que ocorreram:

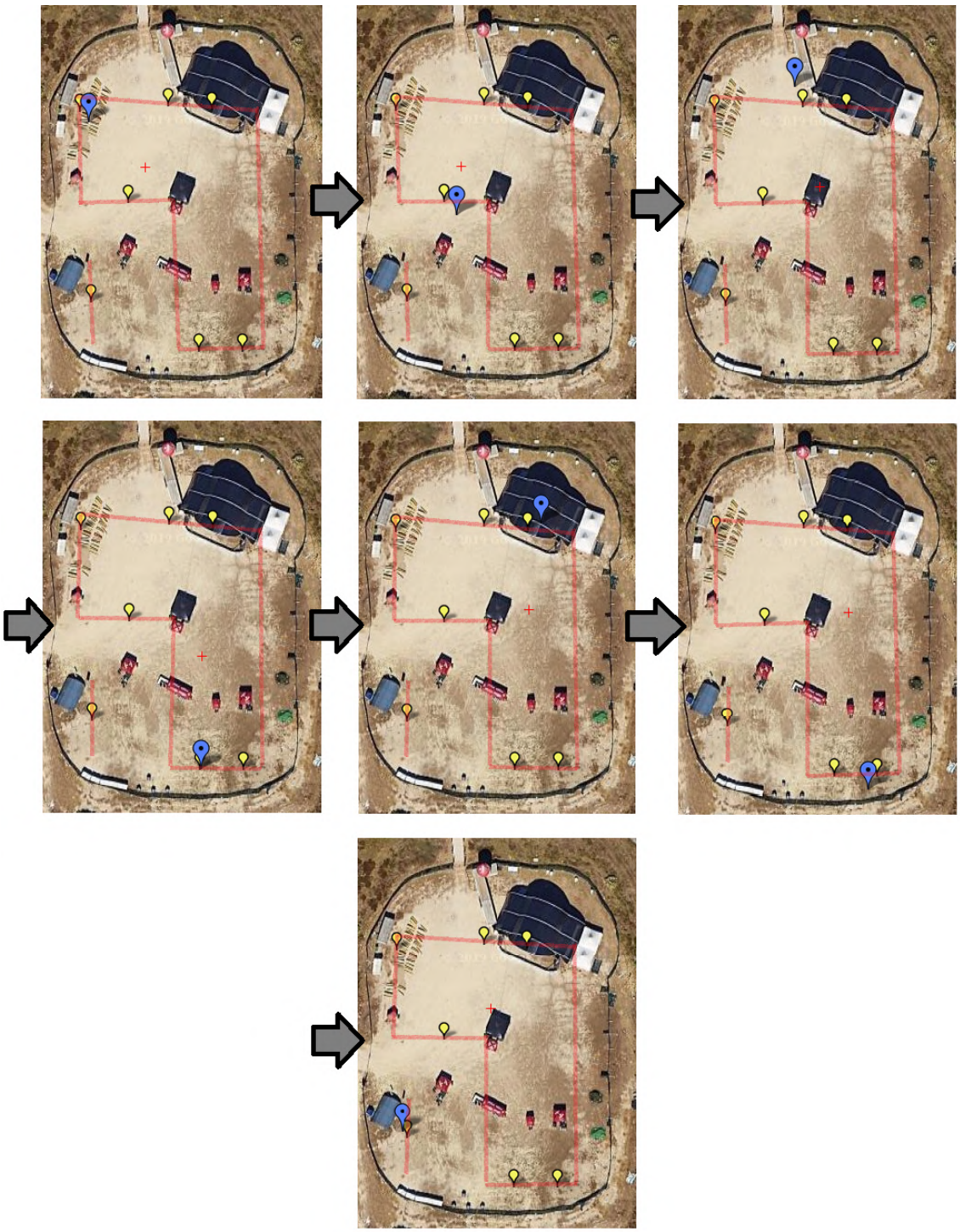

Figura 5.76: Momentos de chegada aos *goals* (0 a 6) associados ao método *Vertical Deep Scanning*.

À chegada ao último *goal*, o sistema deu por terminado o varrimento:

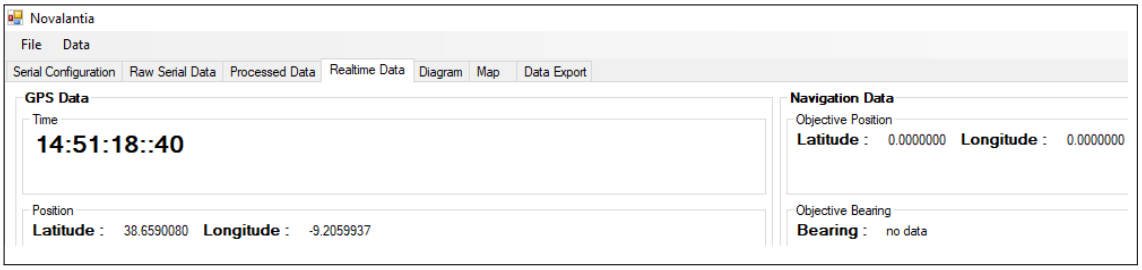

Figura 5.77: Dados do sistema no momento de chegada ao *goal 6* e respetivo fim do método *Vertical Deep Scanning*.

No total o varrimento teve uma duração de 19 minutos e 4 segundos.

5.2.2.7 *Horizontal/Vertical Full Scanning*

Infelizmente não foi possível completar os testes de simulação relativos ao *Horizontal Full Scanning* e ao *Vertical Full Scanning*, devido a diversos fatores, que serão explicados na secção seguinte, na análise de resultados destas simulações.

5.2.2.8 Análise aos resultados anteriores

No final de todas (as possíveis) simulações procedeu-se à análise dos resultados de cada teste.

Ao todo, dos 8 métodos de varrimento 6 foram testados com sucesso, não tendo sido possível completar os dois últimos (*Horizontal/Vertical Full Scanning*). Contudo, os 6 métodos simulados com sucesso tiveram também bastantes dificuldades associadas, sendo mesmo necessário a repetição (mais do que uma vez) de grande parte dos testes.

Tendo em conta que estas simulações foram feitas a caminhar (a passo normal) e numa área não superior a 100 metros quadrados, os tempos de prova deveriam ser bastante mais baixos do que os obtidos, sendo isto um fator revelador de ocorrência de problemas nestes testes.

Para um melhor entendimento das dificuldades inerentes às simulações, a figura 5.78 apresenta um gráfico de comparação da duração do sistema a detetar cada um dos *goals*, entre os métodos.

