ESCOLA NAVAL

talante bien faire

Teresa Sofia Vidinha da Costa

Otimização, em tempo real, de trajetórias para veleiros

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

Alfeite

2021

Teresa Sofia Vidinha da Costa

Otimização, em tempo real, de trajetórias para veleiros

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

Orientação de: Miguel Ângelo Pereira Bento Moreira
Co-orientação de: Jorge Manuel Lourenço Gorricha

O Aluno Mestrando,

Teresa Costa

O Orientador,

Prof. Miguel Moreira

Alfeite
2021

Aut inveniam viam aut faciam,

latim para "Vou encontrar um caminho ou faço um."

Hannibal, 218 A.C.

Dedicado à minha mãe Teresa, ao meu pai Jorge e aos meus irmãos Miguel e João.

Agradecimentos

Agradeço extremamente a todos que, direta e indiretamente, permitiram que esta dissertação de mestrado fosse possível.

Agradeço:

- Ao meu orientador, Prof. Miguel Bento Moreira, que desde o primeiro dia transmitiu motivação, apoio e coragem e se mostrou disponível para levar este desafio a bom porto. Ao meu co-orientador CFR M Lourenço Gorricha por ser impulsionador de programação na vidas dos cadetes.
- Ao Prof. Ricardo Moura, que despendeu do seu tempo para me ajudar a traduzir o programa de Matlab para Python e ao Tiago pela paciência para rever o programa comigo.
- Ao curso capitão-tenente Raul Alexandre Cascais por terem sido a minha família naval, e em particular às camaradas ASPOF Melo Jerónimo e ASPOF Reis Sintra com quem, por ao longo destes 5 anos, terem estado sempre presentes e nunca me terem falhado quando mais precisei, criei laços de grande amizade, espírito de camaradagem e lealdade.
- À câmara de oficiais do NRP *Corte-Real* que me recebeu de braços abertos e cujos excelentes exemplos me servirão de inspiração para este meu início de carreira como Oficial.
- Ao João, à Sofia e à Tânia por me mostrarem nestes últimos tempos que as amizades antigas quando são verdadeiras, o passar do tempo apenas as fortalece.
- À Raquel por me inspirar a querer fazer mais e melhor. Por ser um porto seguro e de apoio nesta caminhada.
- Finalmente mas não menos importante, aos meus pais e aos meus irmãos, pela educação, empenho, amor e dedicação ao longo de toda a minha vida. E à família Casquilho, nomeadamente à Joana, à Elsa e ao José, por serem a família que escolhi com o coração.

Resumo

A navegação à vela entre dois pontos não pode em geral ser realizada adoptando a correspondente derrota direta. As caraterísticas do veleiro e a direção do vento obrigam a adotar trajetórias constituídas por diferentes pernadas realizadas a rumos distintos do rumo direto. A escolha das pernadas influência diretamente o tempo de trajeto. A escolha do trajeto que minimize o tempo de percurso é um problema de otimização complexo de difícil abordagem analítica.

A disponibilização duma ferramenta computacional que resolva este problema é, sem dúvida alguma, uma mais-valia para os utilizadores de veleiros de qualquer dimensão. No contexto da vela autónoma a disponibilização desta ferramenta, mais do que ser uma mais-valia, é um fator essencial na utilização deste tipo de meios.

Nesta dissertação, usando técnicas de otimização inspiradas na simulação do recozimento, desenvolveremos uma ferramenta destinada a calcular trajetórias que minimizam o tempo de percurso dum veleiro entre dois pontos, utilizando um número de pernadas fixado inicialmente (duas ou mais pernadas). Não consideraremos, para simplificar, os efeitos das correntes e assumir-se-á que o campo de velocidades do vento é uniforme e estacionário. A ferramenta computacional desenvolvida é parametrizável com recurso ao diagrama polar de velocidades de qualquer veleiro. Simulações numéricas usando veleiros com diferentes caraterísticas, diferentes trajetos e diferentes configurações do campo de velocidade do vento serão apresentadas e discutidas.

O recurso a um modelo fenomenológico, parametrizável, de penalização temporal das mudanças de rumo, destinadas a descrever as pernadas geradas, conduz à geração de soluções geometricamente mais simples e com menos pernadas.

Palavras-chave: Velejar, Otimização, Vela Robótica, Simulação do Recozimento, Matlab, Python

Abstract

In general, sailing between two points can't be carried out by adopting a corresponding direct course. The characteristics of sail boats and the wind direction compel it to adopt trajectories with different courses to steer, each of them with a distinctive bearing, when compared to the direct course. The choice of courses to steer has a direct influence over the journey time. The choice of a path that minimizes the travel time is a complex optimization problem, encompassing a difficult analytical approach.

The availability of a computational tool to solve this problem is undoubtedly an added value asset for users of sailboats of any size. In the scope of autonomous sailing, the availability of this tool, more than being an added value asset, is an essential factor in the use of this type of means.

In this work, using optimization techniques based on the simulation of annealing, we will develop a tool to calculate trajectories allowing the minimization of a sailboat travel time between two points, using a predetermined number of courses to steer (two or more). In order to simplify the problem, we will not consider the effects of the currents, and it will be assumed that the range of wind speeds is uniform and stationary. The computational tool developed is configurable using the polar diagram of any sailboat.

We will present and discuss numerical simulations using sailboats with different characteristics, different paths and different configurations of the wind speed range. The use of a phenomenological and parameterizable model, with temporal penalization of course changes, designed to describe the generated courses to steer, produces solutions which are geometrically simpler and with a lower number of courses to steer.

Keywords: Sail, Optimization, Robotic Sailing, Simulation annealing, Matlab, Python

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

CINAV	**C**entro de **I**nvestigação **N**aval
GPS	**G**lobal **P**ositioning **S**ystem
NOAA	**N**ational **O**ceanic and **A**tmospheric **A**dministration
AIS	**A**utomatic **I**dentification **S**ystem
UKHO	**U**nited **K**ingdom **H**ydrographic **O**ffice
COG	**C**ourse **O**ver **G**round
SOG	**S**peed **O**ver **G**round
NMEA	**N**ational **M**arine **E**lectronics **A**ssociation
SA	**S**imulated **A**nnealing
TS	**T**abu **S**earch
GRASP	**G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
ILS	**I**terated **L**ocal **S**earch
VNS	**V**ariable **N**eighborhood **S**earch
RN	**R**edes **N**euronais
CF	**C**olónia de **F**ormigas
PSO	**P**article **S**warm **O**ptimization
EA	**E**volutionary **A**lgorithms
AG	**A**lgoritmos **G**enéticos
SS	**S**catter **S**earch
AMP	**A**daptive **M**emory **G**round
FAA	**F**orça de **A**rrasto
FSA	**F**orça de **S**ustentação
SR	**S**imulação de **R**ecozimento
OTV	**O**timização da **T**rajetória de **V**eleiros
APOSV	**A**ngulo **P**or **O**nde **S**opra o **V**ento
BEV	**B**ordo **E**ntrada de **V**ento
BOL	**Bol**ina
NE	**N**avio **E**cola
NRP	**N**avio da **R**epública **P**ortuguesa
TTOCP	**T**empo **T**rajeto **Ó**timo **C**om **P**enalização
TTOSP	**T**empo **T**rajeto **Ó**timo **S**em **P**enalização

CPU **C**entral **P**rocessing **U**nit

Lista de Símbolos

α_{opt}	Ângulo entre a direção do vento e a *Layline*	[º]
ν_{opt}	Velocidade ótima	[m/s]
γ	Ângulo entre a direção do vento e a bóia da regata	[º]
ϕ	Ângulo	[º]
A	Posição inicial	
P	Ponto de guinada	
B	Posição final	
f	Função Objetivo	
$\vec{v}$	Velocidade do vento verdadeiro	[knot]
$\vec{v}_{ap}$	Velocidade do vento aparente	[knot]
$\vec{V}$	Velocidade do veleiro	[knot]
θ	Rumo	[knot]
σ	Celeridade do veleiro	[U]
ρ_{ar}	Densidade do ar	[kg/m^3]
ρ_{agua}	Densidade da água	[kg/m^3]
T	Tempo de trajeto	[h]
k_1	Constante	
k_2	Constante	
α	Ângulo total de guinada	[º]
F_s	Força de sustentação	[N]
F_r	Força resultante	[N]
F_l	Força lateral	[N]
F_a	Força de arrasto	[N]
F_u	Força útil	[N]
v_a	Vento aparente	[knot]

Capítulo 1

Introdução

1.1 Motivação

A navegação à vela entre dois pontos não pode em geral ser realizada adotando a correspondente derrota ortodrómica. As caraterísticas do veleiro e a direção do vento obrigam a adotar trajetórias constituídas por diferentes pernadas realizadas a rumos distintos do rumo direto. A escolha das pernadas e o seu comprimento influência diretamente o tempo de trajeto. A escolha do trajeto que minimize o tempo de percurso é um problema de otimização complexo de difícil abordagem analítica.

A disponibilização duma ferramenta computacional que aborde e resolva este problema é, sem dúvida alguma, uma mais-valia para os utilizadores de veleiros de qualquer dimensão.

No contexto da vela autónoma a disponibilização desta ferramenta, mais do que ser uma mais-valia, é um fator essencial na utilização deste tipo de meios, cujo desenvolvimento pretende ser protagonizado pela Escola Naval/CINAV.

Neste trabalho iremos proceder ao desenvolvimento de um algoritmo que minimize o referido tempo de trajeto recorrendo a duas ou mais pernadas. Tal algoritmo será dotado duma função de penalização temporal associada às mudanças de bordo que se verificam na realização das pernadas. Para além da criação de conhecimento na área, este trabalho tem em vista a integração dos algoritmos desenvolvidos na biblioteca de sub-rotinas de veleiros robóticas autónomos a desenvolver e utilizar no contexto do projeto *ROBSAILTEAM*. Posteriormente o algoritmo será desenvolvido em *Python*™ tendo em vista a sua aplicação em tempo real pelo sistema de controlo destes mesmos veleiros autónomos.

Recentemente foi desenvolvido um algoritmo para minimizar o tempo de trajeto entre dois pontos recorrendo a duas pernadas (Fedorchuk, 2020).

Neste trabalho (Fedorchuk, 2020), para além de se considerarem apenas trajetos com duas pernadas (1 único ponto de guinada), a definição inicial do espaço de pesquisa teve por base uma discretização regular do espaço de manobra com passos parametrizáveis pelo utilizador. A implementação do método de otimização (método heurístico inspirado no método da Simulação do Recozimento ou *simulated annealing*) não considerou qualquer penalização temporal associada à realização da mudança de rumo em cada ponto de guinada (Fedorchuk, 2020).

No presente trabalho considerar-se-ão trajetos com um número de pernadas fixado inicialmente pelo utilizador. A definição inicial do espaço de pesquisa será realizada gerando aleatoriamente diferentes trajetos com o número de pernadas fixado inicialmente. Na implementação do método de otimização (método heurístico inspirado no método da Simulação do recozimento ou *simulated annealing*) e tendo em vista promover a geração de trajetos ótimos mais realistas, utilizaremos uma função de penalização temporal associada à realização da mudança de rumo em cada ponto de guinada. Finalmente o programa será traduzido para *Python*™ (e verificado) tendo em vista a sua utilização em tempo real e a breve trecho no sistema de comando e controlo de veleiros autónomos no contexto do projeto ROBSAILTEAM da EN.

1.2 O problema e os seus diferentes níveis de complexidade

Pese embora não seja nossa intenção abordar a resolução das formas mais complexas do problema da minimização do tempo de trajeto entre dois pontos, contextualizaremos de seguida os diferentes níveis de complexidade que se podem colocar:

- O campo da velocidade do vento e corrente, durante todo o percurso do veleiro é espacialmente uniforme e estacionário;
- O campo de velocidades do vento e corrente não é espacialmente uniforme, sendo no entanto estacionário;
- O campo de velocidades do vento e corrente não é espacialmente uniforme, nem estacionário.

Será a primeira das formulações apresentadas que abordaremos neste trabalho.

1.3 Enquadramento do problema - Minimizar o tempo de trajeto do ponto A para o ponto B

Usando técnicas de otimização baseadas na simulação do recozimento, desenvolveremos uma ferramenta destinada a calcular trajetórias que minimizam o tempo de percurso dum veleiro entre dois pontos, utilizando um número de pernadas fixado inicialmente (duas ou mais pernadas). O campo de aplicação destina-se a trajetos curtos onde não se considerará, como limitação deste nível de complexidade, os efeitos das correntes e assumir-se-á que o campo de velocidades do vento é uniforme e estacionário. Se os cálculos forem atualizados em "tempo real"as limitações de utilização poderão ser diminutas.

A ferramenta computacional desenvolvida é parametrizável com recurso ao diagrama polar de velocidades de qualquer veleiro.

Foi criada uma função penalizadora que, quando aplicada, faz com que o programa apresente o trajeto ótimo, ou seja, neste caso com o menor tempo, tendo em conta o tempo que se iria perder quando se guina.

1.4 Estado da Arte

Nesta secção iremos apresentar e discutir as principais metodologias desenvolvidas no passado e utilizadas para abordar a resolução do problema da minimização do tempo de trajeto de um veleiro entre dois pontos.

1.4.1 Otimização de rotas para veleiros sob condições meteorológicas incertas

A vela de competição tais como a *America's Cup,Cowes Races, Mug Races, The Ocean Race* (Marcin Życzkowski, 2017) fez aumentar o investimento no estudo do problema da diminuição do tempo de trajeto realizado pelos veleiros. As competições de vela maioritariamente designadas por regatas podem ser de pequeno ou de longo curso sendo que as primeiras caracterizam-se por ser predominante a prática de navegação costeira e as segundas por navegação oceânica.

Foi em 2001 que Andy Philpott e Andrew Mason(A. Philpott, 2001) realizaram um estudo da rota de Veleiros, tendo em conta minimizar o tempo necessário para percorrer um trajeto entre dois pontos. A metodologia desenvolvida dirige-se à resolução de problemas quando o campo de velocidade do vento não é estacionário. Estabeleceram dois modelos, um de pequeno curso e outro de longo curso. O modelo que melhor se insere no nosso estudo é o de pequeno curso, o qual vai ser analisado agora.

O problema foi dividido em várias etapas e a decisão adotada na resolução de cada etapa é dependente da decisão anteriormente tomada. A decisão ótima é determinada através do conjunto de variáveis onde se encontram todas as decisões tomadas. De modo a obter as diferentes velocidades do veleiro para cada condição de vento foi utilizado o diagrama polar desse mesmo veleiro.

Para a programação do vento utilizaram cadeias de Markov e com base em observações da direção do vento, calcularam o melhor rumo a seguir em cada ponto do percurso de forma a minimizar o tempo previsto de chegada ao próximo nó.

Uma cadeia de Markov consiste numa sucessão de estados dum sistema em que cada estado futuro é caracterizado probabilisticamente apenas pelo estado presente não dependendo de nenhum estado passado (Pessoa, 2011).

O algoritmo de programação criado foi desenvolvido em C++. Assumindo uma intensidade de vento constante, ou seja, resolução do problema estacionário, a sucessão de direções do vento foi modelada recorrendo a uma cadeia de Markov cujos sucessivos estados estavam na base da determinação dos trajetos ótimos entre posições determinadas (A. Philpott, 2001).

1.4.1.1 Otimização Estocástica de trajetórias de vela em regatas

O trabalho realizado pelos autores (Dalang, 2010) conjuntamente com a equipa de vela suíça Alinghi e o navegador da mesma J. Vila para a regata *America's Cup* em 2007 tinha como objetivo permitir ajustar sempre que necessário o rumo do veleiro sempre que se verificassem alterações no estado do vento. A metodologia proposta foi aplicada a uma situação de regata entre dois veleiros ilustrada na Figura 1.1:

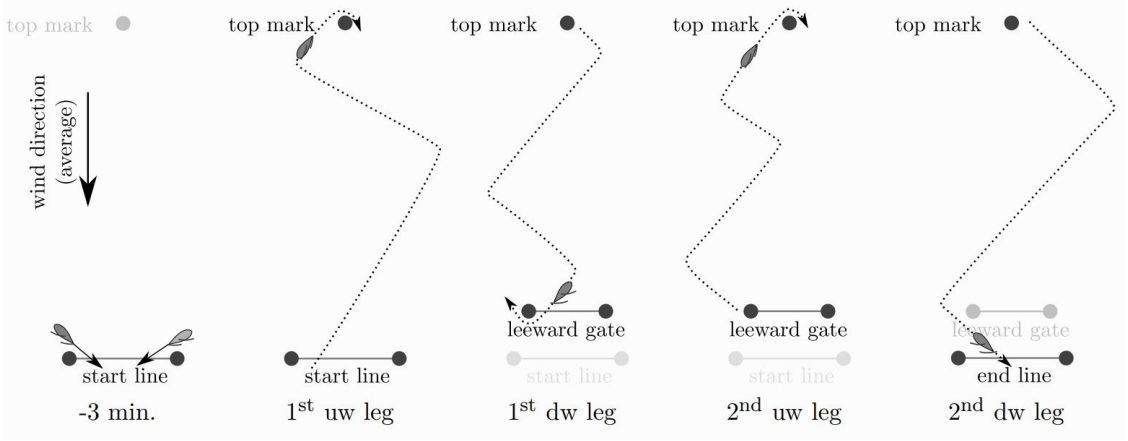


FIGURA 1.1: Regata em estudo. "*uw leg*"é referente a uma pernada à bolina e "*dw leg*"a uma outra à popa.

Percebendo que a direção do vento e a sua velocidade variam ao longo do tempo e do espaço de maneiras imprevisíveis identificaram como principais objetivos do problema as seguintes questões:

- Escolha da pernada inicial: o veleiro assim que sai da linha de partida deverá optar por uma pernada a bombordo ou a estibordo do trajeto da regata?
- Quais deverão ser os momentos de guinada?