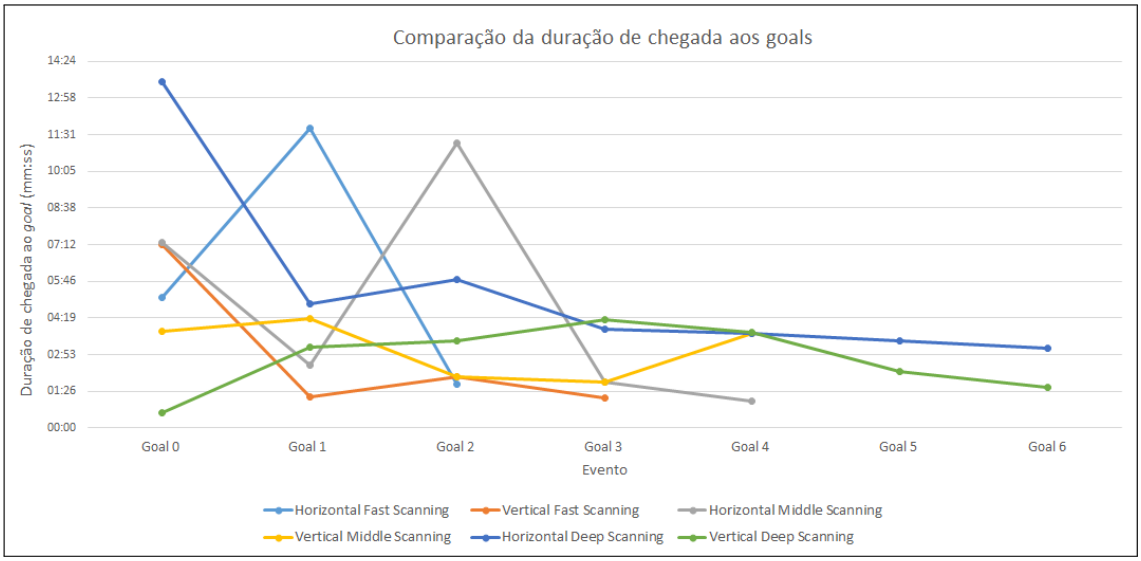

Figura 5.78: Comparação dos tempos de deteção de cada *goal*, para cada método.

Observando o gráfico, pode ser confuso tirar ilações à primeira vista. E pode ser confuso precisamente devido à discrepância de valores de duração entre cada *goal*, para o mesmo método.

Numa situação ideal, cada uma das linhas do gráfico (que representa cada método) teria um comportamento linear, onde a duração de chegada seria mais ou menos uniforme de *goal* para *goal*. No entanto não foi isso que se verificou. Pelo contrário, os valores variaram de forma bastante diferente.

Veja-se por exemplo a linha correspondente ao *Horizontal Middle Scanning* (cinzenta): O sistema demorou 7 minutos e 18 segundos a "chegar" ao *goal* 0, 2 minutos e 27 segundos para o *goal* 1, 11 minutos e 11 segundos para o *goal* 2, 1 minuto e 48 segundos para o *goal* 3, e 1 minuto e 3 segundos para o *goal* 4. Ora estes valores diferem bastante entre si, não seguindo nenhuma linearidade e expondo as tais dificuldades/problemas encontrados nas simulações.

Nas simulações feitas, a distância entre os *goals* era de 30 a 60 metros, pelo que a duração de chegada a um novo *goal* (a andar, transportando o kit e o computador) não deveria ultrapassar os 90 segundos segundos. Nos dados recolhidos, houve tempos que ultrapassaram os 2, 5 e até 10 minutos.

A principal razão para a ocorrência destes tempos fora do normal prende-se com o facto de o sensor GPS ter perdido sinal com muita frequência. Em muitas situações, o GPS deixava de atualizar a sua posição, mesmo que o "veleiro" (neste caso o kit) continuasse a progredir, fazendo com que o sistema assumisse que se encontrava parado. Portanto, houve várias ocasiões em que o sistema físico chegou ao destino, mas como o GPS perdeu o sinal entretanto, não era assumido que o destino tinha sido alcançado, sendo depois necessário percorrer várias posições e esperar vários minutos até que o GPS recuperasse o sinal.

Esta perda de sinal do GPS pode ter várias causas:

- **Defeito do próprio GPS:** Pode ter havido a possibilidade de o GPS possuir alguma anomalia ou já não estar nas melhores condições, influenciando diretamente a frequência de atualização da sua localização;
- **Ambiente externo:** Um factor que influencia a leitura de dados do GPS é o meio externo. Em ambientes de céu aberto, sem obstáculos a "tapar" a comunicação do GPS com os satélites, o sinal GPS é mais forte e portanto obtém-se uma melhor leitura. Por outro lado, em locais que dificultam a chegada do sinal ao GPS, como locais com muitas árvores ou prédios altos, a leitura pode ser defeituosa. Este facto foi averiguado, principalmente na simulação do método *Horizontal Deep Scanning*. Ao contrário das outras (realizadas num descampado no campus da FCT-UNL), esta simulação foi feita num local diferente: entre os edifícios da FCT-UNL (que também é composto por algumas árvores), tendo sido precisamente esta simulação aquela que maior tempo demorou a ser completada (38 minutos e 26 segundos);
- **Condições atmosféricas:** O sinal de GPS também é afetado pelo estado da atmosfera, pois em situações de céu nublado ou com nuvens densas a comunicação entre o GPS e os satélites pode ser bastante diminuida (dependendo obviamente da qualidade e precisão do sensor). Nas alturas de realização destes testes, as condições atmosférias eram adversas, com o céu bastante nublado e tendo inclusivamente ocorrido precipitação, que levou ao abortamento de simulações que estavam em execução;
- **Interferência externa:** Pode ocorrer situações em que o sinal de GPS é afetado ou bloqueado por outros dispositivos. Os aviões, por exemplo, podem interferir com o sinal GPS, pois muitos destes são constituídos por um material que bloqueia este sinal (propsitadamente, para evitar interferências com o controlo do avião). Este fator também foi averiguado, pois aviões sobrevoam com muita frequência a área no momento das simulações (por ficar bastante próximo de Lisboa), e sempre que se sucedeu, o GPS perdeu o sinal.

Algumas (ou todas) destas razões originaram as tais dificuldades mencionadas anteriormente. Conjugando estas dificuldades às condições atmosféricas adversas (com muitos períodos de chuva na fase de validação desta dissertação) e aos problemas técnicos (o computador usado possuía uma fraca autonomia, pelo que varrimentos mais demorados tiveram de ser abortados a meio) resultaram em várias tentativas falhadas de simulações, o que tornou inviável a obtenção de simulações bem sucedidas para os métodos mais extensos e complexos, o *Horizontal Full Scanning* e o *Vertical Full Scanning* (como já tinha sido referido anteriormente), ao passo que para os outros métodos foi, felizmente, possível obter testes completos.

No entanto, mesmo não tendo sido possível obter simulações com sucesso para os dois métodos anteriores, os problemas e as suas causas foram identificados, chegando-se à conclusão que nada têm a ver com a integração dos módulos desenvolvidos.

Desta forma, é possível afirmar que os módulos foram integrados com sucesso no projeto.

5.2.3 Integração do *Smart Scanning*

A última etapa deste tipo de validações foi testar a integração do mecanismo **Smart Scanning** no projeto.

Para tal, foram realizados duas experiências, ambas com a profundidade de varrimento predefinida para *deep*. A única variante entre os dois testes foi o ângulo do catavento, de forma a testar a mesma profundidade de varrimento para os dois tipos de estratégia de navegação (horizontal e vertical).

Para auxiliar a percepção da execução deste mecanismo, foram inseridos *"prints"* no código para fazer *debug* do sistema.

Antes do inicio dos testes, foram definidos os tempos associados aos *deadlines*. Os valores definidos foram:

- *Deadline 1*: 3 minutos;
- *Deadline 2*: 7 minutos;

Às *deadlines* foram propositadamente definidos valores baixos, pois o objetivo foi apenas testar a eficácia deste mecanismo. Em situações de provas reais, estes valores devem ser ajustados (recomenda-se usar 15 minutos para a *deadline 1* e 20 minutos para a *deadline 2*) consoante o tempo limite.

Primeira experiência

Para o primeiro teste, o método definido pelo sistema foi o *Horizontal Fast Scanning*, tendo sido as coordenadas de bóias introduzidas iguais às que foram usadas no teste de simulação do varrimento deste mesmo método (na secção 5.2.2.5).

Após o inicio do teste, o sistema guardou o tempo de começo de prova (calculado no *setup*), como mostra a figura seguinte:

```
Setup
Start Time: 16:55;
In minutes: 1015;
Sailboat setup Done!
```

Figura 5.79: Tempo de inicio do teste - *Horizontal Deep Scanning*.

Além disso, os *goals* associados ao método foram calculados e guardados no vetor *route_race[]*. No momento inicial da navegação, o índice de destino apontava para a posição 2 do vetor (correspondente ao *goal* 0):

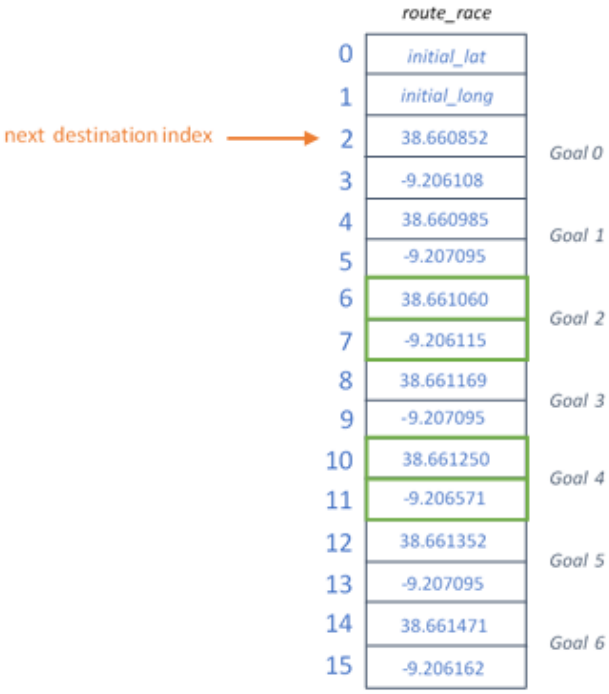

Figura 5.80: *Goals* gerados no *route_race[]* e respetivo índice de destino.

A figura seguinte representa um *screenshot* do monitor série do Arduino, para perceber as variáveis do sistema no inicio da navegação:

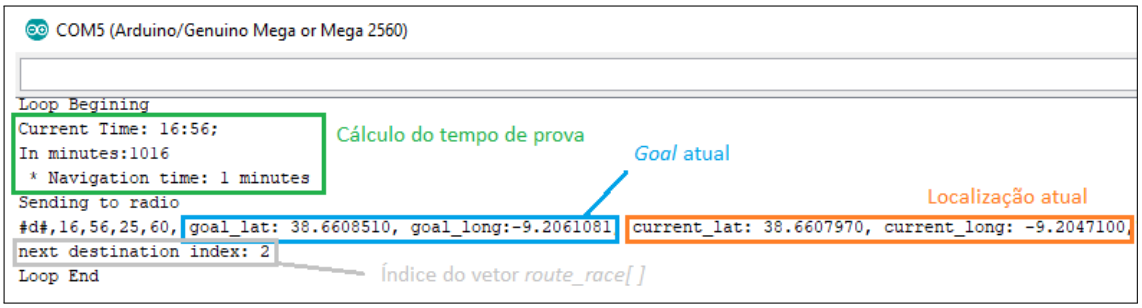

Figura 5.81: *Debug* ao comportamento do sistema no inicio da teste.

Após o inicio do teste, aguardou-se (na mesma posição) que o tempo de prova excede-se a *deadline* 1 (3 minutos). No instante em que isso se sucedeu, foi feita uma captura de ecrã do monitor série do Arduino, onde é possível compreender o comportamento do sistema:

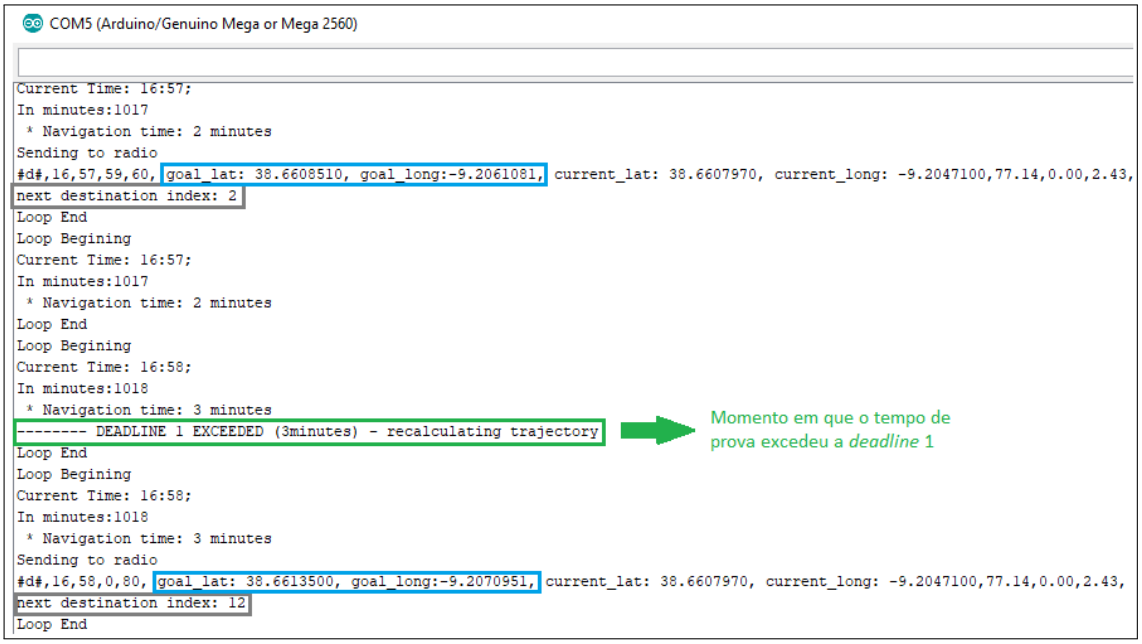

Figura 5.82: Momento em que o tempo de prova excedeu a primeira *deadline*.

Como se pode ver na figura acima, após a *deadline* 1 ter sido excedida, o índice do vetor (destacado a cinzento) *route_race[]* foi atualizado com o valor 12, ficando a apontar para o *goal* 5. A azul são destacadas as coordenadas do próximo *goal*, que também foram atualizadas.

Segunda experiência

A segunda experiência teve inicio no instante de tempo:

```
Setup
Start Time: 16:45;
In minutes:1005
Sailboat setup Done!
```

Figura 5.83: Tempo de inicio do teste - *Vertical Deep Scanning*.

Os *goals* gerados foram os seguintes:

- Goal 0: (38.659415, -9.205919);
- Goal 1: (38.659210, -9.205779);
- Goal 2: (38.659429, -9.205664);
- Goal 3: (38.658873, -9.205576);
- Goal 4: (38.659421, -9.205536);
- Goal 5: (38.658874, -9.205449);

- Goal 6: (38.658985, -9.205886);

Nesta experiência o objetivo foi validar a segunda *deadline*. Para tal, após o início do teste, foram percorridos todos os *goals* até ao *checkpoint* 1 (caso contrário, seria disparada a *deadline* 1), ou seja, o *goal* 2.