Tendo em conta que o ângulo ótimo é a diferença angular entre o rumo e o vento verdadeiro no qual a velocidade é máxima, mostra-se que os trajetos mais rápidos devem ser realizados em rumos paralelos às linhas denominadas de *laylines* na 1.2.

O ângulo ótimo é o ângulo que o veleiro tem que fazer com o vento verdadeiro, neste caso à bolina, por forma a alcançar a velocidade máxima possível nessas condições. Depende diretamente das caraterísticas do veleiro e da velocidade do vento.

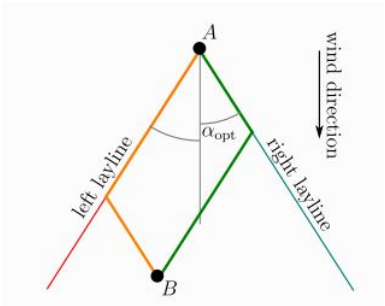


FIGURA 1.2: Laylines definidas tendo em conta o ângulo ótimo.

Os autores centraram-se em dois modelos, o primeiro como "prova de conceito", isto é, mostrar que os métodos de otimização estocástica (resolução estacionária apenas) são capazes de resolver problemas sentidos por velejadores experientes. O segundo modelo é baseado no primeiro, tentativamente mais realista (resolução não estacionária, mais complexa).

1º Modelo

Tendo por base a discretização espacial do campo da regata, os investigadores descreveram o vento recorrendo a uma cadeia de Markov. Tomando valores entre $\pm\gamma$ (direção do vento). Este é um ângulo medido em graus em relação ao norte (onde por convenção o ângulo é medido no sentido horário, de modo que um vento de 0º significa que vem de norte, então um vento de 90º está a vir de este).

Nestas circunstâncias foi postulado que o estado do vento podia alterar-se com probabilidade P após cada passo do processo.

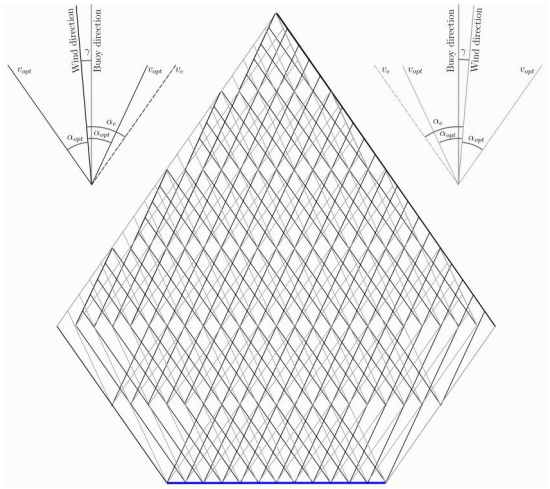


FIGURA 1.3: Campo da regata para $\pm\gamma$

Em cada etapa do processo, há um conjunto finito de ações permitidas a realizar pelo veleiro. De notar que o veleiro navega sempre no ângulo ótimo α_{opt} em relação à velocidade v_{opt} e direção do vento exceto se: o veleiro estiver no exterior de uma *layline*, como por exemplo uma *layline* a bombordo, que corresponde a direções do vento positivas observa-se que muda para direções de vento negativas. Neste caso o veleiro encontra-se fora da linha do vento atual (o que é uma situação desfavorável). O tempo de trajeto de um nó para o outro é calculado pelas v_{opt} e v_e através de cálculos trigonométricos e por fim assumiram que o objetivo do veleiro era então minimizar o tempo despendido para ir até à marca de topo na regata. Para mais detalhes consultar (Dalang, 2010).

2º Modelo

Nesta abordagem a direção e a intensidade do vento forma modeladas recorrendo a cadeias de Markov. Para mais detalhes consultar (Dalang, 2010).

Na Figura 1.4 ilustra-se a segmentação da área da regata em diferentes regiões, que correspondem aos diferentes conjuntos de ações permitidas parametrizadas pelo 1º modelo.

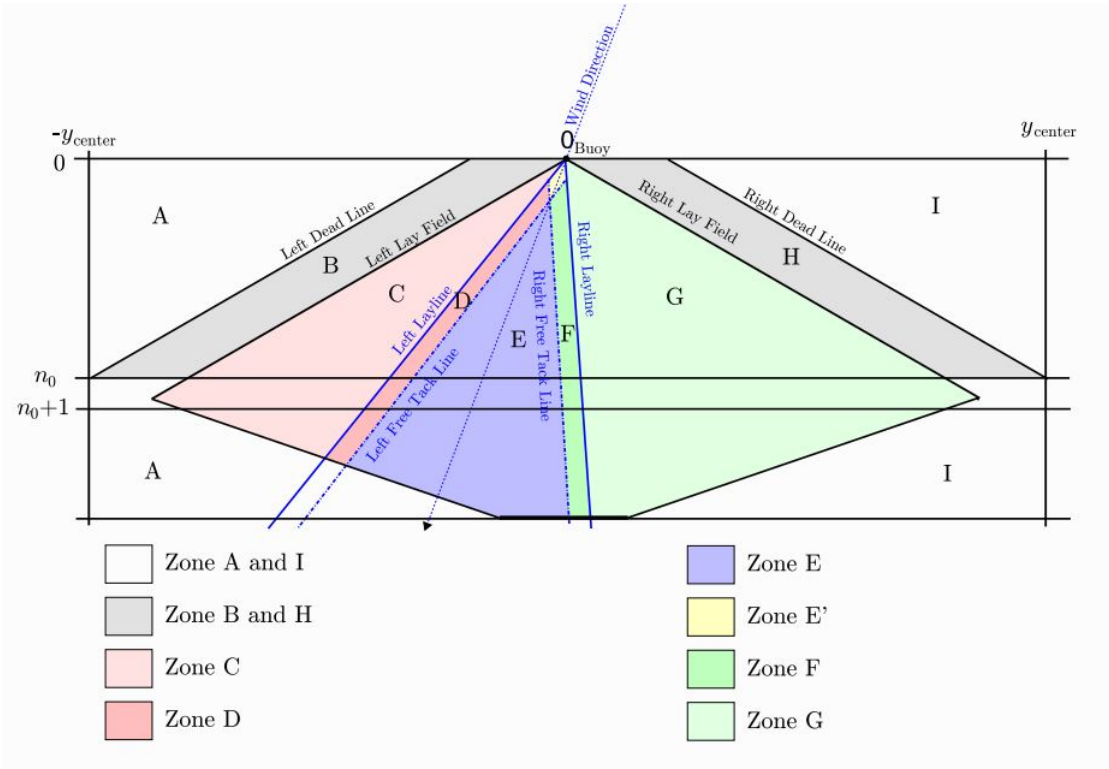


FIGURA 1.4: A segmentação da área da regata de acordo com o conjunto de ações permitidas. Nas zonas A e I os veleiros nunca entram. Nas zonas B e H, só podem dirigir-se diretamente para a marca superior. Nas restantes zonas, podem escolher guinar ou não e seguir um dos rumos disponíveis. Nas zonas D e H, "guinar contra a layline" é permitido. Perto de a marca superior, na zona E1, algumas considerações especiais são necessárias. Na zona E' são necessárias algumas considerações especiais.(Dalang, 2010)

Após modelarem o vento e os restantes parâmetros (ver (Dalang, 2010)) chegaram às seguintes conclusões:

- O modelo fornece diretamente o tempo de viagem estimado $\tau(w, h, x, y)$ até a marca superior sob a otimização programada pelos autores;
- Os dados fornecem a liderança esperada sobre o adversário, desde que a localização do mesmo seja conhecida;

- Numa determinada posição, o veleiro pode considerar não seguir o trajeto ótimo recomendado pelo modelo o que se pode justificar pelas considerações táticas relacionadas com a presença do adversário que o modelo não tem em conta;
- A equipa usou com sucesso o modelo para escolher o melhor rumo durante a primeira etapa da regata;
- Numa ocasião permitiu que o veleiro guinasse no momento ótimo colocando-o em vantagem relativamente ao adversário.

1.4.1.2 Utilização de um algoritmo A* para calculo de trajetórias de um veleiro autónomo

O veleiro autónomo "Avalon"foi construído com o intuito de participar na "*The Microtransat Challenge*", uma regata transatlântica que visa estimular o desenvolvimento e a criação de conhecimento em veleiros autónomos através de uma competição saudável. O veleiro concebido foi equipado com um sistema de comando e controle que incluía o planeamento de navegação. A componente de planeamento da navegação focou-se na resolução, quer do problema oceânico, quer do problema local. Nesta abordagem o vento considerado não é estacionário.

O planeamento local é o que se prende mais com a temática abordada sendo exposto de seguida. Os autores utilizaram um algoritmo A* para identificar o caminho mais rápido para um determinado destino. O A* (lê-se A estrela) é um algoritmo de inteligência artificial [1] usado na pesquisa heurística na procura do caminho ótimo, que neste caso representa o caminho mais curto.

Para cada nó, A* usa uma função heurística que calcula uma estimativa do valor total de um caminho utilizado nesse nó. Funções heurísticas são específicas de cada problema. Estimam o custo do caminho mais acessível do estado atual para o estado final. Matematicamente a função heurística $h(n)$ é o custo estimado do caminho mais económico do nó n até o nó objetivo. Se n é o objetivo, então $h(n) = 0$ (Zanchin, 2018).

As operações que o algoritmo executa encontram-se apresentadas no Anexo I por um fluxograma adaptado de (Zanchin, 2018).

[1] Um algoritmo de Inteligência Artificial consiste no conceito de sistemas inteligentes. Um sistema inteligente é definido como um sistema que incorpora inteligência em aplicações geradas por máquinas, capazes de realizar pesquisa e otimização em conjunto com a capacidade de aprendizagem em simultâneo que executam tarefas complexas de automação que não seriam viáveis por meio da programação tradicional.

O algoritmo começa no ponto inicial, e percorre toda a rede até chegar ao último ponto da mesma. Para cada ponto da rede é atribuído um determinado custo, uma estimativa de quanto tempo demora o veleiro a navegar do ponto inicial até ao ponto final. O caminho ótimo prende-se então com a capacidade de seguir os caminhos que têm o menor peso do inicio ao fim do problema.

Para ter em conta o peso aliado ao movimento do veleiro durante a regata trabalharam em 3 dimensões no espaço de pesquisa onde x e y representam a posição geográfica e θ o rumo. Os autores consideraram assim os seguintes elementos de avaliação (Erckens et al., 2010):

- tempo de navegação desde o 1º nó
- tempo estimado de navegação até ao nó final (heurística);
- penalização da mudança de rumo;
- penalização da manobra;
- penalização de compensação a partir da conexão direta entre o nó inicial e o final;

1.4.1.3 Planeamento dinâmico para um veleiro autónomo

Esta abordagem ao problema de planeamento de trajetos tem como base a transformação das zonas nas quais o veleiro não deve navegar em obstáculos e utiliza um algoritmo de campo potencial para guiar o veleiro até ao destino desejado, evitando assim esses mesmos obstáculos.

Esta técnica constrói uma função potencial, em que os obstáculos têm um potencial repulsivo, ao contrário das metas que têm potenciais atrativos. Sendo que o objetivo é seguir longe dos obstáculos e cada vez mais próximo da reta final ou do ponto de chegada (da Sila, 2011).

O campo potencial foi caracterizado da seguinte forma (Pêtrès et al., 2011):

1. Potencial global: é construído sobre todo o mapa do planeamento que está a ser processado e tem como objetivo direcionar o veleiro para o ponto de chegada e afastá-lo dos obstáculos. Está associado ao ponto de chegada e aos obstáculos.

   Deste modo para encaminhar o veleiro para o ponto de chegada um potencial relativo para o ponto de chegada P_g foi criado em todos os pontos P do mapa, onde G_g é o gradiente desejado e $dist(P, P_g)$ a distância euclidiana entre P e P_g.

Uma vez que G_g é uma constante e o potencial P_g assemelha-se graficamente a um cone centrado em P.

$$P_g = G_g.dist(P, P_g) \tag{1.1}$$

Posteriormente um potencial relativo para os obstáculos P_0 foi construído para cada ponto P do mapa. Onde k é um escalar ajustável. Este potencial tende para infinito à medida que o veleiro se aproxima dos obstáculos o que o impede de colidir com os mesmos.

$$P_0 = \frac{k}{dist(P, P_0))} \tag{1.2}$$

2. Potencial local: é construído na vizinhança da posição do veleiro. É recalculado assim que ocorre uma mudança de posição do veleiro de acordo com o vento ou mudança de rumo. Este potencial local tem em consideração a navegação à:

   - Bolina:
   $$\begin{cases} P_{bolina} = G_{bolina}.dist(P_W, P), 0 < |\theta| < \theta \\ P_{bolina} = 0 \end{cases} \tag{1.3}$$
   P_{bolina} é o potencial referente a velejar à bolina e é construído para cada ponto P_W numa janela centrada na localização do veleiro P. A amplitude de P_{bolina} é dada em função do ângulo θ entre o vetor $\overrightarrow{PP_W}$ e o ângulo do vento verdadeiro (TWA). É linear dentro da zona à bolina (definida pelo ângulo θ_{up} ver Figura 1.5) e nulo fora.

   - Popa:
   $$\begin{cases} P_{popa} = G_{popa}.dist(P_W, P), 0 < |\theta - \pi| < \theta_{popa} \\ P_{popa} = 0 \end{cases} \tag{1.4}$$
   P_{popa} é o potencial referente a velejar à popa e é construído para cada ponto P_W numa janela centrada na localização do veleiro P. A amplitude de P_{popa} é dada em função do ângulo θ e o ângulo do vento verdadeiro (TWA). É linear dentro da zona com vento pela popa (definida pelo ângulo θ_{down} ver Figura 1.5) e nulo fora.

   - Manobra de mudança de bordo:
   $$\begin{cases} P_{manobra} = G_{manobra}.dist(P_W, P), \theta_{bolina} < |\theta| < \pi - \theta_{popa} \\ P_{manobra} = 0 \end{cases} \tag{1.5}$$

$P_{manobra}$ é o potencial que foi tido em consideração para a manobra tanto de orçar como arribar e é construído para cada ponto P_W numa janela centrada na localização do veleiro P. A amplitude de P_{popa} é dada em função do ângulo θ e o último rumo. E onde o $G_{manobra}$ do potencial é parametrizável. Se o veleiro navega com ventos vindos pela esquerda relativamente a ele (ver Figura 1.5), o $P_{manobra}$ é definido pelo lado direito da direção do vento para evitar que o veleiro guine com muita frequência.

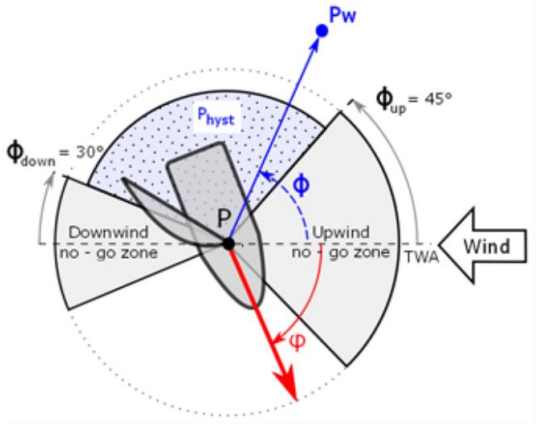


FIGURA 1.5: Esquema da zona à bolina e de vento pela popa onde o veleiro não deve navegar. A zona a ponteado representa a região onde $P_{manobra} \neq 0$. (Pêtrès et al., 2011)

Tendo em vista analisar os resultados simularam um ambiente em que: os obstáculos são estáticos e de posição conhecida. O vento é considerado constante tanto na direção como na velocidade, resolução estacionária do problema. Apesar das restrições do ambiente uma vez que o potencial local é calculado por partes em função da visibilidade do vento e dos obstáculos em redor do veleiro esta abordagem de planeamento dinâmico para um veleiro autónomo consegue ser implementada sem alterações na base do sistema.

Após as simulações, os autores chegaram à conclusão que a presente abordagem fornece trajetórias coerentes tendo em conta as caraterísticas de um veleiro e respeitando as zonas aconselhadas a não navegar pelo programa, tanto à bolina como com vento pela popa (ver Figura 1.6). O número de guinadas pode também ser parametrizado pelo potencial de manobra para uma melhor adequação ao tempo que demoram as mesmas.

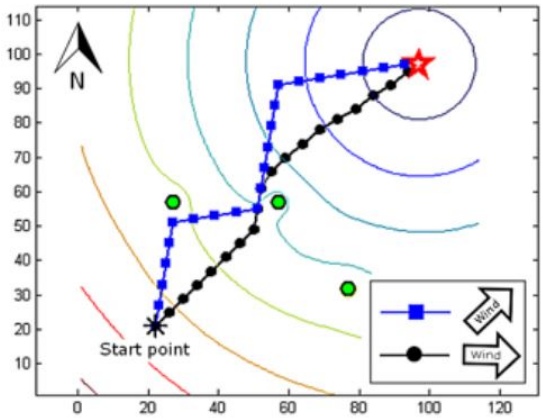


FIGURA 1.6: Trajetos obtidos com ventos vindos de Oeste e Sudeste.

1.4.1.4 Método determinístico multi-objetivo para trajetórias de veleiros

Nesta abordagem, (Marcin Życzkowski, 2017) tiveram em consideração a determinação de trajetos percorridos em tempo mínimo que cumprissem adicionalmente requisitos de segurança e conforto. Para tal criaram uma função objetivo que minimizaram recorrendo ao algoritmo de *Dijkstra*. Este algoritmo permite calcular o melhor trajeto entre dois pontos recorrendo a grafos (estruturas constituídas por um conjunto de pontos chamados vértices, ligados entre si por caminhos designados arestas). Na abordagem do problema as arestas são caraterizadas usando apenas pesos positivos (de Carvalho, 2008).