Após isto, aguardou-se que o tempo de prova ultrapassasse o tempo da *deadline* 2. A figura 5.84 representa o instante tempo que se verificou este acontecimento.

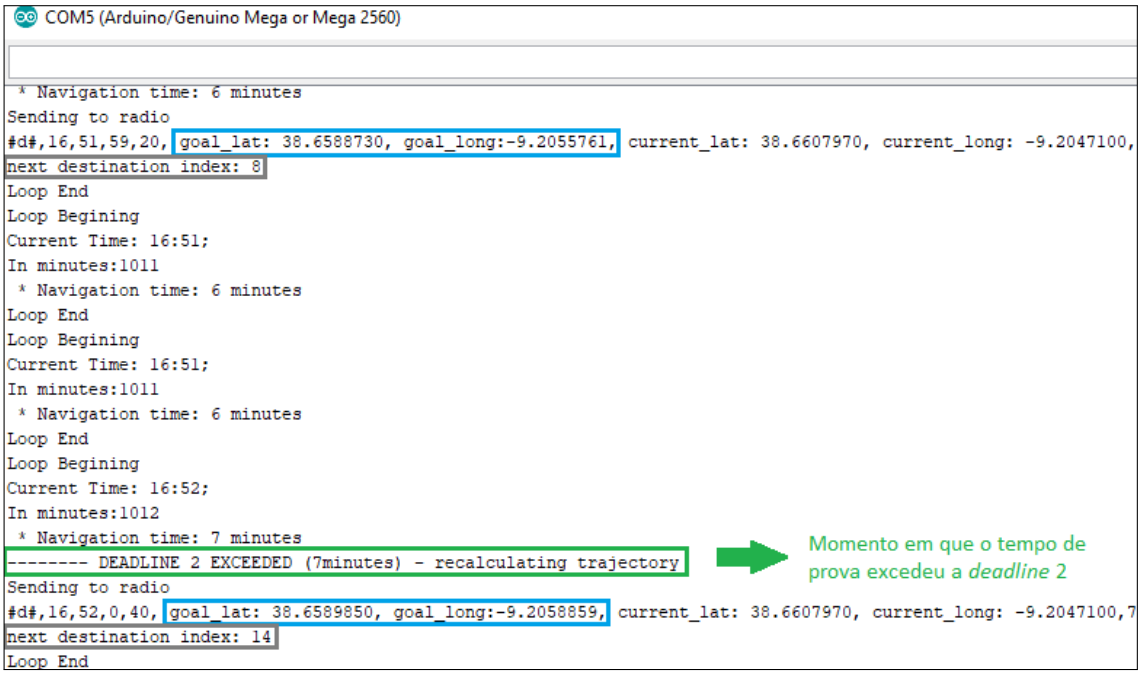

Figura 5.84: Momento em que o tempo de prova excedeu a segunda *deadline*.

Antes de disparar a *deadline* 2, o sistema tinha como objetivo o *goal* 3 (índice 8). Após o tempo de prova exceder este tempo limite, o índice do vetor *route_race[]* foi atualizado para o valor 14, que aponta para o *goal* 6, cujas coordenadas se encontram representadas no segundo retângulo azul.

5.2.3.1 Análise aos resultados anteriores

A realização destes testes permitiram concluir que a integração do mecanismo *Smart Scanning* foi bem sucedida.

Os dois testes efetuados visaram a validação deste módulo, fazendo variar a estratégia de navegação e a *deadline* excedida.

Na primeira situação, o sistema tinha como objetivo o *goal* 0, e ao exceder este tempo (sem ter atravessado o *checkpoint* 1), o trajeto foi recalculado, passando a assumir o *goal* 5 como próximo destino.

Na segunda situação, o sistema percorreu todos os *goals* até ao *checkpoint* 1, antes de exceder a primeira *deadline* (deixando esta sem efeito). No entanto, ao não conseguir

atravessar o *checkpoint* 2 a tempo, o trajeto foi também recalculado, inicializando o *goal* 6 como destino.

O comportamento do sistema em ambos os testes foi exatamente aquele que foi descrito no capítulo da apresentação da solução proposta, correspondendo assim às expetativas.

5.3 Evento REX

Como foi referido anteriormente, a Marinha Portuguesa organizou um evento na Base Naval de Lisboa, o *Robotics EXercise 2019*, com o objetivo de facilitar a realização de provas marítimas e testes de sistemas robóticos neste ambiente, tendo sido a Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa convidada a participar neste evento.

O evento teve uma duração de 5 dias, sendo que apenas foi possível testar o veleiro na água por 3 ocasiões, tendo sido o resto do tempo usado para fazer configurações e corrigir certos aspetos no código.

Para os testes aquáticos, foi necessário equipar o veleiro com o respetivo *hardware*: conectou-se o cata-vento no topo do mastro; o GPS ficou localizado ligeiramente mais abaixo do cata-vento (quanto mais alto estiver o sensor, melhor o sinal); e os restantes equipamentos (placa Arduino, bússola e *powerbank*) colocaram-se dentro da caixa acrílica.

O objetivo do primeiro teste foi simples, atribuir três coordenadas de pontos para este percorrer, num género de *regatta*, para perceber como se comporta o veleiro numa situação de navegação real.

Figura 5.85: REX - primeira tentativa.

Contudo, este primeiro teste não correu como o esperado. Após o seu inicio, o veleiro navegou sempre na mesma direção, sem mostrar alguma reação, quer na atuação das velas quer na atuação do leme. Após alguns minutos, este foi retirado da água, sendo que depois de algumas averiguações, verificou-se que havia fios mal conectados, levando a que o sistema se encontrasse desligado no momento da navegação, explicando assim o comportamento do veleiro.

Na segunda tentativa, fez-se ajustes para evitar que o problema anterior se voltasse a suceder. Desta forma, atribuíram-se as mesmas coordenadas de destino que na tentativa anterior.

No momento que se colocou o veleiro na água, confirmou-se de que o sistema se encontrava ativo, pois foi visível o trabalho dos servos das velas e do leme, com movimentos de caçar/folgar as velas e mudanças de direção (respetivamente).

No entanto, o veleiro não se comportou da maneira esperada. Ao invés de navegar na direção do objetivo, este realizou manobras completamente desenquadradas do o esperado, trocando várias vezes o sentido de navegação. Depois de retirado na água, reparou-se que os fios usados pelo servo para caçar/folgar as velas estavam completamente desfasados um do outro. Isto porque um dos fios ficou embaralhado e o servo deixou de poder atuar nele (isto aconteceu com muita frequência, mesmo antes de se colocar o veleiro na água) , não permitindo que o veleiro consegui-se manobrar as velas corretamente, o que explica as complicações demonstradas na navegação.

Na terceira e última tentativa, fizeram-se ajustes aos valores máximo e mínimo do servo das velas, retirando ângulo e capacidade de as folgar, mas evitando que o fenómeno anterior se voltasse a repetir.

Figura 5.86: REX - terceira tentativa.

Após a colocação na água, verificou-se que o problema anterior não voltou a ocorrer. No entanto, o comportamento do veleiro não correspondeu ao esperado novamente. Verificou-se que este não conseguia comportar-se de forma correta, dando a entender que em algumas situações tentava orientar-se com o destino, mas não era bem sucedido (as possíveis razões para este sucedido serão explicadas posteriormente).

Contudo, depois de retirado da água (mas ainda dentro do barco de resgate) foi feito uma validação para perceber se o sistema interno estava a ter o comportamento correto. Basicamente, foi realizada uma simulação de navegação do veleiro, mas ao invés de este estar na água, foi elevado no próprio barco de resgate, sendo que a deslocação da embarcação foi feita de acordo com o comportamento do veleiro (por exemplo, se o leme do

veleiro rodasse 30º para estibordo, a pessoa que controlava a embarcação também rodava o leme desta 30º para estibordo; se o leme do veleiro estivesse alinhado ao centro, por exemplo, o leme do barco de resgate também se alinharia ao centro, limitando-se a ir em frente).

Ora verificou-se que as indicações do veleiro corresponderam exatamente ao comportamento esperado (ou seja, o veleiro tentou alinhar-se com o destino e deslocar-se até lá), tendo este percorrido (indiretamente) os três *goals* inicializados.

5.3.1 Análise aos resultados do evento

Após o terceiro teste não houve, infelizmente, tempo para a realização de mais validações (não foi possível testar a navegação com os módulos desenvolvidos nesta dissertação, pois, além da falta de tempo, o veleiro não se mostrou capaz de conseguir navegar nas condições que se verificaram, pelo que também não seria capaz de simular uma prova de varrimento). No entanto, a participação neste evento permitiu retirar várias conclusões acerca da navegação do veleiro.

Relativamente ao primeiro teste, não há muito a considerar, tendo em conta que se verificou que o sistema do veleiro se encontrava desligado.

O segundo teste permitiu perceber que os fios do servo das velas têm uma grande facilidade em ficarem embaralhados, impedindo o veleiro de navegar em condições. de forma a resolver este problema a curto prazo, foram alterados os valores máximos. No entanto, futuramente devem ser consideradas outras soluções.

Na terceira e última experiência, o veleiro não foi capaz de percorrer os *goals* propostos. No entanto, averiguou-se posteriormente que o comportamento do leme foi o correto (com a experiência a bordo do barco de resgate). A origem desta incapacidade de cumprir o trajeto esperado pode ter várias hipóteses:

- Condições externas adversas: dada a localização dos testes (área compreendida entre fragatas), o vento e as correntes existentes eram muito irregulares, podendo ser um fator determinante para este insucesso;

- Manobra das velas ineficiente: há possibilidade de que os valores do servo das velas máximo e mínimo, definidos para evitar a ocorrência do problema dos fios, tenha impedido a manobra correta das velas e consequentemente a performance da navegação;

- Cata-vento pouco eficaz: o cata-vento usado neste evento demonstrou não ser ideal para o veleiro (não o sensor de *Hall*, mas sim a parte giratória) pois deu a entender que não roda com muita facilidade, não acompanhando a direção do vento como deveria e influenciando as decisões tomadas pelo veleiro;

- Estrutura física do veleiro desajustada: Um dos aspetos mais evidentes nestes testes foi o facto de o veleiro, em muitos momentos, navegar quase ”deitado”, ou seja,

com uma grande inclinação (a figura 5.86 é exemplo disso). Nesta posição, tanto o leme como as velas deixam praticamente de ter influência na navegação, impedindo o veleiro de executar as manobras pretendidas. A causa desta extrema inclinação está relacionada com a própria estrutura física do veleiro, mais especificamente a relação entre o casco/quilha e a área das velas. Velas demasiado grandes (em relação ao casco) cobrem uma maior área de incidência do vento, aumentando a força exercida por este, que não é compensada inversamente pelo casco e pela quilha. Esta diferença de forças resulta numa inclinação excessiva do veleiro, proporcional à força do vento.

5.4 Análise global dos resultados

No final de todos os testes e validações, foi feito um balanço total, onde foram retiradas as principais ideias e conclusões das anteriores validações.

Relativamente aos testes unitários, os resultados foram extremamente positivos. Tanto as implementações dos métodos de varrimento como a implementação da decisão da estratégia de navegação foram testadas individualmente, tendo os resultados sido os esperados. Desta forma, concluiu-se portanto que estes podiam ser integrados no projeto e proceder-se à fase de testes de integração.

Nos testes relativos à validação integral, os resultados também foram bastante bons. Foi inicialmente testada a integração dos módulos dos métodos de varrimento e da decisão da estratégia de navegação no ceio do projeto. Para todos os experimentos, o sistema calculou de forma inequívoca as coordenadas dos *goals* esperados, consoante o *input* proveniente das leituras feitas pelo cata-vento. Posteriormente, na simulação dos varrimentos, ocorreram alguns problemas relativos à deteção da chegada aos destinos, que (conjugados com outros fatores), impediram a obtenção de simulações completas relativamente aos métodos *Horizontal Full Scanning* e *Vertical Full Scanning*. No entanto, estes problemas/erros ocorridos em nada se relacionaram com os módulos implementados nesta dissertação, mas sim com o sensor de GPS e com outros fatores mencionados anteriormente. Portanto, no que toca ao conteúdo desenvolvido nesta dissertação, a análise é bastante positiva.

Por último, foi possível fazer testes em água, no evento REX, onde o veleiro foi posto à prova com situações reais de navegação. Infelizmente, o este não foi capaz de navegar em boas condições e consequentemente completar uma regatta, não tendo sido portanto possível realizar testes que validassem especificamente a integração dos módulos desenvolvidos, o que teria sido bastante positivo para retirar ilações acerca das decisões tomadas e da solução implementada. Apesar de tudo, este evento foi bastante útil pois permitiu a deteção de alguns problemas no veleiro que o impedem de efetuar uma boa navegação, servindo de balanço para melhorias futuras.

Capítulo 6

Conclusão e Trabalho Futuro

Nesta dissertação foi apresentada uma solução flexível e adaptável para realizar a prova de varrimento da competição *World Robotic Sailing Championship*.

A prova tem algumas regras e limitações que foram tidas em conta na decisão desta estratégia, nomeadamente o tempo limite (30 minutos), a área da prova e o sistema de pontuação.

Esta solução baseou-se portanto no conceito de flexibilidade, indo contra o típico conceito de varrimento rígido predefinido, onde os fatores externos não têm influência na estratégia adotada. Tendo em conta que a maior dificuldade da navegação à vela é navegar contra o vento (onde é necessário bolinar), a ideia foi implementar uma estratégia que variasse consoante a direção do vento, mais especificamente, navegar (aproximadamente) perpendicularmente ao vento. Desta forma, as situações de bolina são teoricamente mais reduzidas.

Além disso, desenvolveram-se quatro tipos de profundidade de varrimento, com diferentes graus de pontuação e de estimativa de tempo de prova, dando uma maior versatilidade e opção de escolha para "atacar" a prova.

Por último, foi implementado um varrimento "inteligente", onde é predefinido um tipo de varrimento mais profundo (mais vantajoso em termos pontuais), tendo o sistema capacidade de recalcular o trajeto na ocorrência de problemas ou dificuldades na navegação, para evitar a desqualificação da prova.

Resumindo, o sistema deverá ter capacidade para "decidir" qual a estratégia de navegação que mais se adequa à direção do vento, além de possuir autonomia para adotar um trajeto mais direto em caso de dificuldades na execução do varrimento predefinido.