A função objetivo entra em consideração com os seguintes parâmetros:

→ número de momentos de guinada;

→ número de pernadas do planeamento;

→ tempo em cada pernada;

→ fator de desconforto - adornamento;

→ penalização temporal associada à execução de mudanças de rumo,

→ penalização associada ao desconforto sentido pelos velejadores aquando a mudança de rumo.

De referir que o algoritmo que é utilizado apresenta as seguintes funcionalidades adicionais:

- o peso calculado dinamicamente de cada bordo que tem em conta o adornamento do veleiro e o fator de desconforto associado a ele;

- custos adicionais relativamente a alterações de rumo e penalizações associadas aos momentos de guinada, que são adicionados à soma dos pesos.

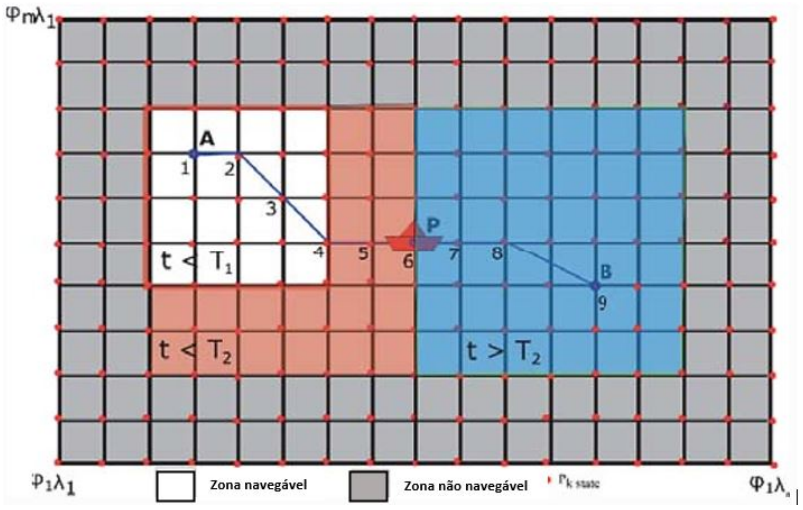


FIGURA 1.7: Rede considerada para três previsões do tempo diferentes tidas em consideração para calcular a melhor rota.

Após várias simulações numéricas concluíram que: o número de guinadas diminui com o aumento do custo de cada mudança de rumo; Concluiu-se também que ter em consideração o critério do conforto dos velejadores resulta no aumento de tempo total da realização do trajeto;

O método descrito fornece resultados dependendo dos valores configurados pelos velejadores tais como o critério do conforto, as previsões meteorológicas, entre outros. Este facto origina planeamentos distintos como seria de esperar.

É importante referir que apenas os trabalhos apresentados em 1.4.1.2 e 1.4.1.4 têm em conta o fator de penalização temporal nas mudanças de rumo e que nenhum apresenta as correções do rumo a seguir "em tempo real".

1.4.2 Aplicações comerciais

As aplicações comerciais dedicadas à otimização de rotas podem classificar-se nos seguintes tipos tendo em conta as variáveis que consideram:

- Meteorológicas
- Posição GPS em tempo real
- Correntes e marés

Apresentaremos de seguida as principais aplicações que têm em vista planear trajetos tendo em conta as caraterísticas do veleiro:

Aplicações	Pago	Android & iOS	Posição GPS	Condições METOC	Cartas Náuticas NOAA	Calculador de Rota	Características do Veleiro
iNavX	√	√	√	-	√	√	-
iSailGPS	√	**iOS**	√	-	√	√	-
NAVIONICS	√	√	√	√	√	√	-
SeaNav	√	**iOS**	√	√	√	√	-
Transas iSailor	√	√	√	√	√	√	-
Marine Navigator	√	**Android**	√	-	√	√	-
qtVlm	√	√	√	√	√	√	√

TABELA 1.1: Aplicações comerciais

1.4.2.1 iNavX

Nesta aplicação, o utilizador tem a possibilidade de escolher, dentro da base de dados da aplicação, qual a carta que melhor se adequa ao trajeto que vai praticar. E consequentemente se a pretende vizualizar através de um gráfico Raster ou Vetoriais[2]. Apesar de ter funcionalidades tais como, criar pontos de passagem, rotas, ferramentas de medição entre pontos, acesso ao GPS, análise de correntes, a aplicação não possui nenhuma ferramenta de cálculo de tempo de trajeto («iNavX», 2019).

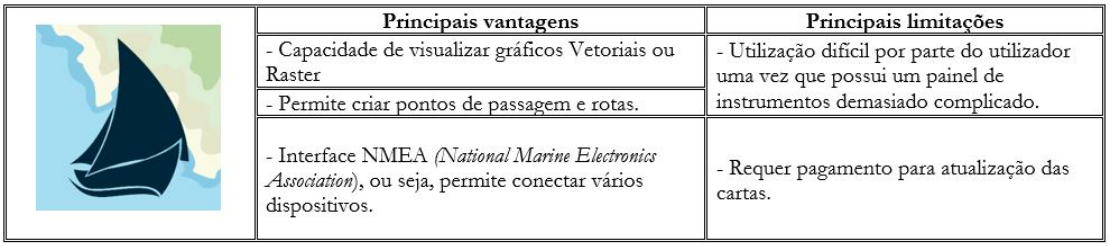


FIGURA 1.8: Vantagens e Desvantagens iNavX

[2]Forma de codificação de gráficos. Formados por pixeis, pequenos pontos que, de acordo com diferentes cores e tonalidades compõem a imagem. Os gráficos vetoriais por outro lado, são formados por pontos, linhas e curvas. Podem ser redimensionados sem perder qualidade.

1.4.2.2 iSailGPS

Utiliza a base de dados *NOAA(Nautical Oceanic and Atmospheric Administration)*. Foca o acompanhamento do veleiro dando ao utilizador informação de para onde está a ir, onde está e onde já esteve. Possui ferramentas de seleção de cartas, medição de distâncias e ainda uma ferramenta que, quando o utilizador cria um ponto de passagem a aplicação, fornece a distancia do veleiro ao ponto, o rumo, a velocidade do veleiro e ainda o tempo estimado até ao ponto de passagem (Inc., 2021).

	Principais vantagens	Principais limitações
	-Exibe a rota, velocidade, rumo e posição geográfica exata na parte superior do ecrã.	-Apenas iOS.
	-Projetado para navegar em tempo real através de dados GPS.	
	-Adequado para navegação costeira.	

FIGURA 1.9: Vantagens e Desvantagens iSailGPS

1.4.2.3 NAVIONICS

Providência, ao utilizador, cartas náuticas em detalhe para vários tipos de atividades náuticas, como pesca, vela, mergulho, entre outros. Fornece uma detalhada representação do fundo do mar e de lagos em vários tons de azul de modo a identificar as diferentes profundidades assim como a possibilidade de ter acesso à meteorologia e marés. A partir do AIS apresenta em tempo real outras embarcações. Permite traçar planeamentos, medir distâncias e aplicar marcas em pontos escolhidos pelo utilizador. Apesar de todas as suas funcionalidades esta aplicação também não apresenta nenhuma ferramenta que permita calcular o tempo mínimo de trajeto entre pontos de passagem (S.r.l., 2021).

	Principais vantagens	Principais limitações
	- Detalhada representação do fundo do mar e de lagos em vários tons de azul.	- As regiões e ferramentas extras são vendidos separadamente.
	-Criação e edição de rotas diretamente.	
	- Rotas e pontos de passagem podem ser transferidos via Wireless.	

FIGURA 1.10: Vantagens e Desvantagens NAVIONICS

1.4.2.4 SeaNav

Navega e planeia usando cartas NOAA (EUA) e cartas UKHO (Irlanda e Reino Unido). Os gráficos são atualizados conforme se vai visualizando o mapa. Permite planear rotas e segui-las em tempo real uma vez que possui GPS. Visualização de bóias, luzes e navios em realidade aumentada. Utiliza também o AIS para permitir a visualização de outras embarcações. Apesar de apresentar ao utilizador a velocidade em tempo real e rumo da embarcação também não tem ferramenta especifica para previsão de tempos de trajeto (P. M. Ltd, 2014).

	Principais vantagens	Principais limitações
	-Oferece uma visão aprimorada dos objetos circundantes (realidade aumentada).	- Apenas funciona com iOS
	- Ficheiros das cartas atualizados automaticamente.	

FIGURA 1.11: Vantagens e Desvantagens SeaNav

1.4.2.5 Transas iSailor

À semelhança das aplicações anteriores a Transas iSailor apresenta ao utilizador a posição atual do veleiro, velocidade, rumo, rotas definidas pelo mesmo, sendo que é uma aplicação mais dirigida para embarcações à vela (W. V. Ltd, 2010).

	Principais vantagens	Principais limitações
	-Aplicação para velejadores com pouca experiência, sendo que possui uma interface extremamente percetível e um manual de utilizador.	- A aplicação para descarregar não requer pagamento, contudo para aceder a todas as suas funcionalidades especificas da mesma requer.
	-Possível conectar com *smartwatch* e visualizar através do mesmo as condições, velocidade e direção, do vento.	

FIGURA 1.12: Vantagens e Desvantagens Transas iSailor

1.4.2.6 Marine Navigator

Apresenta, à semelhança das aplicações anteriores, a direção real do deslocamento de um navio, entre dois pontos, em relação à superfície da terra e permite criar rotas. Dependendo da localização da embarcação os dados apresentados são atualizados automaticamente via GPS. Dispõe as cartas náuticas no ecrã e as funcionalidades anteriormente descritas. Possui uma funcionalidade chamada *Compass Up* que permite, ao apontar o telemóvel ou *tablet* para a proa, a aplicação trace

no ecrã uma linha do rumo verdadeiro podendo quem está a tomar conta do leme perceber quanto o veleiro está a ser afetado pela corrente (Koenig, 2018).

	Principais vantagens	Principais limitações
	-Ferramenta "*Compass Up*".	-Apenas compatível com dispositivos Android.
	-O utilizador pode adicionar rotas e pontos de passagem.	

FIGURA 1.13: Vantagens e Desvantagens Marine Navigator

1.4.2.7 qtVlm

Programa de navegação desenhado para embarcações à vela uma vez que tem em consideração o diagrama polar do veleiro. Apresenta também as condições meteorológicas. Tendo a possibilidade de trabalhar com vários formatos de cartas. Inclui módulo AIS e um módulo de instrumentos que permite recuperar dados de várias fontes NMEA(*National Marine Electronics Association*) ou GPS interno. Mostra ao utilizador dados de desempenho do veleiro como velocidades praticadas e trajetos efetuados em navegações anteriores (Meltemus, 2017).

	Principais vantagens	Principais limitações
	- Calcula o trajeto usando modelos de rotas e rotas de tempo.	- É gratuito para Windows, MacOS, Linux e Raspberry e também para Android e iOS contudo a versão completa tem que ser paga.
	- Entra com as características específicas do veleiro, como diagrama polar.	
	- Trabalha com diferentes tipos de cartas.	

FIGURA 1.14: Vantagens e Desvantagens qtVlm

Verificou-se que a **qtVlm** com os dados das características do veleiro e utilizando as cartas de tempo que recolhe em tempo real, assim como a posição, determina a rota que minimiza o tempo de trajeto para chegar ao destino. Calcula muitas informações úteis em tempo real, como a seleção de velas e a velocidade de avanço na direção pretendida, que é um termo bastante utilizado na vela principalmente em regatas que indica a velocidade do veleiro na direção do vento (*Velocity Made Good*).

Contactado um membro da equipa da produção da aplicação, este informou que "A aplicação para viagens de longa distância com base no algoritmo da rota

e condições climáticas futuras (arquivos *gribs*[3] que contêm dados de previsão do tempo como vento, correntes, ondas, pressão, entre outros) para criar as rotas utiliza um método personalizado das isócronas, e seguidamente um método que utiliza o resultado anterior e o otimiza"[4](Philippe Lelong, 2020) tendo em vista obter a minimização do tempo de trajeto.

A qtVlm é ainda capaz de, com os dados de vento em tempo real, compensar a diferença entre a situação real do vento e os dados encontrados nas gribs, contudo para avaliar decisões de longo prazo, apenas usa dados grib.

[3]Uma "Grib"é um formato de dados conciso normalmente utilizado em meteorologia para armazenar dados históricos e meteorológicos de previsão do tempo

[4]Tradução livre da autora. No original: *"For long distance sailing using real-time data is not enough to optimize the course of a boat, you also need to take into account future weather conditions, based on routing algorithm. qtVlm does that based on gribs (files containing forecast weather data like wind, currents, waves, pressure, CAPE, etc. There are 2 modules exploiting that, one is the routing module using a customized isochrone method, and the second one is the route module which normally takes the result of a routing and re-optimize it, but can also be used independently"*(Philippe Lelong, 2020)

Capítulo 2

Enquadramento Teórico

2.1 Física da vela

Neste sub-capitulo vamos de uma forma sintética abordar as bases da vela tendo em vista melhor enquadrar o problema em estudo.

2.1.1 Vento aparente e Vento Verdadeiro

A generalidade das pessoas já verificou que a ação do vento sobre si depende não só das caraterísticas do vento em si, como também do movimento do observador. O mesmo acontece a velejar. Com efeito, é possível relacionar entre si três velocidades que atuam simultaneamente. A velocidade do vento verdadeiro, $\vec{v}$, que é a velocidade do vento em relação à agua, a velocidade do vento aparente, $\vec{v}_{ap}$, que é a velocidade do vento medida no veleiro em movimento e ainda a velocidade do veleiro em si, $\vec{V}$, medida relativamente à superfície da água (Kimball, 2010).

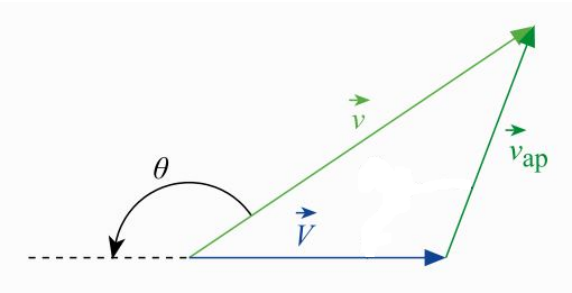


FIGURA 2.1: Relação entre as velocidades: verdadeira do vento (verde-claro),do vento aparente(verde-escuro) e a velocidade do veleiro (azul) (e Sá, 2005).

De notar que o ângulo θ, entre 0^{o} e 180^{o}, carateriza a marcação de entrada do vento no navio. Se θ for nulo ter-se-á vento de proa. Se θ for 180^{o} ter-se-á vento de popa.

2.1.2 Forças Aplicadas a um Veleiro

Apoiada no trabalho de (Afonso, 2016) sistematizou-se quais o tipo de forças que sujeitam um veleiro. Primeiramente há que ter em conta que estas forças são aerodinâmicas e hidrodinâmicas uma vez que, o movimento realizado pelo veleiro é a combinação entre estas duas, ou seja, existe a força exercida pelo vento sobre a vela e também a força exercida pela água nas obras vivas da embarcação.

As forças hidrodinâmicas requerem uma modelação complexa e o estudo das mesmas está fora no âmbito deste trabalho preferindo-se assim apenas modelar de forma simples as forças aerodinâmicas de modo a entender como o vento se relaciona com a propulsão do veleiro.

(Afonso, 2016) afirma que as "forças aerodinâmicas, resultam da integração da pressão e da força viscosa na superfície da vela."As forças aerodinâmicas podem ser decompostas em:

- Força de arrasto (F_{AA}) com sentido do vento aparente;
- Força de sustentação (F_{SA}) perpendicular ao vento aparente;

Seguidamente apresentaremos a sustentação do efeito dessas forças.

Embarcação à vela à bolina

A navegação à bolina carateriza-se por navegar com vento cujo o ângulo de entrada é menor do que 90º. Esta forma de navegar recorrendo a rotas em "zigue-zague"permite progredir na direção de objetivos que se encontram a barlavento. A análise das forças envolvidas permite verificar a impossibilidade de uma embarcação navegar diretamente contra o vento (sem fazer bordos), uma vez que nessa situação a força de sustentação tem componente perpendicular à linha proa-popa e a de arrasto o sentido do vento.

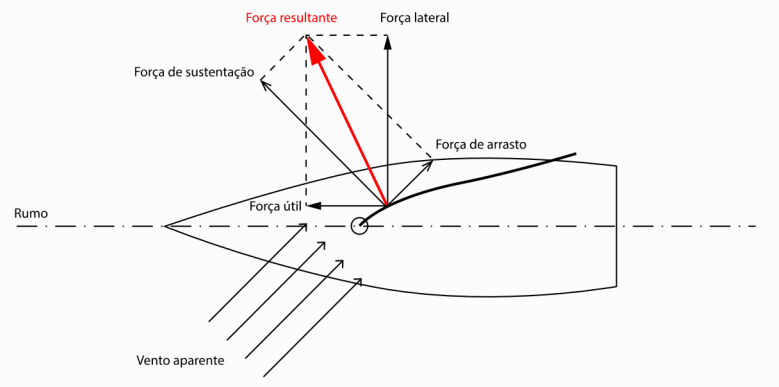


FIGURA 2.2: Representação das forças aerodinâmicas presentes num veleiro à bolina

Embarcação à vela com vento pela popa

Pode mostrar-se nestas circunstâncias que a força útil praticamente coincide com a força de arrasto. Veja-se a Figura 2.3. Pode observar-se também que a força de sustentação é quase nula. Neste caso é possível justificar a impossibilidade de navegar com uma velocidade superior à do vento real (Afonso, 2016).