A implementação desta solução foi inicialmente feita à parte do projeto, no desenvolvimento dos tipos de varrimento (*Horizontal Fast Scanning*, *Vertical Middle Scanning*, etc), usando o *software Visual Studio 2017* por ser uma ferramenta mais intuitiva e facilitadora

de *debug* (em relação ao Arduino IDE). A restante implementação foi feita diretamente no código do projeto *Novalantia*, tendo sido posteriormente adicionados os tipos de varrimento desenvolvidos.

No que toca aos testes, o balanço foi bastante positivo. Os testes de integração permitiram averiguar se os módulos implementados estavam em condições de serem testados juntamente com o resto do projeto. Após esta validação (com sucesso), foram então feitos testes de integração, para atestar se o sistema se comportava corretamente com as adições feitas. Os resultados desta parte foram também bastante bons. O único ponto menos positivo a referir (que não está propriamente relacionado com o conteúdo desenvolvido nesta dissertação) prende-se com o facto de nos testes de simulação realizados, o GPS ter perdido várias vezes o sinal, que dificultou bastante a deteção de chegada aos objetivos calculados pelo sistema.

Relativamente aos testes realizados no evento REX, não foi possível validar os módulos desenvolvidos numa situação real de navegação, pois o veleiro não conseguiu navegar de forma correta, mesmo em experiências mais simples como uma regatta. No entanto, foram bastante importantes para retirar ilações para o futuro.

A realização desta dissertação teve algumas implicações e dificuldades que são importantes referir. O facto de a documentação existente sobre a navegação implementada e o correspondente código dos anos anteriores ser praticamente nula, dificultou bastante a compreensão do projeto. Daí, houve a necessidade de ser explicado o conteúdo (em anexo) do projeto desenvolvido anteriormente, assim como os módulos acrescentados relacionados com a esta dissertação, com o objetivo de documentar as funcionalidades mais importantes do código, para permitir uma maior facilidade de compreensão sobre o que já foi desenvolvido aos alunos que no futuro venham a trabalhar no projeto do veleiro. Outra grande dificuldade encontrada esteve relacionada com a montagem do equipamento do veleiro, pois muitos componentes/fios/conectores não se encontravam no melhor estado.

Num balanço final, pode concluir-se que as soluções apresentadas foram validadas com sucesso, ficando, no entanto, a faltar validar o conteúdo desta dissertação numa prova real de varrimento.

Além disso, esta dissertação está aberta a melhorias, deixando várias ideias para trabalhar futuramente:

- Relativamente à decisão da estratégia de navegação, esta é feita antes do inicio do varrimento propriamente dito, através do cálculo da média das amostras provenientes do cata-vento. Futuramente, poderia pensar-se numa solução que, além de efetuar esta decisão apenas no inicio da navegação, possa fazê-lo também durante a execução do varrimento;

- Os próprios métodos de varrimento implementados também estão abertos a melhorias, caso se encontre uma forma de aperfeiçoar o varrimento;

- Aconselha-se também a adotar outra estratégia relativamente ao servo das velas, ou ao fio que as puxa, devido à sua debilidade;
- Nos testes efetuados no evento REX, notou-se também nas dificuldades do veleiro em manter uma postura vertical, estando este facto relacionado com a relação entre a área abrangida pelas velas e o casco/quilha. Deste modo seria também aconselhável a encontrar uma solução para este problema.

Bibliografia

[1] World Robotic Sailing Championship. *World Robotic Sailing Championship 2017 - Notice of race and competition rules*. https://docs.wixstatic.com/ugd/eacf69_dcd85950bf6f48b68a68aa0cbf20ca82.pdf. 2017.

[2] L. Gomes, A. Costa, D. Fernandes, H. Marques e F. Anjos. "Improving instrumentation support and control strategies for autonomous sailboats in a regatta contest". Em: *ISRC'16 - 9th International Robotic Sailing Conference* (2016).

[3] J. Esteves, L. Gomes e A. Costa. "Collision Avoidance System for an Autonomous Sailboat". Em: *IECON'2017 – 43rd Annual Conference of the IEEE Industrial Electronics Society* (2017).

[4] S. Abbey. *How to Sail a Boat*. URL: https://www.wikihow.com/Sail-a-Boat. (accessed: 26.01.2019).

[5] T. J. de Almeida Pereira. "eVentos 3 – Desenvolvimento de controlador difuso para navegação autónoma de veleiro". Tese de mestrado. Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, set. de 2015.

[6] Associação Náutica da Gafanha da Encarnação. *Navegação à Vela*. URL: http://ange.pt/index.php/arquivo/navegacao. (accessed: 26.01.2019).

[7] Mundo da Vela. *Rumo e Mareações*. URL: https://sites.google.com/site/marvelavento/lingua-language/portugues/conceitos/rumo-e-mareacoes. (accessed: 27.01.2019).

[8] S. Chakraborty. *How Does A Rudder Help In Turning A Ship?* URL: https://www.marineinsight.com/naval-architecture/rudder-ship-turning/. (accessed: 27.01.2019).

[9] *FASt Autonomous Sailboat*. URL: https://paginas.fe.up.pt/~jca/fast/. (accessed: 30.01.2019).

[10] "Campeão Mundial em 2012, FASt a competir no Rio Lima". Em: *Olhar Viana do Castelo* (2016). DOI: https://olharvianadocastelo.blogspot.com/2016/09/campeao-mundial-em-2012-fast-competir.html.

[11] J. C. Alves e N. A. Cruz. *Um sistema computacional reconfigurável embarcado num veleiro autónomo*. Rel. téc. Faculdade de Engenharia da Universidade do Porto.

[12] URL: https://paginas.fe.up.pt/~jca/wrsc/competitors/team-ssa/ssa.jpg. (accessed: 01.02.2019).

[13] G.-A. Busser. "Design and Implementation of a Navigation Algorithm for an Autonomous Sailboat". Tese de mestrado. Swiss Federal Institute of Technology Zurich, 2009.

[14] *AberSailbot*. URL: https://abersailbot.co.uk/. (accessed: 12.02.2019).

[15] *Autonomous Sailing Robot Project*. URL: https://www.southampton.ac.uk/~yc6n13/mrobot.html. (accessed: 12.02.2019).

[16] URL: http://www.sotonsailrobot.org/#. (accessed: 12.02.2019).

[17] H. M. C. Marques. "Visualizer of sailboat navigation integrating instrumentation emulators". Tese de mestrado. Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, mar. de 2017.

[18] L. Gomes, M. Santos, T. Pereira e A. Costa. "Model-based development of an Autonomous Sailing Yacht controller". Em: *ICARSC'2015 – 2015 IEEE International Conference on Autonomous Robot Systems and Competitions* (2015). DOI: 10.1109/ICARSC.2015.20.

[19] J. Monteiro. *Comunicação Série e Paralela*. Rel. téc. Lisboa: Instituto Superior Técnico, mai. de 2009.

[20] Philips Semiconductors. *The I2C-Bus specification*. 2000. URL: https://www.csd.uoc.gr/~hy428/reading/i2c_spec.pdf.

[21] URL: https://cdn.sparkfun.com/assets/6/4/7/1/e/51ae0000ce395f645d000000.png. (accessed: 13.02.2019).

[22] R. Toulson e T. Wilmshurst. "Fast and Effective Embedded Systems Design". Em: 2ª ed. Newnes, out. de 2016. Cap. 4.3. ISBN: 9780081009031.

[23] *Getting Started with Arduino and Genuino MEGA2560*. URL: https://www.arduino.cc/en/Guide/ArduinoMega2560. (accessed: 13.02.2019).

[24] *AS5161: 12-Bit Magnetic Angle Position Sensor*. Revision 1.1. AMS.

[25] *MT3329 Data Sheet*. Version 0.5. MediaTek.

[26] *CMPS11 - Tilt Compensated Compass Module*. URL: https://www.robot-electronics.co.uk/htm/cmps11doc.htm.

[27] URL: https://www.researchgate.net/figure/Definitions-of-reference-frames-and-ship-motions-surge-sway-heave-roll-pitch-and-yaw_fig1_245386997. (accessed: 13.02.2019).

[28] J. M. F. Esteves. "Desenvolvimento de um Sistema anticolisão para um veleiro com navegação autónoma". Tese de mestrado. Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, set. de 2017.

[29] P. N.M. P. Maricato. "Navegação autónoma de veleiro com compensação de correntes". Tese de mestrado. Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, mar. de 2019.

ANEXO I

Explicação do ficheiro WRSC_NOVALANTIA.ino

I.1 *setup()*

```
void setup() {
  Serial.begin(DEBUG_SERIAL_BAUDRATE);
  Serial3.begin(GPS_SERIAL_BAUDRATE);

  rudder.attach(RUDDER_SERVO_PIN);
  sail.attach(SAIL_SERVO_PIN);

  remoteRudderPosition = (RUDDER_MAX_PULSE_WIDTH + RUDDER_MIN_PULSE_WIDTH) / 2;
  remoteSailPosition = SAIL_MAX_PULSE_WIDTH;

  //servo actuation
  servo_act(remoteRudderPosition, remoteSailPosition);                                   1

  // start reading pwm signal with interrupt
  attachInterrupt(digitalPinToInterrupt(ROTARY_ENCODER_PIN), rising, RISING);            2

  route_race[2] = 38.660702;
  route_race[3] = -9.204103;
  route_race[4] = 38.661064;
  route_race[5] = -9.20548;
  route_race[6] = 38.660764;
  route_race[7] = -9.206893;
  route_race[8] = 38.660858;
  route_race[9] = -9.204855;
  //Code 200 end of regatta
  route_race[10] = 200;

  cone_parameters[0] = 5;
  cone_parameters[1] = -5;
  cone_parameters[2] = 5;
  cone_parameters[3] = -5;
  cone_parameters[4] = 5;
  cone_parameters[5] = -5;
  cone_parameters[6] = 5;
  cone_parameters[7] = -5;                                                               3

  debugDelay = millis();

  //Petri net setup
  createInitial_Controlador_Bolina_NetMarking(&marking);
  init_Controlador_Bolina_OutputSignals(&place_out, &ev_out);
  Controlador_Bolina_InitializeIO(&bowline);
  Controlador_Bolina_GetInputSignals(&prev_inputs, NULL);                                4

  //Send the goals to radio
  remote.sendPlannedTrack(route_race, cone_parameters);                                  5
}
```

Figura I.1: Função *setup()* do ficheiro "WRSC_NOVALANTIA.ino".

- **1)** Definição dos valores iniciais do servos do leme e das velas, respetivamente, e posterior atuação nos mesmos (*servo_act()*). Neste caso, foi definido o valor do servo do leme para que este fique centrado com a embarcação, e o valor do servo das velas para estas ficarem caçadas (SAIL_MAX_PULSE_WIDTH - caçadas, SAIL_MIN_PULSE_WIDTH - folgadas);
- **2)** Função que inicia a leitura dos sinais PWM associados ao cata-vento;
- **3)** Definição da rota, ou seja, dos *goals* a atingir. As posições pares do vetor *route_race[]* correspondem à coordenada latitude dos *goals*, enquanto que as posições ímpares

correspondem à coordenada longitude. Estas apenas começam a ser atribuídas a partir do índice 2, pois as posições 0 e 1 dizem respeito às coordenadas da localização atual do veleiro. O vetor *cone_parameters* diz respeito aos parâmetros das "balizas" dos *goals* (ver subsecção sobre o ficheiro Bowline_math.cpp);

- **4)** Funções associadas à rede de Petri e ao controlador (não devem ser alteradas);

- **5)** Função para enviar a rota (os vetores com as coordenadas dos *goals* e com os parâmetros das "balizas" destes) para o rádio.

I.2 *loop()*

A primeira instrução - *data.dataGps()* - desta função verifica a receção de dados pelo sensor GPS (ver explicação do ficheiro Data_acquisition.cpp).

As instruções do corpo da seguinte condição - *if(stringComplete)* - dizem respeito à comunicação com o controlo remoto, onde são atualizados os valores dos servos das velas e do leme, introduzidos pelo utilizador do controlo.

Após estes dois blocos de código, o sistema verifica se a navegação será autónoma (executando as instruções marcadas pela área a amarelo) ou se será manual, executando a função que atua nos servos - servo_act(remoteRudderPosition, remoteSailPosition) - com os valores provenientes do controlo remoto.

Focando no código que diz respeito à navegação autónoma (a amarelo), este pode ser dividido em 7 principais partes.

```
void loop() {
  data.dataGps();

  if (stringComplete) {
    remote.remoteCommands(msg_received, &remoteControl, &remoteSailPosition, &remoteRudderPosition,route_race,cone_parameters);
    msg_received = "";
    stringComplete = false;
  }

  if (!remoteControl) {
    wind_alignment = windAlignment;
    wind_velocity = abs(roll) * 6;

    if (bowline.arrival_destination(actual_lat, actual_lng)) {
      next_destination = next_destination + 2;
      next_origin = next_origin + 2;
    }                                                                                                          1

    data.retrieveCompassData(&bearing, &pitch, &roll);
    data.windVanePosition(&windAlignment, &wind_alignment,&pwm_value);                                         2

    if (data.dataGps()) {
      data.direction_destination(bearing, &angle_edges, &route, &goalDistance,
        route_race[next_destination], route_race[next_destination + 1]);
      data.actual_position(&actual_lat, &actual_lng);                                                          3

      if (change_origin && actual_lng != 0 && actual_lat != 0) {
        route_race[0] = actual_lng;
        route_race[1] = actual_lat;
        change_origin = false;
      }                                                                                                        4

      bowline.fill_struct(angle_edges, bearing, wind_alignment, route_race[next_origin + 1], route_race[next_origin],
        route_race[next_destination + 1], route_race[next_destination], actual_lng, actual_lat,
        cone_parameters[next_origin], cone_parameters[next_origin + 1]);                                       5

      if (trace_control != TRACE_PAUSE) {
        Controlador_Bolina_ExecutionStep(&marking, &inputs, &prev_inputs, &place_out, &ev_out);
      }
      else {
        Controlador_Bolina_GetInputSignals(&inputs, NULL);
      }
      if (trace_control > TRACE_PAUSE) {
        --trace_control;
      }
      Controlador_Bolina_LoopDelay();
      if (get_Controlador_Bolina_PlaceOutputSignals()->Out_angle != 0) {
        bowline.bowline_navigation_coordinates(&longitude_bowline, &latitude_bowline, actual_lng, actual_lat,
          route_race[next_destination + 1], route_race[next_destination],
          (get_Controlador_Bolina_PlaceOutputSignals()->Out_angle) * BOWLINE_ANGLE);
        data.direction_destination(bearing, &angle_edges, &route, &goalDistance, latitude_bowline, longitude_bowline);
      }
      wrsc_controller_v2InferenceEngine((double)wind_velocity, (double)route, (double)windAlignment,
        &sails_activation, &rudder_activation);
      servo_act(map(rudder_activation, 0, 100, RUDDER_MAX_PULSE_WIDTH, RUDDER_MIN_PULSE_WIDTH),
        map(sails_activation, 0, 100, SAIL_MAX_PULSE_WIDTH, SAIL_MIN_PULSE_WIDTH));                           6
    }
    if (millis() - debugDelay >= debugDelayTime) {
      remote.radioFeedbackFullInfo(
        data.gps.time.hour(), data.gps.time.minute(), data.gps.time.second(), data.gps.time.centisecond(),
        route_race[next_destination], route_race[next_destination + 1], data.gps.location.lat(), data.gps.location.lng(),
        data.gps.course.deg(), data.gps.speed.kmph(), data.gps.hdop.hdop()*3.0, wind_alignment, wind_velocity,
        (int)bearing, pitch, roll, route);                                                                     7
    }
    debugDelay = millis();
  }
  else
    servo_act(remoteRudderPosition, remoteSailPosition);
}
```