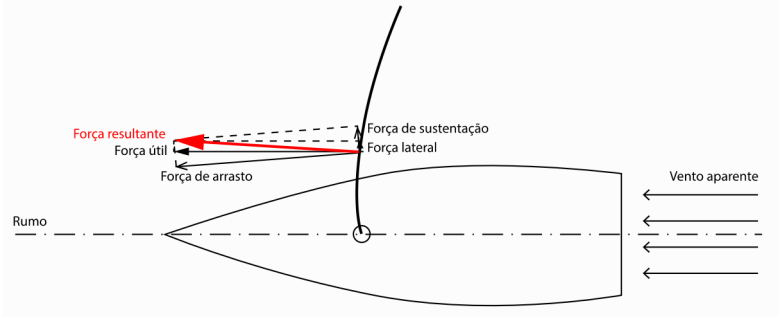


FIGURA 2.3: Representação das forças aerodinâmicas presentes num veleiro com vento pela popa

Embarcação à vela com vento pelo través

Com esta marcação, a componente da força aerodinâmica resultante na direção e sentido do rumo do veleiro, pode ser muito significativa podendo originar velocidades superiores à própria velocidade do vento (e Sá, 2005).

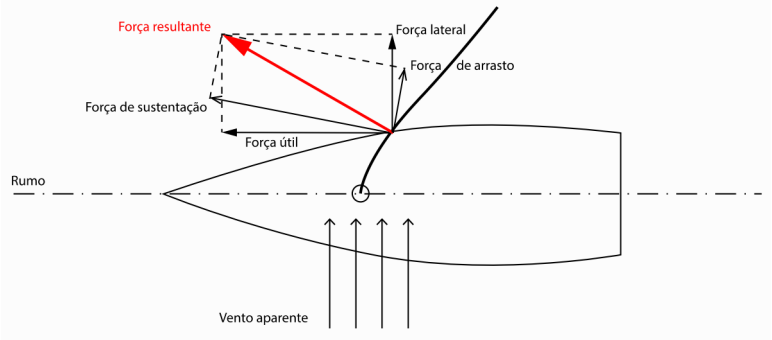


FIGURA 2.4: Representação das forças aerodinâmicas presentes num veleiro com vento pela través

2.1.3 Diagrama Polar

Um diagrama polar de velocidade é um gráfico que permite determinar a velocidade estabilizada dum veleiro em função da velocidade do vento verdadeiro e do seu ângulo de entrada na embarcação. Uma vez que um veleiro é construído de forma simétrica o seu diagrama polar também o é. Desta forma os diagramas polares apresentam apenas a informação relativa a ângulos de entrada do vento verdadeiro entre 0º e 180º. Na Figura 2.5 pode observar-se um diagrama polar.

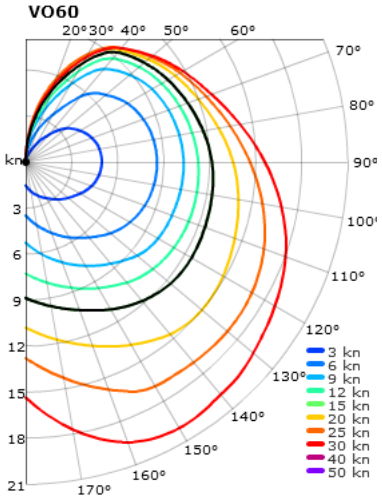


FIGURA 2.5: Diagrama Polar de um VO60, classe de veleiros utilizada na *Volvo Ocean Race*

2.2 Otimização

A otimização é um conceito/processo especialmente importante em muitas áreas da atividade humana e biológica. Caracteriza-se por alcançar o melhor ou o

mais favorável (mínimo ou máximo) de uma determinada situação (Kulkarni et al., 2017).

Genericamente a otimização consiste em descobrir num espaço de procura, determinado pelo problema, o máximo ou o mínimo de uma determinada função objetivo que à medida que procura essa otimização, responde também às restrições impostas por esses mesmos problemas. Pode traduzir-se matematicamente por

$$\text{Função Objetivo} \quad f(X) \quad \text{com,} \tag{2.1}$$

$$\text{s número de restrições de desigualdades} \quad g_j(X) \leq 0, \quad j = 1, 2, ..., s \tag{2.2}$$

$$\text{w número de restrições de igualdade} \quad h_j(X) = 0, \quad j = 1, 2, ..., w \tag{2.3}$$

$$\text{onde o número de variáveis é definido por} \quad x_i, \quad i = 1, 2, ..., n \tag{2.4}$$

$$\text{ou pelo vetor} \quad X = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \tag{2.5}$$

Os diferentes métodos de otimização podem classificar-se em métodos exatos e heurísticos. Seguidamente apresentaremos genericamente os métodos acabados de referir:

2.2.1 Métodos exatos

Os métodos exatos são algoritmos de pesquisa exaustiva que verificam todo o conjunto de soluções de um determinado problema até encontrar a solução ótima, garantindo a mesma. Contudo, o facto de ser uma pesquisa exaustiva faz aumentar a sua complexidade e a sua aplicação a problemas também mais complexos, culmina em diversas dificuldades para encontrar a solução ótima num curto espaço de tempo (Junior & Cechin, 2006). É uma abordagem que só pode ser utilizada para problemas com poucas instâncias, para serem em tempo considerado útil, eficientes.

2.2.2 Métodos heurísticos

Os algoritmos heurísticos têm como objetivo produzir soluções boas num período de tempo razoável, apoiando-se do conhecimento sobre a estrutura e variáveis do problema, para combater o elevado tempo que demora a procura de uma

solução ótima (de Castro Honorato Silva, 2013). Adotam muitas vezes uma estratégia intuitiva onde exploram não todas as soluções possíveis mas apenas parte delas. Não garantem que a solução apresentada é a ótima, porém apresentam uma boa solução num tempo considerado curto.

2.2.3 Métodos meta-heurísticos

Estes são algoritmos que encontram boas soluções e por vezes a ótima. Têm por base uma heurística subordinada que se adequa a cada problema específico. As suas principais caraterísticas consistem em evitar as desvantagens dos métodos anteriores, tendo a capacidade de explorar um conjunto de soluções onde se evitam os ótimos locais e são encontradas soluções em tempo útil(de Castro Honorato Silva, 2013). Pode-se ainda dividir os algoritmos meta-heurísticos em dois tipos (Blum & Roli, 2003):

♢ Os que alcançam uma solução única a cada iteração, exemplos destes são:

- *Simulated Annealing* (SA) , Simulação do Recozimento (SR)
- *Tabu Search* (TS) , Pesquisa Tabu
- *Greedy Randomized Adaptive Search Procedure* (GRASP) , Procedimento de pesquisa aleatória ambiciosa
- *Iterated Local Search* (ILS) , Pesquisa local Iterada
- *Variable Neighborhood Search* (VNS) , Pesquisa de Vizinhança Variável

♢ E os que alcançam uma população de soluções em cada iteração:

- *Evolutionary Algorithms* (EA), Algoritmos Evolucionários
- *Genetic algorithm* (AG), Algoritmos Genéticos
- *Scatter Search* (SS), Pesquisa Dispersa
- *Adaptive Memory Procedures* (AMP), Procedimentos de memória adaptativa

A SR surgiu da analogia feita por Kirkpatrick (Anjo, 1994) e Černy quando relacionaram o processo físico metalúrgico de recozimento e a resolução de problemas de otimização. O processo físico é composto por dois momentos chaves: primeiramente eleva-se a temperatura a que o sólido está submetido de maneira a que ele atinja o seu ponto de fusão; de seguida diminui-se lentamente a temperatura de modo que as partículas que constituem o solido se disponham numa rede cristalina

teórica, no estado de energia mínima. As soluções em problemas de otimização combinatória assemelha-se aos estados do sistema físico descrito e a função custo está associada à energia de cada estado.

A simulação do recozimento tem a particularidade de apresentar uma implementação intuitiva, despojada de complexidades e que configura uma abordagem eficaz a problemas de otimização não lineares em domínios de pesquisa complexos (Gonzalez et al., 2007).Esta técnica, devidamente parametrizada e calibrada, constitui assim um método de optimização global que permite determinar boas soluções.

2.3 Sistemas e Equipamentos

Atualmente os sistemas de comando e controlo de numerosos veículos autónomos estão a ser implementados utilizando software *Python*™ instalado em plataformas *Rasperry Pi*. Este tipo de plataforma é não só relativamente poderosa para executar algoritmos complexos como também é de dimensões (veja-se a Figura 2.6) apropriadas à sua instalação em embarcações autónomas como aquelas que se utilizam presentemente e planeiam vir a utilizar na Escola Naval (com mais de 1 m de comprimento).

FIGURA 2.6: Ilustração do Raspberry Pi 4 que se pretende utilizar futuramente. Dimensões: 56mm de largura por 85 mm de comprimento

O Rasperry Pi 4 mais recentemente desenvolvido e representado na Figura 2.6 possui um sistema de otimização de energia, uma entrada para a Ethernet (permitindo a ligação a redes locais) assim como capacidade de Wireless e Bluetooth, um processador *quad core* de 64bits e duas entradas USB 3.0 e USB 2.0. Dispõe ainda de duas entradas Micro HDMI que possibilitam a visualização em dois dispositivos com uma resolução 4K. Dependendo do modelo oferece opção de escolha de RAM entre 2GB, 4GB e 8GB. Não obstante é possível inserir também um cartão Micro-SD

para armazenar o sistema operativo como também os dados recolhidos durante as operações (FOUNDATION, s.d.).

De referir que a linguagem de programação *Python*™ está presentemente disseminada pela comunidade científica e universitária. Com efeito, esta é uma linguagem bastante versátil e popular que atualmente suporta tanto a programação orientada a objetos como a programação estruturada podendo operar tanto em sistemas *Windows, MacOS, Linux*, entre outros. Sendo um dos principais méritos desta o facto ser projetada para ser legível, de fácil manutenção e com suporte avançado para mecanismos de reutilização de *software*. Conta também com uma comunidade a nível mundial extremamente ativa e é *Open Source*, ou seja, é desenvolvido abertamente para o público/comunidade, podendo todas as pessoas, vê-lo, modifica-lo e distribuí-lo conforme as suas necessidades.

Pelas razões referidas e tendo em vista a possibilidade de se vir a utilizar os algoritmos desenvolvidos em sistemas de comando e controlo ancorados na plataforma Raspberry Pi, procederemos à transcrição para a linguagem *Python*™ (ver Apêndice C) as rotinas de cálculo dos trajetos de tempo mínimo originalmente desenvolvidos em *MATLAB*®

Acrescentamos que a execução dos algoritmos desenvolvidos, para em tempo real determinar os trajetos para chegar ao destino em tempo mínimo, exigem que a embarcação disponha de equipamentos para aceder, em tempo real, à velocidade verdadeira do vento, à velocidade da embarcação e à sua posição geográfica. Para tal, torna-se necessário dispor de equipamentos que permitam, direta ou indiretamente, obter as informações referidas tendo em vista o seu processamento pelos algoritmos desenvolvidos. A programação desta funcionalidade (comunicação-equipamentos-algoritmo) será transferida para trabalho futuro.

Os equipamentos que tipicamente são utilizados para se aceder em tempo real à velocidade verdadeira do vento, à velocidade da embarcação e à sua posição geográfica são:

- Recetor GPS
- Anemómetro
- Odómetro

Recetor GPS

É um sistema de navegação por satélites que fornece a um recetor informações de localização e tempo real de um equipamento/veículo independentemente das condições meteorológicas. A taxa de erro relativamente à localização obtida depende do numero de satélites visíveis na determinada posição. O sistema possuiu uma constelação de 4 a 24 satélites para através da distância entre eles efetuarem uma triangulação obtendo a posição pretendida (El-Rabbany, 2022).

Figura 2.7: Recetor GPS.

Anemómetro

Os anemómetros são instrumentos que têm como objetivo medir a velocidade de fluidos e em casos específicos a sua direção. Neste caso a medição da velocidade do ar prende-se com fatores meteorológicos mais especificamente o vento (Almeida, 2004). Este equipamento permite fornecer aos algoritmos a velocidade relativa do vento a partir da qual se pode determinar a velocidade verdadeira do vento (conhecendo a velocidade da embarcação).

FIGURA 2.8: Anemómetro.

Odómetro

Este equipamento permite fornecer aos algoritmos a velocidade relativa da embarcação. A velocidade obtém-se a partir das diferenças entre a pressão normal (pressão estática) da água e a pressão resultante do movimento do navio através da água (pressão dinâmica). Quanto maior for a velocidade do navio sobre a água, maior será a diferença entre as duas pressões, maior a velocidade.

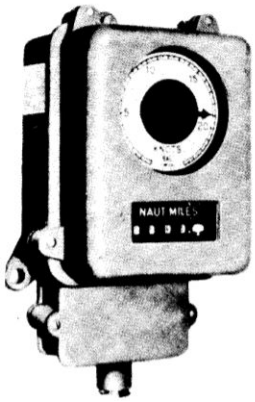


FIGURA 2.9: Odómetro.

Capítulo 3

Formulação do Problema e Desenvolvimento dos Algoritmos

3.1 Formulação do Problema

Pretende-se minimizar o tempo de trajeto por parte de um veleiro, entre dois pontos admitindo que o vento é uniforme e estacionário e as correntes nulas. Conhecido o campo de velocidade do vento, o tempo de trajeto T entre os pontos A e B, pode ser calculado recorrendo ao integral de linha:

$$T = \int_{\Gamma} \frac{ds}{v} \tag{3.1}$$

em que Γ representa o trajeto adotado por parte do veleiro e v representa o valor absoluto da velocidade em cada ponto do trajeto. O valor absoluto da velocidade do veleiro, $v = v\left(\mathbf{r}, \frac{\dot{\mathbf{r}}}{\|\dot{\mathbf{r}}\|}\right)$ dependerá da posição $\mathbf{r}$ do veleiro e também da direção do vetor tangente $\frac{\dot{\mathbf{r}}}{\|\mathbf{r}\|}$, à trajetória Γ adotada. A dependência referida, entre a posição $\mathbf{r}$ do veleiro e também da direção do vetor tangente $\frac{\dot{\mathbf{r}}}{\|\mathbf{r}\|}$, à trajetória Γ, é regulada pela características do veleiro e expressa normalmente pelo diagrama polar da velocidade do mesmo. Refira-se que o diagrama polar de velocidade dum veleiro fornece a sua velocidade para diferentes rumos, velocidade esta associada ao ângulo θ de entrada do vento verdadeiro no veleiro.

A minimização de 3.1 é um problema que não pode ser resolvido recorrendo a técnicas elementares de otimização uma vez que a função a minimizar depende de uma infinidade de variáveis (cada uma das coordenadas que permitem definir o caminho Γ). Trata-se dum problema do âmbito do cálculo variacional[5] que poderia

[5]Família particular de problemas de otimização caracterizados por estarem definidos em espaços (funções) de dimensão infinita e pelo facto da função objetivo ser descrita por um operador integral.(Leitão, 2001)

ser abordado neste contexto recorrendo ao princípio do tempo mínimo de Fermat (Casaca & Henriques, 2004). A abordagem pela via variacional permitiria obter um sistema de equações diferenciais não lineares, as equações de Euler-Lagrange, cuja solução representa o caminho ótimo Γ.

Outra possível abordagem deste problema seria a utilização de técnicas da geometria diferencial através das quais o trajecto ótimo seria uma geodésica apropriadamente determinada (da Rocha, 2020).

As metodologias anteriormente descritas não são numericamente expeditas uma vez que exigem na determinação de cada trajeto ótimo, a resolução numérica das equações diferenciais não lineares (as equações de Euler-Lagrange ou as equações geodésicas). Por outro lado, estas técnicas deixam de fora possíveis trajetos ótimos constituídos pela simples concatenação de trajetos retilíneos, os quais integram soluções ótimas do problema na presença de campos uniformes e estacionários de velocidade do vento.

Assim, na abordagem ao problema descrito, decidiu-se utilizar uma técnica heurística de otimização inspirada no método de simulação do recozimento.

Na aplicação da técnica desenvolvida o trajeto do ponto $A = (x_A, y_A)$ para o ponto $B = (x_B, y_B)$ é modelado através da concatenação, dum certo número, de segmentos orientados $\bar{L}_i$, $i = 1, ..., N$ (pernadas) que ligam os pontos indicados e que são perfeitamente definidos pelos correspondentes $N-1$ pontos de guinada (eventual mudança de rumo $G_i = (x_i, y_i)$, $i = 1, ..., N-1$ (vértices):

$$\bar{L}_i = G_1 - A, \tag{3.2}$$

$$\bar{L}_i + 1 = G_{i+1} - G_i, i = 1, ..., N-2 \tag{3.3}$$

$$\bar{L}_N = B - G_{N-1} \tag{3.4}$$

Conhecido o campo de velocidade do vento (suposto uniforme e estacionário) e representado por $t = t(\bar{L})$ a função que estima o tempo despendido pelo veleiro na realização do percurso com a orientação descrita pelo segmento $\bar{L}$, o tempo total de trajeto T associado à definição dos pontos de guinada $G_i, i = 1, ..., N-1$, será:

$$T(G_1, G_2, ..., G_{N-1}) = \sum_{i=1}^{N} t(\bar{L}_i) \tag{3.5}$$

A aplicação do método inspirado no recozimento simulado terá assim, como objetivo, minimizar a função 3.5 num domínio geográfico que contenha as localizações $\{A, G_1, G_2, ..., G_{N-1}, B\}$ suficientemente afastadas da respetiva fronteira. Uma solução boa será constituída pela sequência $\{A, G_1, G_2, ..., G_{N-1}, B\}$ de localizações de pontos de guinada que definem um tempo de trajeto próximo do tempo mínimo global.

A metodologia de otimização aplicada, baseou-se no estabelecimento aleatório de uma população apropriada de diferentes trajetos (cada trajeto é uma sequência ordenada de localizações de pontos de guinada) cujos pontos de guinada são sucessivamente perturbados (de época para época). Após cada perturbação são preservados para a época seguinte, não só os melhores trajetos, como também um certo número de trajetos menos favoráveis. O facto de época para época se preservarem trajetos menos favoráveis permitirá iludir a ocorrência de mínimos locais. De época para época a perturbação (aleatória) dos pontos de guinada de cada um dos trajetos preservados ocorre com uma menor amplitude.

Claramente esta metodologia apresenta algumas analogias com o método da Simulação do Recozimento. Em primeiro lugar, a sucessão de perturbações das posições dos pontos de guinada, cuja amplitude diminui lentamente de época para época, reproduz a estratégia de lenta descida ("recozimento") de "temperatura" aplicada no método de Simulação de Recozimento. Em segundo lugar, a aceitação de um certo número trajetos menos favoráveis em cada época, reproduz o mecanismo aleatório de aceitação, ou não, de configurações menos favoráveis que é aplicada no método de Simulação do Recozimento, recorrendo a uma regra probabilística.

Na Figura3.1 ilustramos o tempo de trajeto do melhor representante em cada época determinado com o algoritmo desenvolvido. A observação do gráfico permite observar a tendência de convergência do processo.

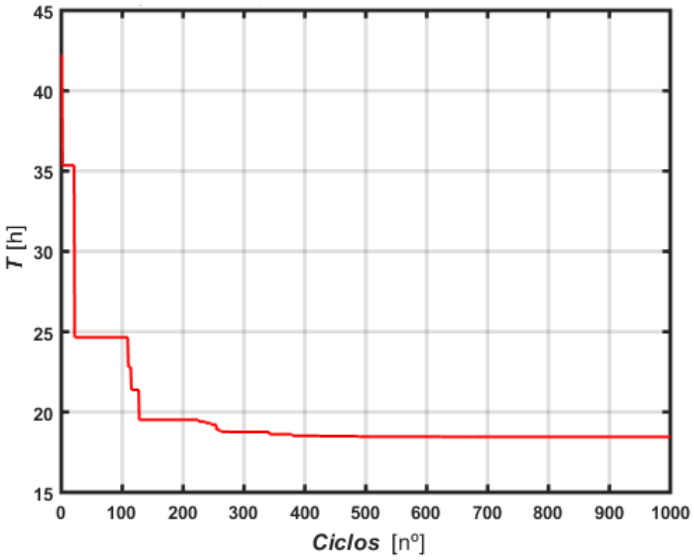


FIGURA 3.1: Evolução do tempo mínimo com o decorrer dos ciclos

Na Figura 3.2 reproduzimos o fluxograma (Fedorchuk, 2020) que descreve genericamente a metodologia apresentada:

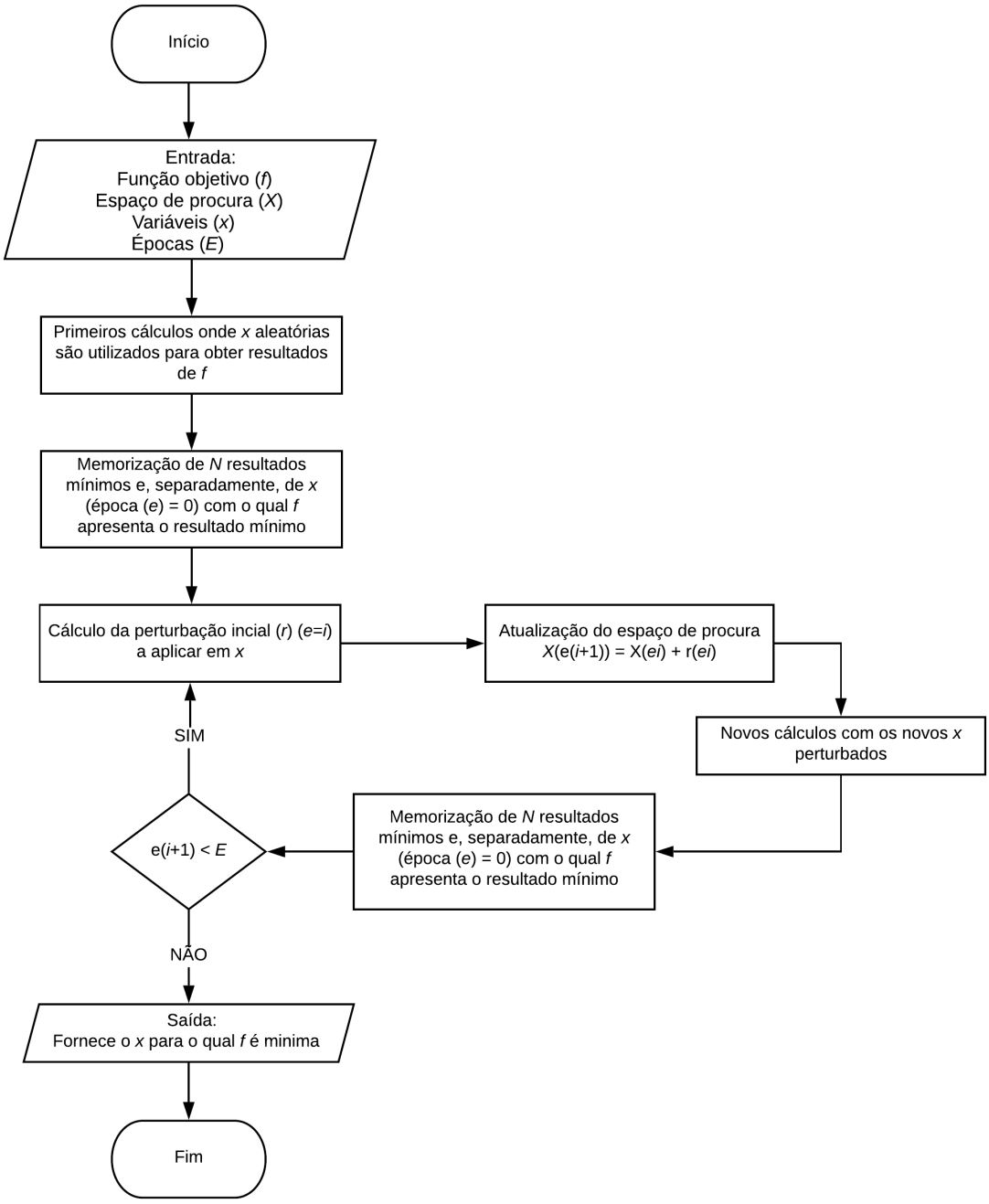


FIGURA 3.2: Fluxograma de como funciona a Simulação de Recozimento.

Uma das particularidade que se acrescentou ao algoritmo inicial foi a criação de um valor de penalização, consoante o trajeto realizado pelo veleiro associado aos pontos de guinada, devido às mudanças de bordo uma vez que a mudança de rumo, nos pontos de guinada, é normalmente acompanhada de perda de velocidade por parte da embarcação à vela.

Ao ato de mudar de bordo com o vento pela proa designa-se virar por D'avante, com o vento pela popa Cambar, tendo as duas o mesmo objetivo: mudar de rumo. A primeira passando a proa pela linha do vento a segunda passando a popa (Ferreira & Martins, 2002) (manobra utilizada quando à bolina a proa passa a linha do vento por forma que a direção de onde sopra o vento mude de um bordo para o outro). Esta manobra provoca uma diminuição da velocidade do veleiro pois a vela ao passar pela linha meia nau (a linha imaginária que vai da proa à popa e divide o veleiro em duas partes iguais) faz com que a vela deixe de ficar preenchida pelo vento e fica a "bater pano"por momentos. O veleiro diminui a velocidade nessa chamada zona morta, até ser de novo colhido pelo vento, no bordo oposto. (Francisco, s.d.)

A inclusão duma penalização temporal associada às mudanças de rumo efetuados nos pontos de guinada é um ingrediente importante na obtenção de trajetos ótimos mais realistas recorrendo ao algoritmo desenvolvido. Tal inclusão terá necessariamente como consequência a obtenção de trajetos que irão dispor os pontos de guinada de forma a "minimizar"as mudanças de rumo durante o trajeto de A para B.

A penalização temporal aplicada a cada ponto de guinada terá as seguintes características:

- Aplicar-se-á apenas a trajetos à bolina que envolvam mudança do bordo de entrada do vento;
- Será proporcional ao valor do ângulo total α de guinada ($0 < \alpha < 180$ graus);
- Será inversamente proporcional à velocidade média de velocidade verificada nas pernadas anteriores ao ponto de guinada considerado.

A função penalizadora a adotar terá as seguintes características:

- Deverá incluir o comportamento anteriormente referido;
- Deverá ser simples e facilmente parametrizável recorrendo a parâmetros calibráveis analiticamente e /ou experimentavelmente.

Sem prejuízo de futuramente se adotar uma função penalizadora temporal experimentalmente ajustada, neste trabalho adotar-se-á a seguinte função penalizadora temporal criada pela autora (em horas),

$$t = k_1 e^{-k_2 V} \frac{\alpha}{90} \tag{3.6}$$

em que α representa o ângulo total de guinada ($0 < \alpha < 180$) em graus, V representa a velocidade média nas pernadas adjacentes, em nós e k_1 e k_2 representam duas constantes para calibração do modelo que será postulado consoante as caraterísticas especificas de cada veleiro.

A escolha dos parâmetros k_1 e k_2 postulados para o veleiro Alba, com 30m de comprimento e o NE Sagres, com 89m de comprimento e correspondentes velocidades máximas de 11 nós e 16 nós, impõem penalizações temporais que se situam entre os 5 segundos a 1 minuto 45 segundos e 11 segundos e 3 minuto 36 segundos, respetivamente para guinadas de 90^{o} (mudança total de rumo em graus) realizadas, de modo respetivo, à velocidade média de $V \approx 2$ nós e $V \approx 17$ nós.

onde,

α - Ângulo total de guinada ($0 < \alpha < 180$) em graus;

V - Velocidade média nas pernadas adjacentes, em nós;

Os cálculos que justificam a parametrização das constantes k_1 e k_2 apresentam-se justificados no apêndice A para o veleiro Alba e no apêndice B para o veleiro NE Sagres.

A penalização a aplicar a cada ponto de guinada originará trajetos mínimos mais realistas.

3.2 Resolução computacional - Programação

No desenvolvimento dos algoritmos destinados a calcular trajetos que permitam a um veleiro num tempo mínimo chegue ao seu destino (com condição de vento uniforme e estacionário), foi utilizada a linguagem de programação *MATLAB*®. O Programa principal é designado por OTV.m (acrónimo de Otimização de Trajetórias de Veleiros) o que faz a gestão dos diferentes sub-rotinas utilizadas.

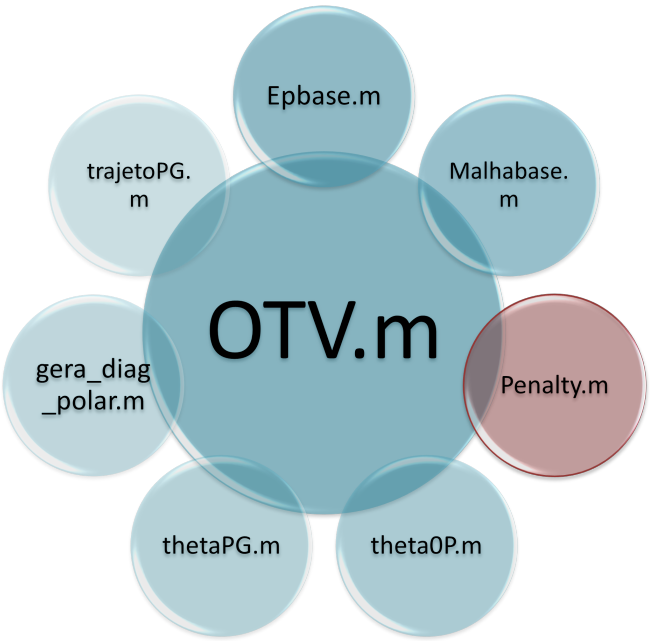


FIGURA 3.3: Mapa mental da estrutura de como funciona o programa

No programa principal, o utilizador pode definir certos parâmetros iniciais, tais como: *Ponto Inicial, Ponto Final, Penalização, Velocidade do Vento, Azimute para Onde Sopra o Vento, Passos (define uma distância de referência "resolução"para gerar a resolução espacial média do espaço de pesquisa) e Número de Pontos de Guinada.* De referir a possibilidade do utilizador definir igualmente o número de iterações (épocas) que o processo de otimização de trajetos envolverá.

Terminados os cálculos, o programa apresenta gráficos de controlo com o ponto inicial e final, pontos de guinada, o trajeto a realizar pelo veleiro, o tempo de trajeto, a direção de onde sopra o vento e a sua velocidade. Imprime ainda um gráfico de evolução do tempo mínimo dos melhores trajetos determinados em cada época.

O *MATLAB*® é uma ferramenta poderosa que utiliza um software de cálculo de matrizes para a resolução de diversos problemas numéricos. Permite desenvolver um programa principal (OTV.m) que coadjuvado com diversas funções criadas resolvem o problema inicial. No âmbito desta dissertação foram criadas sete funções referidas na Figura 3.3, as quais vão ser descritas de seguida.

3.2.1 Criação do Espaço de Pesquisa

A sub-rotina Epbase.m gera uma coleção de trajetos definidos pelo numero de pontos de guinada previamente definido. O número de trajetos gerados pode ser

ajustado pelo utilizador.

Posto isto, criou-se o espaço de pesquisa consoante o número de pontos de guinada que estamos a considerar, sendo que para isso os pontos de guinada são aleatoriamente gerados tendo em conta as matrizes construídas.

```
function [MXep, MYep, dAB, rAB, resolucao, dimEP]=EPbase(A,B,passos,numPG)
% Criação das matrizes com as coordenadas X (MXep)
% e coordenadas Y (MYep) dos pontos de guinada:
%
% Em cada linha da coluna j encontra-se a coordenada  X
% do ponto de guinada j.
%
% Em cada linha da coluna j encontra-se a coordenada Y
% do ponto de guinada j.
% dAB-distância de A a B
% rAB-rumo de A a B (coordenadas polares [-180 a +180]


dAB=pdist2(A,B,'euclidean');

rAB=atan2d(B(2)-A(2),B(1)-A(1));

resolucao=dAB/passos;

dimEP=2*(passos^2); % Dimensão do espaço de pesquisa (número par)

menorX=min(A(1),B(1));
maiorX=max(A(1),B(1));
menorY=min(A(2),B(2));
maiorY=max(A(2),B(2));

MXep=[];
MYep=[];
for i=1:numPG

    Xbase=(menorX-dAB)+((maiorX+dAB)-(menorX-dAB))*rand(dimEP,1);
    Ybase=(menorY-dAB)+((maiorY+dAB)-(menorY-dAB))*rand(dimEP,1);
    MXep=[MXep Xbase];
    MYep=[MYep Ybase];
end
end
```

FIGURA 3.4: Código que permite a criação do Espaço de Pesquisa

3.2.2 Cálculos para gerar trajetos

Esta sub-rotina, utilizando os ponto de guinada, calcula a distância e o rumo entre o ponto A, os pontos de guinada criados, e o ponto B. Neste caso: distância $A - P_1$; distância de $P_i - P_{i+1}$; distância de $P_n - B$ assim como os rumos correspondentes. Os dados são guardados em forma de matriz, `MdPG` para as distâncias e `MrPG` para os rumos tal como mostra a Figura 3.5:

```
function [MdPG,MrPG]=trajectoPG(MXep,MYep,A,B)
% dA_P1, ..., dPi_Pi+1,...,dPN_B
%
% MdPG-Matriz das distâncias
% MrPG-Matriz dos rumos

% Obs: Os rumos são cálculados em
% em coordenadas polares [-180 a +180]
[m,n]=size(MXep);
%m-número de trajectos (dimensão do espaço de pesquisa)
%n-número de pontos de guidada

Amx=ones(m,1)*A(1); % Vectores coluna com as coordenadas de A e B
Amy=ones(m,1)*A(2); %
Bmx=ones(m,1)*B(1); %
Bmy=ones(m,1)*B(2); %

MXep=[Amx MXep Bmx];
MYep=[Amy MYep Bmy];

MdPG=zeros(m,n+1);
MrPG=zeros(m,n+1);

for i=1:n+1
    dPG=sqrt((MXep(:,i)-MXep(:,i+1)).^2+(MYep(:,i)-MYep(:,i+1)).^2);
    MdPG(:,i)=dPG;
    rPG=atan2d(MYep(:,i+1)-MYep(:,i),MXep(:,i+1)-MXep(:,i));
    MrPG(:,i)=rPG;

end
```

FIGURA 3.5: Código para calcular as distâncias e rumos entre pontos de guinada.

3.2.3 Ângulos de entrada do vento em cada trajeto

Esta sub-rotina calcula os ângulos `theta` de entrada no diagrama polar para computação das velocidades verificadas nas pernadas entre os pontos de guinada.

```
function [thetaAB, adosv]=theta0P(rAB,aposv)
% rAB rumo de A para B em coordenadas polares [-180 a +
% aposv-Azimute Para Onde Sopra o Vento
% adosv-Azimute De Onde Sopra o Vento
% thetaAB Ângulos de entrada no diagrama polar


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Conversão de "aposv" (0[N] a 360) em "adosv" [-180 0[
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aposv_cp=atan2d(cosd(aposv),sind(aposv));% Azimute apos
                                         % coordenadas
if aposv_cp >= 0
    adosv=aposv_cp-180;
else
    adosv=aposv_cp+180;
end

thetaAB=acosd(cosd(rAB)*cosd(adosv)+sind(rAB)*sind(ados
end
```

(A) Theta0P

```
function [MthetaPG]=thetaPG(MrPG,aposv);
% MrPG e rPB rumos em coordenadas polares [-180 a +180]
% aposv-Azimute Para Onde Sopra o Vento
% adosv-Azimute De Onde Sopra o Vento
% MthetaPG- Ângulos de entrada nos diagramas polares (das pernadas)

[m,n] = size(MrPG); % Determina as dimensões da matriz MrPG

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Conversão de "aposv" (0[N] a 360) em "adosv" [-180 0[E] +180[
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aposv_cp=atan2d(cosd(aposv),sind(aposv));% Azimute aposv em
                                         % coordenadas polares
if aposv_cp >= 0
    adosv=aposv_cp-180;
else
    adosv=aposv_cp+180;
end

adosv=ones(m,n)*adosv;

MthetaPG=acosd(cosd(MrPG).*cosd(adosv)+sind(MrPG).*sind(adosv));

end
```

(B) ThetaPG

FIGURA 3.6: Cálculo dos correspondentes ângulos `theta` de entrada no diagrama polar para computação das velocidades verificadas nas pernadas entre os pontos de guinada.