Figura I.2: Função *loop()* do ficheiro "WRSC_NOVALANTIA.ino".

- **1)** Nesta condição verifica-se se o *goal* atual já foi atingido, através da função *bowline.arrival_destination()* que recebe como parâmetros de entrada as coordenadas da localização atual do veleiro. Caso se verifique a condição, atualizam-se as variáveis *next_destination* e *next_origin* com o índice do próximo destino/*goal* e com o índice da posição/*goal* de origem (ou seja, o *goal* que foi atingido), respetivamente (ambos respetivos ao vetor *route_race[]*);
- **2)** Funções que fazem a leitura dos sensores associados ao cata-vento e à bússola, guardando os valores lidos nas variáveis respetivas;
- **3)** Funções do ficheiro Data_acquisition.cpp que fornecem informações acerca do destino e da localização atual do veleiro, respetivamente;
- **4)** Porção de código onde se define a posição inicial do veleiro (índices 0 e 1 do vetor *route_race[]*);
- **5)** Função do ficheiro Bowline_math.cpp que é usada para preencher as variáveis a usar por outras funções deste ficheiro;
- **6)** Código composto por funções associadas à rede de Petri, à navegação à bolina (*bowline.bowline_navigation_coordinates()*), ao controlador (*wrsc_controller_v2InferenceEngine*()) e à atuação dos servos;
- **7)** Função de *feedback* que envia para o rádio a informação das variáveis respetivas - usada essencialmente para *debug*.

ANEXO II

Explicação do ficheiro Data_acquisition.cpp

- ***retrieveCompassData()*** - Função que recolhe os dados lidos pela bússola através do protocolo I2C. Tem como *output* as variáveis *roll*, *pitch* e *bearing* (ângulo da bússola em relação ao norte geográfico);

```
void Data_acquisition :: retrieveCompassData(double *bearing, int *pitch, int *roll){
  byte highByte, lowByte;
  Wire.beginTransmission(COMPASS_I2C_ADDRESS);
  Wire.write(2);
  Wire.endTransmission();
  Wire.requestFrom(COMPASS_I2C_ADDRESS, 4);
  while (Wire.available() < 4);
  highByte = Wire.read();
  lowByte = Wire.read();
  *bearing = word(highByte, lowByte) / 10.0;
  *bearing = normalizedAngleSum(*bearing, MAGNETIC_VARIATION+COMPASS_ERROR);

  *pitch = (int)Wire.read();
  *roll = (int)Wire.read();
}
```

Figura II.1: Função *retrieveCompassData()* do ficheiro "Data_acquisition.cpp".

- ***windVanePosition()*** - Lê os dados do sensor do cata-vento. Tem como *output* as variáveis *pwm_value* (valor PWM diretamente lido pelo sensor), *wind_alignment* (valor lido convertido para uma gama de 0º a 360º) e *windAlignment* (valor lido convertido para uma gama de 0º a 180º);

```
void Data_acquisition :: windVanePosition(int *windAlignment, int *wind_alignment,int * pwm_value){

  int a = *pwm_value;
  *windAlignment=map(a, ROTARY_ENC_MIN_PULSE_WIDTH, ROTARY_ENC_MAX_PULSE_WIDTH, 0, 360)-35;
  *wind_alignment=*windAlignment;

  if(*windAlignment > 180)
    *windAlignment = *windAlignment - 360;
  *windAlignment=abs(*windAlignment);
}
```

Figura II.2: Função *windVanePosition()* do ficheiro "Data_acquisition.cpp".

- ***dataGps()*** - Função que verifica a leitura dos dados do GPS durante um determinado intervalo de tempo. Retorna o valor lógico *"true"* caso o GPS tenha recebido dados válidos neste intervalo de tempo, e retorna *"false"* caso contrário;

```
bool Data_acquisition :: dataGps() {
  unsigned long ms = 100;
  unsigned long start = millis();
  char inChar;
  bool flag= false;

  do
  {
    if (Serial3.available()) {
      inChar = Serial3.read();
      if(gps.encode(inChar))
        flag = true;
    }
  } while ((millis() - start < ms));
  return flag;
}
```

Figura II.3: Função *dataGps()* do ficheiro "Data_acquisition.cpp".

ANEXO III

Implementação do mecanismo *Smart Scanning*

```
9     if(smart_scanning && (scanning_depth == 3 || scanning_depth == 4)){
10      actual_lat = data.gps.location.lat();
11      actual_lng = data.gps.location.lng();
12      if (bowline.arrival_destination(actual_lat, actual_lng)) {
13        if(next_destination == 6){
14           checkpoint_1_reached = true;
15        }
16        else
17          if(next_destination == 10){
18            checkpoint_2_reached = true;
19          }
20
21        next_destination = next_destination + 2;
22        next_origin = next_origin + 2;
23
24        if(route_race[next_destination] == 200){
25          end_of_navigation = true;
26        }
27      }
28      else{
29        current_time = data.gps.time.hour() * 60 + data.gps.time.minute();
30
31        if(checkpoint_1_reached == false && deadline_1_exceeded == false){
32           if(current_time - start_time >= CHECK_POINT_1_DEADLINE){
33              route_race[10] = data.gps.location.lat();
34              route_race[11] = data.gps.location.lng();
35              next_origin = 10;
36              next_destination = 12;
37              deadline_1_exceeded = true;
38              smart_scanning = false;
39           }
40        }
41        else{
42          if(checkpoint_2_reached == false && deadline_2_exceeded == false){
43            if(current_time - start_time >= CHECK_POINT_2_DEADLINE){
44                route_race[12] = data.gps.location.lat();
45                route_race[13] = data.gps.location.lng();
46                next_origin = 12;
47                next_destination = 14;
48                deadline_2_exceeded = true;
49                smart_scanning = false;
50            }
51          }
52        }
53      }
54    }
55    else{
56      actual_lat = data.gps.location.lat();
57      actual_lng = data.gps.location.lng();
58      if (bowline.arrival_destination(actual_lat, actual_lng)) {
59        next_destination = next_destination + 2;
60        next_origin = next_origin + 2;
61
62        if(route_race[next_destination] == 200){
63          end_of_navigation = true;
64        }
65      }
66    }
```

Figura III.1: Implementação do mecanismo *Smart Scanning*.