3.2.4 Implementação dos diagramas polares dos veleiros utilizados nos cálculos

Esta sub-rotina implementa os diagramas polares dos veleiros utilizados nos cálculos. A implementação é realizada por interpolação do vento tendo por base a velocidade conhecida do vento e o diagrama polar do veleiro.

A função gerada por esta (`gera_diag_polar`) é uma função polinomial de grau 1 definida por ramos associada ao ângulo de entrada do vento.

```
function [diag_polar, x, y]=gera_diag_polar_sagres(vento)
% Edifica o pp (piecewise polynomial) cprrespondente ao vento verdadeiro,
% diag_polar, que será usado nos calculos massivos das velocidades

% SAGRES POLAR

%Ângulos de entrada do vento
x=[0, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180];
%Velocidades do veleiro consoante velocidade do vento
if 0 <= vento & vento < 5
    y5=[0,0.2,0.8,2.5,4.5,5.0,5.8,4.2,2.5,2.3,2.0,1.7];
    y=y5*vento/5;
elseif vento < 10
    y5=[0,0.2,0.8,2.5,4.5,5.0,5.8,4.2,3.4,3.0,2.5,1.7];
    y10=[0,0.3,1.0,4.6,6.3,7.1,7.7,6.3,4.8,3.5,2.5,2.2];
    y=y5+(vento-5)*(y10-y5)/5;
elseif vento < 15
    y10=[0,0.3,1.0,4.6,6.2,7.1,7.7,6.3,4.8,3.5,3.2,2.2];
    y15=[0,0.4,1.2,5.0,7.4,8.5,9.2,7.5,6.0,4.2,3.5,2.5];
    y=y10+(vento-10)*(y15-y10)/5;
elseif vento < 20
    y15=[0,0.4,1.2,5.0,7.4,8.5,9.2,7.5,6.0,4.2,3.5,2.5];
    y20=[0,0.5,1.4,6.0,8.5,10.1,11.0,8.5,6.5,5.0,4.0,2.9];
    y=y15+(vento-15)*(y20-y15)/5;
elseif vento < 25
    y20=[0,0.5,1.4,6.0,8.5,10.1,11.0,8.5,6.5,5.0,4.0,2.9];
    y25=[0,0.6,1.6,6.4,9.5,11.0,12.1,9.6,7.3,5.4,4.5,3.2];
    y=y20+(vento-20)*(y25-y20)/5;
elseif vento < 30
    y25=[0,0.6,1.6,6.4,9.5,11.0,12.1,9.6,7.3,5.4,4.5,3.2];
    y30=[0,0.7,1.7,6.6,10.0,11.4,12.5,10.0,7.4,5.6,4.8,3.3];
    y=y25+(vento-25)*(y30-y25)/5;
elseif vento < 35
    y30=[0,0.7,1.7,6.6,10.0,11.4,12.5,10.0,7.4,5.8,4.8,3.3];
    y35=[0,0.8,1.8,7.2,10.4,12.1,12.7,10.4,7.5,6.0,5.0,3.5];
    y=y20+(vento-30)*(y35-y30)/5;
else
    return
end
%Interpolação final que fornece a velocidade do veleiro consoante a
%velocidade do vento
%diag_polar = spline(x,y);

diag_polar = interp1(x,y,'linear','pp');
end
```

FIGURA 3.7: Cria o diagrama polar com base nos dados deste tipo de diagrama fornecido pelo veleiro, neste caso para o NE Sagres.

3.2.5 Utilizada na visualização do vento

Esta sub-rotina destina-se a gerar a visualização gráfica da direção do vento como se ilustra na Figura 3.8.

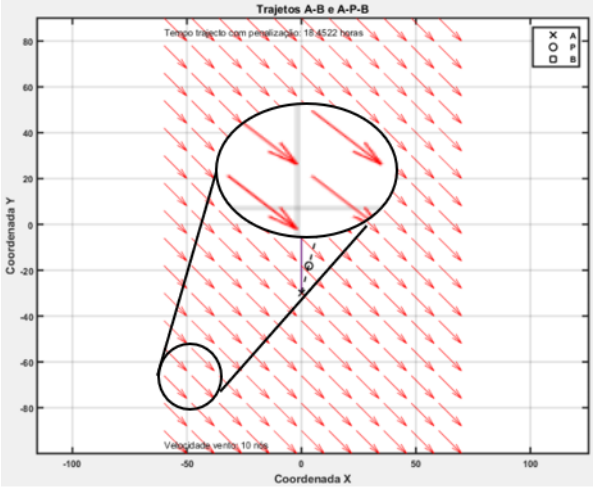


FIGURA 3.8: Setas representativas do vento

```
function [Xbase, Ybase, dAB, rAB, resolucao]=malhabase(A,B,passos)
% dAB- distância de A a B
% rAB- rumo de A a B (coordenadas polares [-180 a +180])


dAB=pdist2(A,B,'euclidean');
rAB=atan2d(B(2)-A(2),B(1)-A(1));
resolucao=dAB/passos;
menorX=min(A(1),B(1));
maiorX=max(A(1),B(1));
menorY=min(A(2),B(2));
maiorY=max(A(2),B(2));

vectorX=menorX-dAB:resolucao:maiorX+dAB;
vectorY=menorY-dAB:resolucao:maiorY+dAB;

[Xbase,Ybase] = meshgrid(vectorX,vectorY);
end
```

FIGURA 3.9: Código para gerar a malha base que é posteriormente utilizada para visualização do fator vento.

3.2.6 Penalização temporal relativa aos pontos de guinada

Esta sub-rotina calcula e atribui tempos de penalização associados à amplitude e caraterísticas das mudanças de rumo nas mudanças de pernada.

Seguidamente apresentamos o significado e os principais valores assumidos pelas principais variáveis utilizadas nos diferentes programas desenvolvidos:

`MrPG` - Rumos em coordenadas polares [-180 a +180][m x (n+1)] ;

`MthetaPG` - Ângulos de entrada nos diagramas polares (das pernadas)[m x (n+1)] ;

`aposv` - Azimute Para Onde Sopra o Vento,

`vel_PG` - Velocidades em cada pernada de A a P a B, [m x (n+1)].

E criados uns novos:

`penalty_PG`- Penalização em cada ponto de guinada;

`BEV`- Bordo Entrada de Vento;

`BOL`- Bolina;

`BEVDIFF`- Se = 0 não há mudança de bordo do ponto de guinada, se = 1 existe mudança de bordo do ponto de guinada;

`BOLSUM` - Matriz que traduz se existe bolina ou não em torno o ponto de guinada;

`MATPENAL` - Matriz das penalizações associadas às guinadas com mudança de bordo em bolina. Se MATPENAL (I,J)=1 há penalização ;

`ANGVAR` - Cálculo do ângulo total de variação de rumo [ANGVAR] ,

`VELMED` - Cálculo da velocidade média em em torno de cada ponto de guinada.

```
function [penalty_PG,BEV,BOL, BEVDIFF, BOLSUM, MATPENAL, ANGVAR, VELMED]=penalty(MrPG,
MthetaPG, vel_PG, aposv,pen)
% CALCULA A PENALIZAÇÃO A APLICAR EM CADA PONTO DE GUINADA
% ESTA FUNÇÃO É PROPRIETÁRIA DO VELEIRO PARA O QUAL O DIAGRAMA POLAR ESTÁ A
% SER UTILIZADO. AS PENALIZAÇÕES DUM ALBA POLAR SERÃO DIFERENTES DO NE
% SAGRES.
% OBS: Se pen == 1 aplica o modelo de penalização, caso contrário
% penalty_PG= 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if pen == 1
[m,n] = size(MrPG); % Determina as dimensões da matriz MrPG

k1=20/1852; % Coeficiente k1 de penalização do modelo
k2=log(20)/20; % Coeficiente k2 de penalização do modelo

% Cálculo do bordo de entrada do vento [MrPG e aposv] em cada pernada
% BEV-Matriz bordo entrada vento
% starbord == +1
% port == -1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Conversão de "aposv" (0[N] a 360) em "adosv" [-180 0[E] +180[
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aposv_cp=atan2d(cosd(aposv),sind(aposv));%Azimute aposv em coordenadas polares
if aposv_cp >= 0
    adosv=aposv_cp-180;
else
    adosv=aposv_cp+180;
end

adosv=ones(m,n)*adosv;
TEMP=MrPG-adosv;
%Computação da matriz BEV bordo de entrada do vento
BEV=TEMP;
[row1,col1] = find(BEV>=-360 & BEV<-180);% +1
[row2,col2] = find(BEV>=-180 & BEV<0);% -1
[row3,col3] = find(BEV>=0 & BEV<180);% +1
[row4,col4] = find(BEV>=180 & BEV<=360);% -1

BEV(row1,col1)=1;
BEV(row2,col2)=-1;
BEV(row3,col3)=1;
BEV(row4,col4)=-1;

BEVDIFF=abs(diff(BEV,1,2)/2);
% Se BEVDIFF=0 não há mudança de bordo do ponto de guinada
% Se BEVDIFF=1 não mudança de bordo do ponto de guinada
%%%%%%%%
% Cálculo da mareação em cada pernada (bolina ou não) [MthetaPG]
% se MthetaPG(i,j) <= 95 é bolina na pernada j do trajecto i
% caso contrário, não .
```

```
%Computação da matriz de bolina BOL
% Bolina == +1, caso contrário Bolina == 0
BOL=MthetaPG;

[row1,col1] = find(BOL>=0 & BOL<95);% +1
BOL=0*BOL;
BOL(row1,col1)=1;

BOL1=BOL(:,2:end);
BOL2=BOL(:,1:end-1);

BOLSUM=BOL1+BOL2;
[row1,col1] = find(BOLSUM>=1.5 & BOLSUM<2.5);
BOLSUM(row1,col1)=1;
% Se BOLSUM  == 0 não há bolinas em torno do ponto de guinada
% Se BOLSUM == +1 há pelo menos umabolina bolina em torno do ponto de guinada

% Matriz MATPENAL das penalizações associadas
% às guinadas com mudança de bordo em bolina
% Se MATPENAL (I,J)=1 há penalização

MATPENAL=floor((BOLSUM+BEVDIFF)/2);

% Cálculo do ângulo total de variação de rumo [ANGVAR]
%  em cada ponto de guinada [MrPG]

MrPG_temp1=MrPG(:,2:end);
MrPG_temp2=MrPG(:,1:end-1);

ANGVAR=floor(abs(MrPG_temp1-MrPG_temp2)/2.5);

% CÁLCULO DA VELOCIDADE MÉDIA VELMED EM TORNO DE CADA
% PONTO DE GUINADA

vel_PG_temp1=vel_PG(:,2:end);
vel_PG_temp2=vel_PG(:,1:end-1);

VELMED=(vel_PG_temp1+vel_PG_temp2)/2;

penalty_PG=k1*exp(-k2*VELMED).*MATPENAL.*ANGVAR/90;
else
    penalty_PG=0;
    BEV =0 ;
    BOL = 0;
    BEVDIFF = 0;
    BOLSUM = 0;
    MATPENAL = 0;
    ANGVAR = 0 ;
    VELMED = 0;

end

return
```

FIGURA 3.10: Como foi estruturada a função penalizadora em código para cada ponto de guinada

Capítulo 4

Simulações Numéricas e Discussão dos Resultados

Neste capítulo, tendo em vista testar e analisar o programa desenvolvido, efetuaremos e discutiremos simulações numéricas. Foram utilizados dois veleiros, o Alba (A) e o NE Sagres (B) que têm caraterísticas diferentes. O NE Sagres é uma barca que possui pano redondo e latino. O Alba é um veleiro que enverga apenas pano latino.

O algoritmo de penalização anteriormente discutido permitir-nos-á observar a diferença que existe no tempo de um trajeto realizado com ou sem penalização para que futuramente o programa escolha o caminho de tempo mais reduzido traduzindo-se por ótimo. Foram executadas simulações tanto com penalização como sem. Teve-se, ainda em conta, o fator mais preponderante de todos, o vento e para cada trajeto correu-se a simulação 20 vezes, as quais foram distribuídas equitativamente por ventos constantes de 10nós, 15nós, 20nós e 25nós, de modo a observar as diferenças e a resposta do programa aos mesmos.

(A) Alba (B) NE Sagres

FIGURA 4.1: Veleiros utilizados para simulações numéricas do programa

Tanto para o veleiro Alba como para o NE Sagres foram simulados cinco tipo de trajetos diferentes:

- Trajetos realizados contra o vento;
- Trajetos realizados à bolina;
- Trajetos realizados com vento pelo través;
- Trajetos realizados a um largo,
- Trajetos realizados com vento pela popa.

Contudo, apenas serão discutidos os trajetos realizados **contra o vento** e **à bolina** uma vez que são os trajetos os quais a função penalizadora claramente demonstra a sua ação. A estruturação deste capitulo será a seguinte:

No sub capitulo 4.1 será analisada e comentada a robustez do programa e os seus tempos de cálculo.

No sub capitulo 4.2 serão analisados os trajetos **SEM** penalização e **COM** penalização tanto contra o vento como à bolina para o veleiro **Alba**. No final será apresentado um gráfico para visualização dos dados numéricos das simulações e posterior análise. O processo será o mesmo para o sub capitulo 4.3 mas para o veleiro **NE Sagres**.

Condições Iniciais

Em todas as simulações o ponto de partida localizar-se-á na posição com as coordenadas A (0,0) e o ponto de chegada estará 10 milhas a norte, isto é, estará na posição com coordenadas B(0,10). Faremos simulações com três pontos de guinada (quatro pernadas) e também simulações com cinco pontos de guinada (seis pernadas) de modo a ter comparação entre os mesmos. A dimensão do espaço de pesquisa será ajustada tendo em conta o número de pontos de guinada referidos. Alteraremos apenas a direção do vento para onde sopra o vento - **Aposv** relativamente a cada trajeto.

4.1 Robustez e Tempos de Cálculo dos Programas Desenvolvidos

4.1.1 Robustez do Programa

Para demonstrar a robustez do programa realizaram-se diversas simulações, neste caso com o veleiro Alba, com vento de 045, vento de Norte, vento de 315 e vento de popa nas mesmas circunstâncias.

Os gráficos dos trajetos ótimos obtidos encontram-se no Apêndice D e os valores dos tempos ótimos de trajeto obtidos em horas na Tabela 4.1 seguinte:

	Tempo trajeto (h)			
	Vento de 045	**Vento Norte**	**Vento de 000**	**Vento de 315**
	2,6688	3,7722	2,0076	1,8516
	2,6708	3,7722	2,0076	1,8516
	2,6705	3,7769	2,0076	1,8516
	2,6681	3,7769	2,0076	1,8516
	2,6681	3,7735	2,0076	1,8516
	2,6696	3,7738	2,0076	1,8516
	2,6712	3,7737	2,0076	1,8516
	2,6675	3,7739	2,0076	1,8516
	2,6691	3,7744	2,0076	1,8516
	2,6706	3,7769	2,0076	1,8516
	2,6712	3,7724	2,0076	1,8516
Média:	**2,6696**	**3,7743**	**2,0076**	**1,8516**
Desvio Padrão:	**0,0012**	**0,0017**	**0**	**0**

Tabela 4.1: Tempo trajeto em horas para analise de robustez.

Podemos observar que os trajetos obtidos (ApêndiceD) são equivalentes e que os tempos ótimos de trajeto obtidos apresentam uma média de 2,6696(h); 3,7746(h); 2,0076(h); 1,8516(h) e correspondentes desvios padrões de 0,0012(h); 0,0017(h); 0 e 0. O desvio padrão carateriza a dispersão dos valores obtidos em relação à média.

O quociente entre o desvio padrão e a média amostral fornece-nos uma estimativa do erro relativo delta, também respetivamente: 0,000461; 0,000447; 0 e 0, associado à computação do trajeto ótimo que consideramos satisfatório.

Observa-se assim que os trajetos calculados apresentam pouca variabilidade. Podemos concluir que o programa é robusto.

4.1.2 Tempos de Cálculo dos Programas Desenvolvidos

Para realizar os cálculos do tempo de CPU (Unidade Central de Processamento), esforço de cálculo exigido pelo programa (não estão contemplados a componente gráfica e de gravação no disco das variáveis de ambiente) utilizou-se um computador pessoal ©ASUS A560U com um processsador Intel®Core™i5-8250U CPU a 1.60GHz e com 8GB RAM.

	Tempo de CPU(s) para 3 pontos de guinada	
	COM penalização	SEM penalização
Vento Norte	0,8153	0,4571
Vento de 045	0,9796	0,4583
Vento de 090	0,9264	0,4488
Vento de 135	0,7284	0,4526
Vento de 180	0,6557	0,4565

Tabela 4.2: Esforço de cálculo para 3 pontos de guinada.

	Tempo de CPU(s) para 5 pontos de guinada	
	COM penalização	SEM penalização
Vento Norte	1,4647	0,5192
Vento de 045	1,5064	0,5233
Vento de 090	1,5207	0,4985
Vento de 135	1,0542	0,5033
Vento de 180	0,91122	0,5169

Tabela 4.3: Esforço de cálculo para 5 pontos de guinada.

Foram simulados trajetos para comparar tempos de cálculo com três e cinco pontos de guinada para ventos de Norte, de 045, de 090, de 135 e de 180. Tanto com e sem penalização. Todos os trajetos foram simulados com vento a 15 nós e cada tipo de vento foi simulado dez vezes. As Tabelas 4.2 e 4.3 ilustram a média

dos valores das dez simulações (os valores individuais encontram-se no Apêndice E) e demonstram que tanto num caso como outro o programa, quando aplica a função penalizadora, exige um esforço superior de cálculo, neste caso o dobro de quando o programa corre sem a função penalizadora.

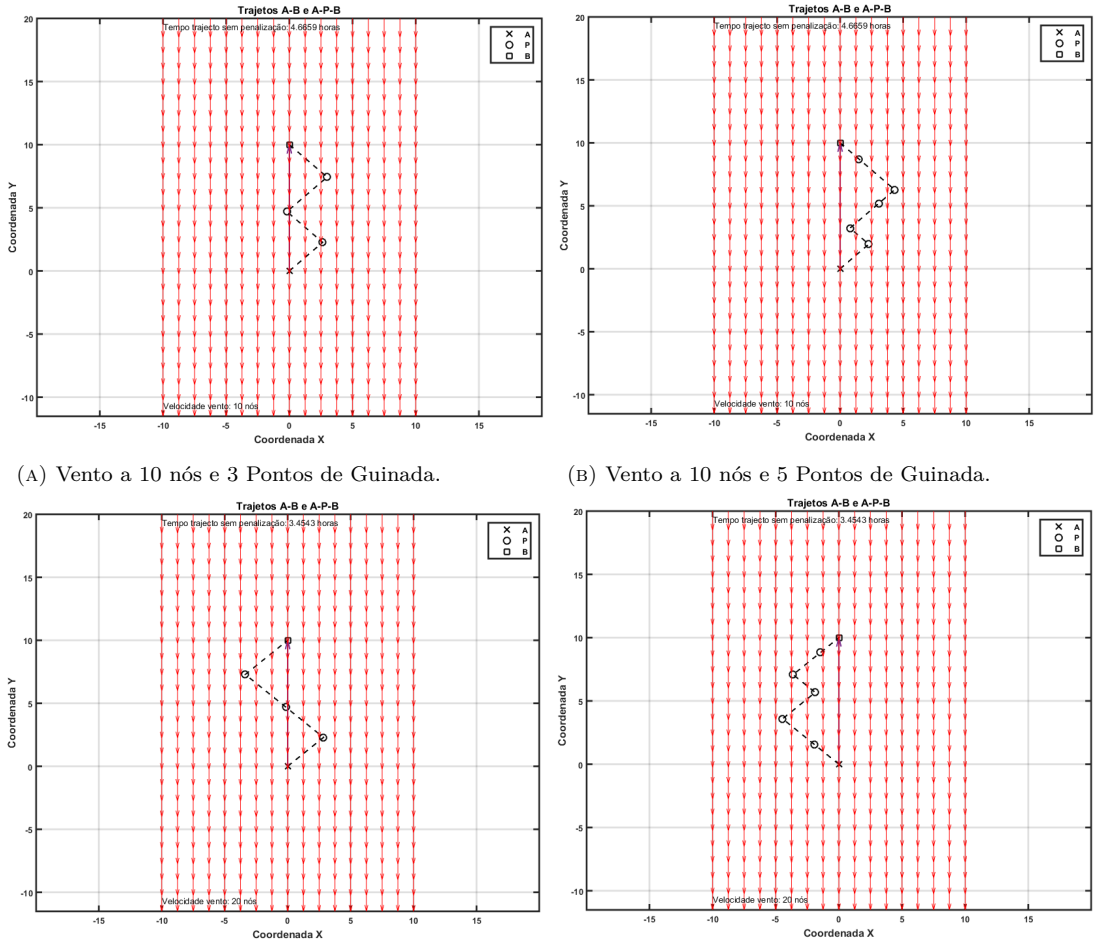
4.1.3 Cálculos em Tempo Real

Como calculado anteriormente, o esforço de cálculo é entre 0,5 segundos a 2 segundos, o que permite a possibilidade de se efetuarem ajustes em "tempo real". Ou seja, quando os dados forem recolhidos através de sensores, o programa tem a capacidade de recalcular o trajeto para as condições sentidas no momento sem demorar muito tempo útil. Esta opção pode ser utilizada em circunstancias como regatas de vela robótica.

4.2 Trajetos Veleiro Alba

4.2.1 Simulações com vento Norte

SEM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

(D) Vento a 20 nós e 5 Pontos de Guinada

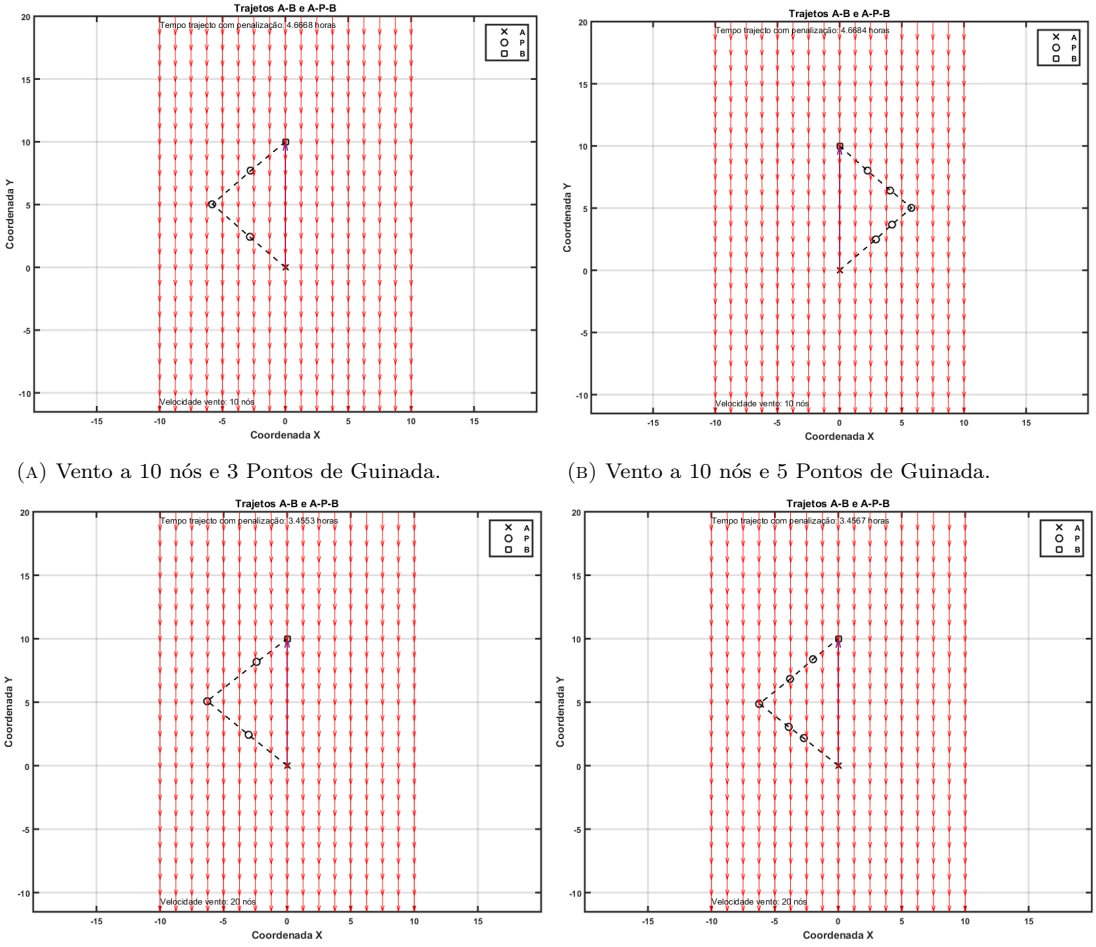
FIGURA 4.2: Trajetos do Veleiro Alba realizados contra o vento sem penalização.

Tendo em conta as condições iniciais anteriormente referidas observa-se que num trajeto contra o vento o veleiro para conseguir chegar do ponto A ao B terá que efetuar mudanças de bordo. O tempo de trajeto diminui consoante a velocidade do vento, ou seja, um percurso a 10 nós é efetuado num tempo maior enquanto que um percurso realizado a 20 nós levaria menos horas, comparativamente.

De referir, que apesar de inicialmente os pontos de guinada escolhidos serem três ou cinco, o programa automaticamente reconhece que para determinados

trajetos basta apenas dois ou até mesmo um. Como observado na Figura 4.2c, do primeiro ponto de guinada para o segundo observa-se ainda um terceiro ponto colinear com a reta traçada. O que demonstra que está a ir de encontro à solução ótima, uma vez que quanto mais pontos de guinada mais tempo de trajeto.

COM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

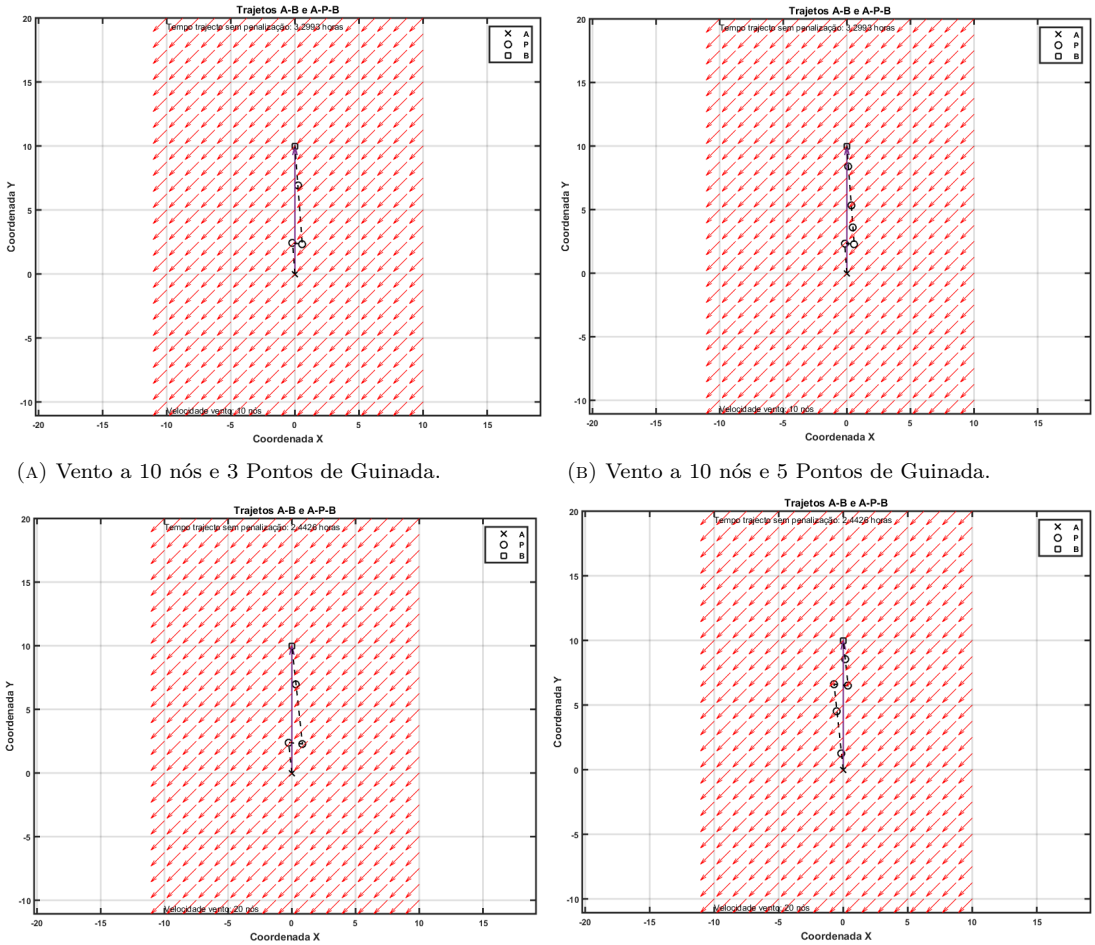
(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.3: Trajetos do Veleiro Alba realizados contra o vento com penalização.

Ao observar-se a Figura 4.2a e 4.3a vê-se claramente o anteriormente descrito, ou seja, ao aplicar a função penalizadora esta faz com que o trajeto passe a ter varias pernadas, neste caso passa de quatro pernadas para apenas duas. O que diminui substancialmente o tempo que o veleiro demora a efetuar o trajeto, pois cada mudança de rumo tem um tempo associado. Quantas mais mudanças de rumo, mais tempo terá o trajeto.

4.2.2 Simulações com vento de 045

SEM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

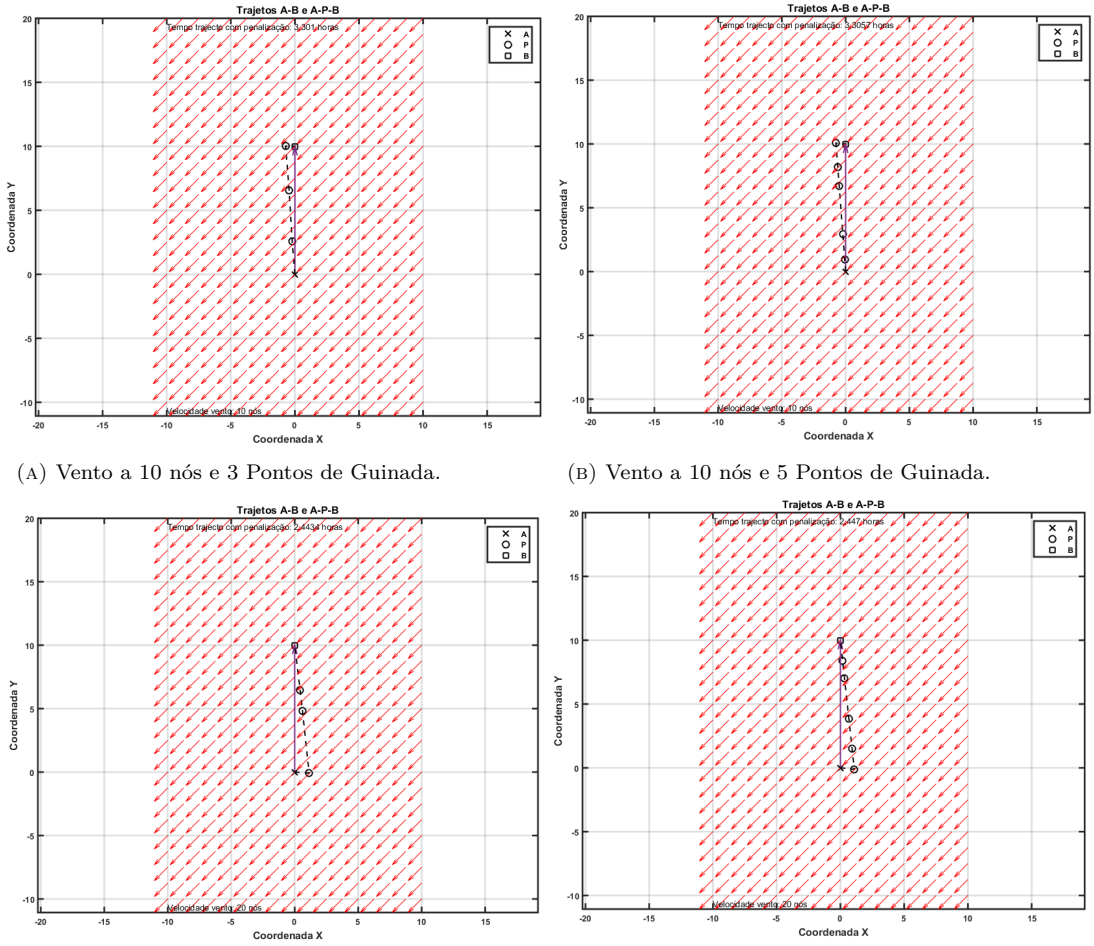
(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.4: Trajetos do Veleiro Alba realizados à bolina sem penalização.

COM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

(D) Vento a 20 nós e 5 Pontos de Guinada

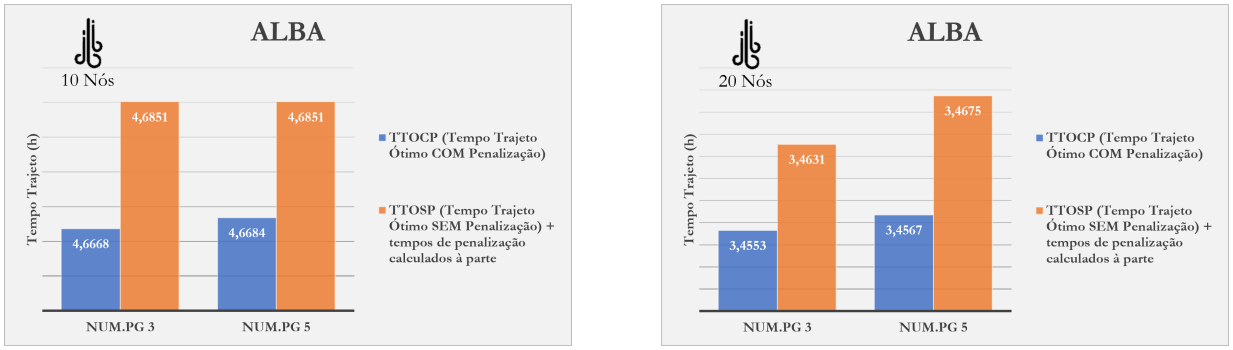
FIGURA 4.5: Trajetos do Veleiro Alba realizados à bolina com penalização.

Apesar de à bolina o veleiro realizar um trajeto mais fechado relativamente ao rumo, consegue-se observar que o programa responde como verificado anteriormente, ou seja, para trajetos que, sem penalização tinham dois pontos de guinada, como o trajeto da Figura 4.4c, quando o programa aplicou o fator da penalização temporal o trajeto passou a ter apenas um ponto de guinada (Figura 4.5c)

Como se pode observar o trajeto ótimo calculado considerando a ação da função penalizadora apresenta pernadas colineares equivalentes a um trajeto com apenas um ponto de guinada. O que indica que o programa está a escrutinar todos os caminhos possíveis e a escolher o de tempo mínimo como ótimo, não acrescentando guinadas, quando não necessárias, quando o utilizador considerou mais pontos de guinada.

Gráficos COM penalização Vs SEM penalização

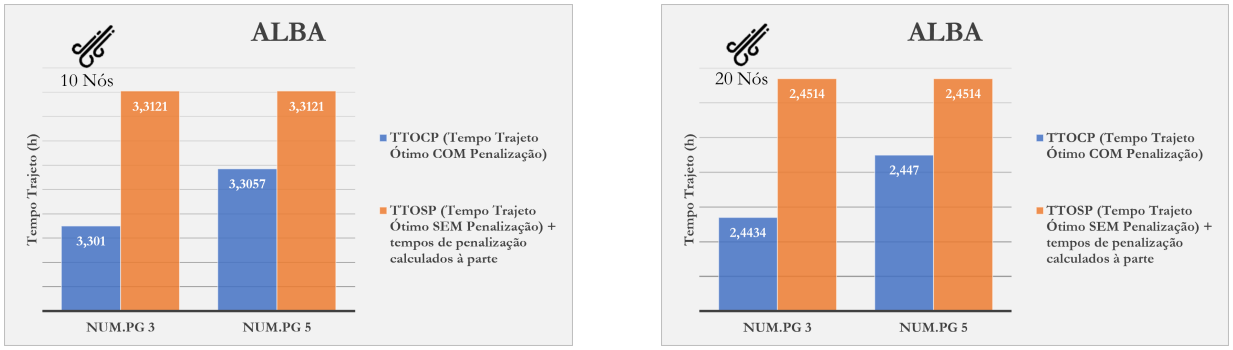
Vento Norte



(A) (B)

FIGURA 4.6: Tempo de trajeto em horas realizados pelo veleiro Alba contra o vento a **10** nós no 1º gráfico e a **20** nós no 2º gráfico. Do lado esquerdo do gráfico estão representados os trajetos com três pontos de guinada e à direita cinco pontos de guinada. A cor azul encontra-se o Tempo de Trajeto Ótimo **com** penalização e a laranja encontra-se o Tempo de Trajeto Ótimo **sem** penalização mais os tempos de penalização calculados à parte.

Vento de 045



(A) (B)

FIGURA 4.7: Tempo de trajeto em horas realizados pelo veleiro Alba à bolina a **10** nós no 1º gráfico e a **20** nós no 2º gráfico. Do lado esquerdo do gráfico estão representados os trajetos com três pontos de guinada e à direita cinco pontos de guinada. A cor azul encontra-se o Tempo de Trajeto Ótimo **com** penalização e a laranja encontra-se o Tempo de Trajeto Ótimo **sem** penalização mais os tempos de penalização calculados à parte.

Através destes gráficos conseguimos comparar de forma expedita os tempos de trajetos que cada simulação está a gerar. É de realçar que é devido à soma entre os tempos de penalização calculados à parte e do Tempo de Trajeto Ótimo SEM

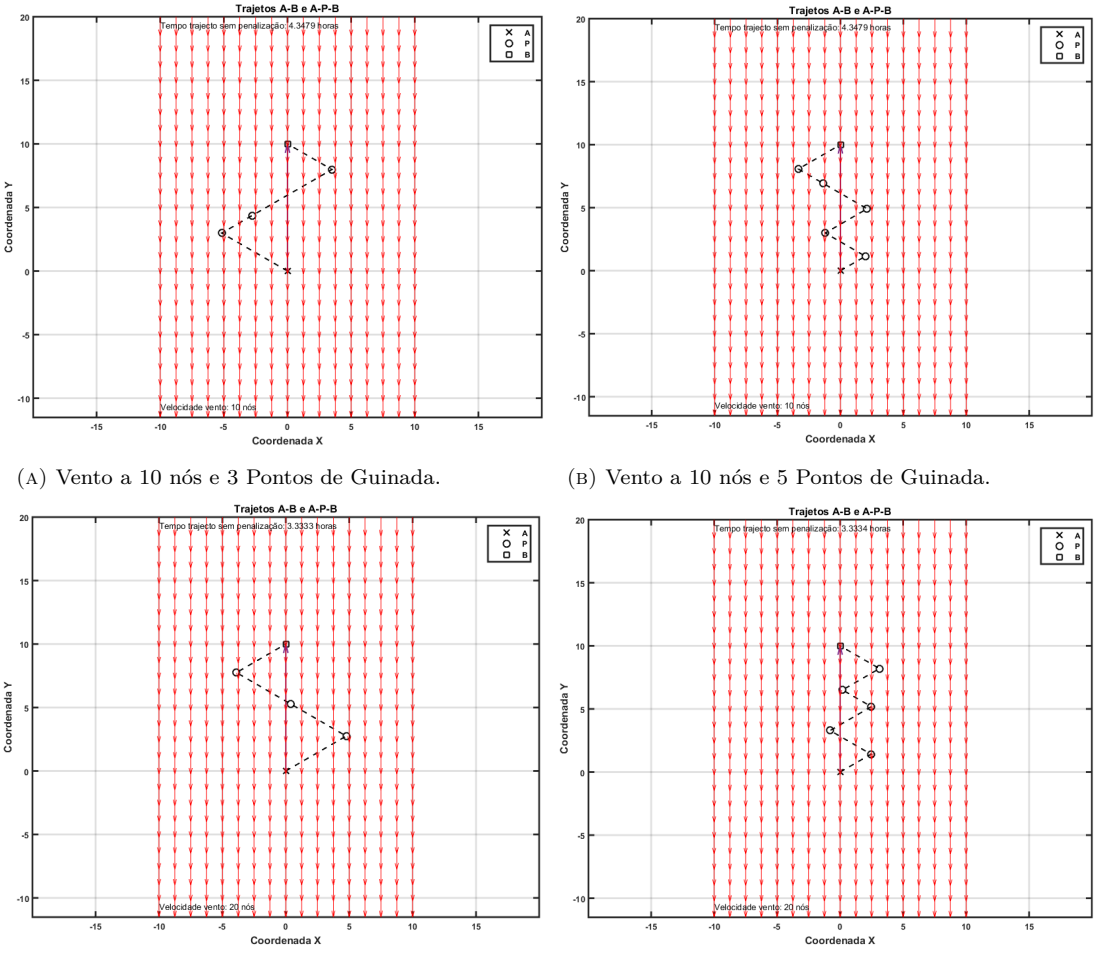
Penalização que este seria de facto maior que o Tempo de Trajeto Ótimo COM Penalização.

Podemos concluir que a adição da penalização temporal associada às mudanças de rumo gera trajetos ótimos mais realistas.

4.3 Trajetos Veleiro NE Sagres

4.3.1 Simulações com vento Norte

SEM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

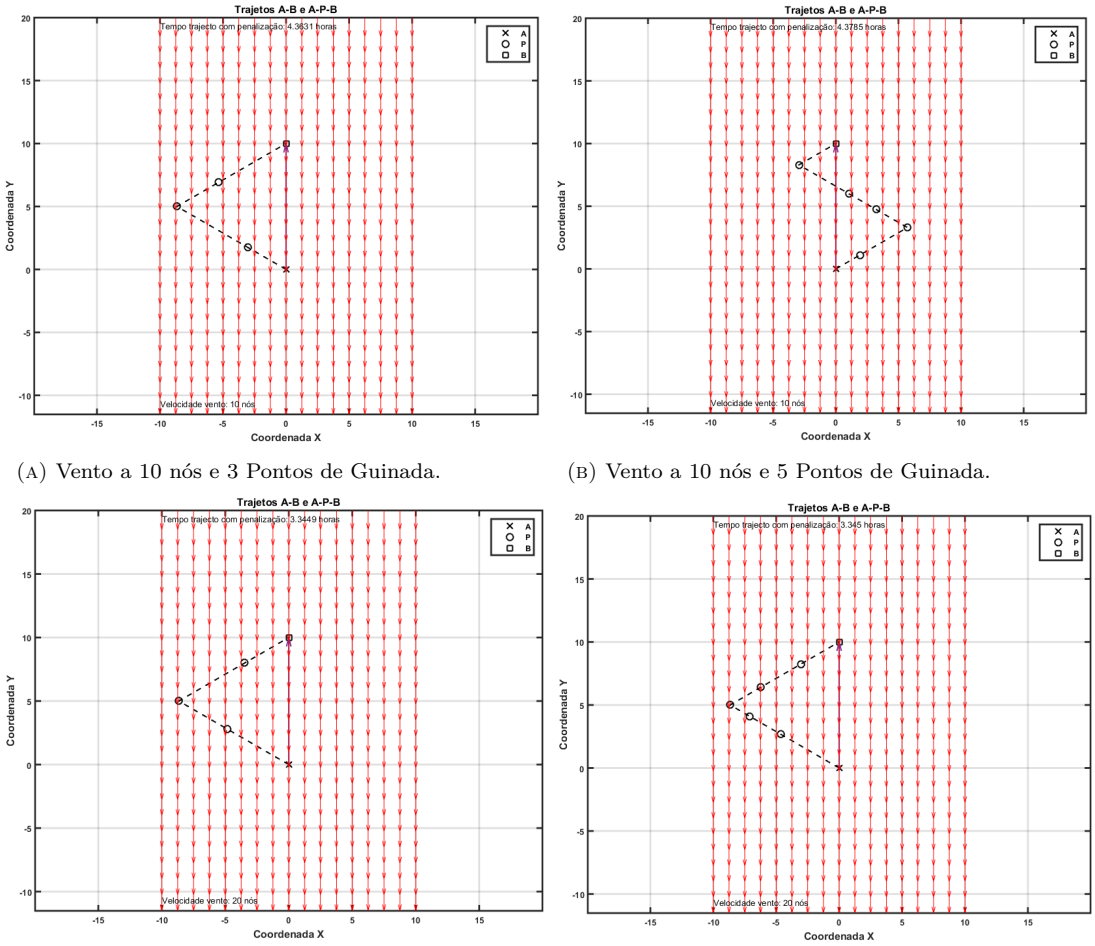
(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.8: Trajetos do Veleiro NE Sagres realizados contra o vento sem penalização.

COM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

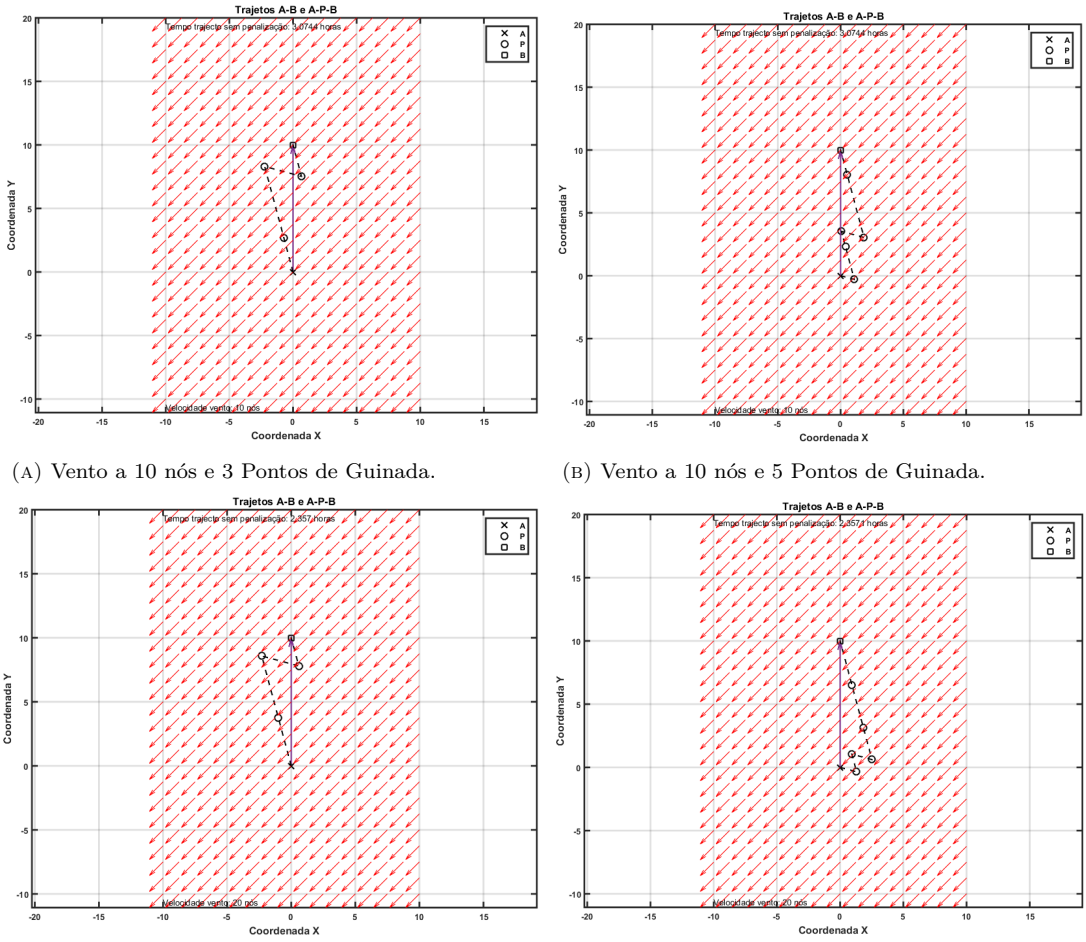
(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.9: Trajetos do Veleiro NE Sagres realizados contra o vento com penalização.

Também aqui podemos observar que os trajetos gerados com a função de penalização temporal (com uma exeção) apresentam pernadas colineares equivalentes a trajetos com um único ponto de guinada, necessariamente mais realistas.

4.3.2 Simulações com vento de 045

SEM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

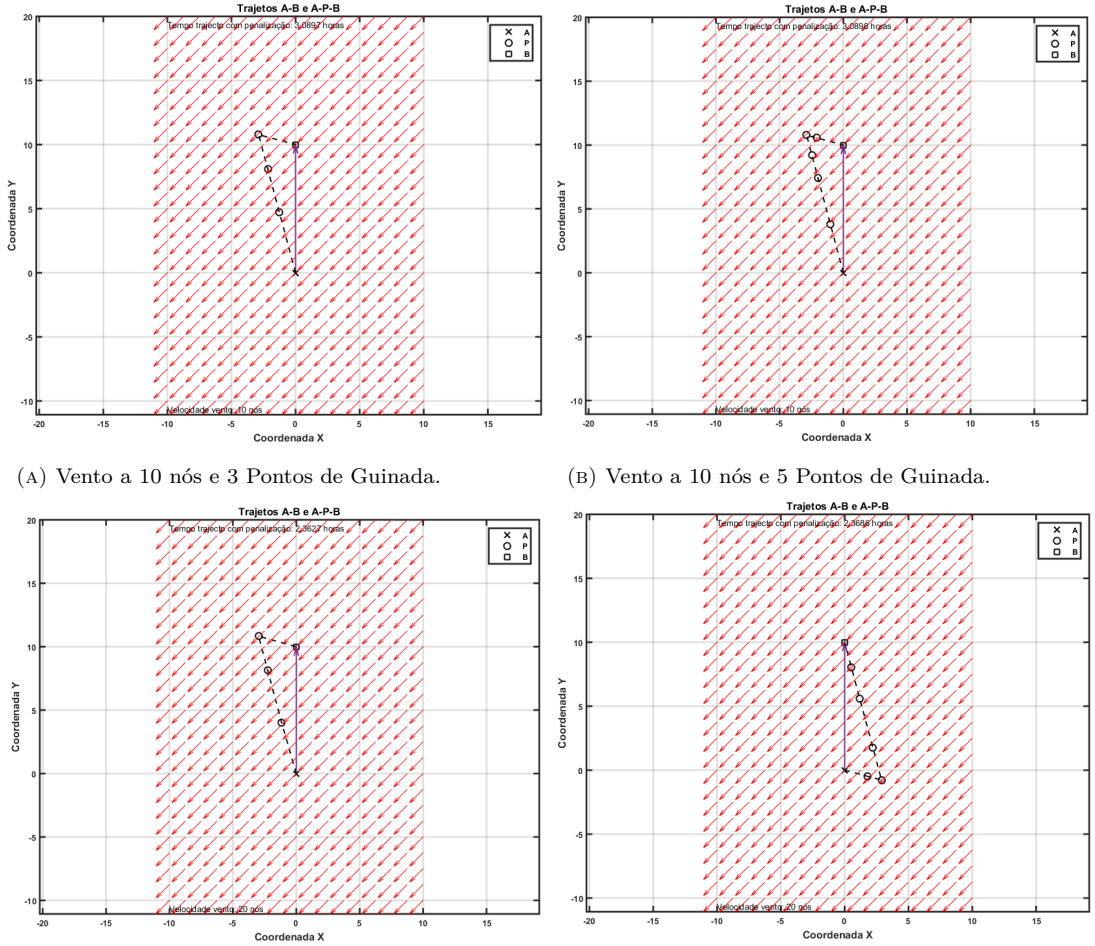
(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.10: Trajetos do Veleiro NE Sagres realizados à bolina sem penalização.

COM penalização



(A) Vento a 10 nós e 3 Pontos de Guinada.

(B) Vento a 10 nós e 5 Pontos de Guinada.

(C) Vento a 20 nós e 3 Pontos de Guinada

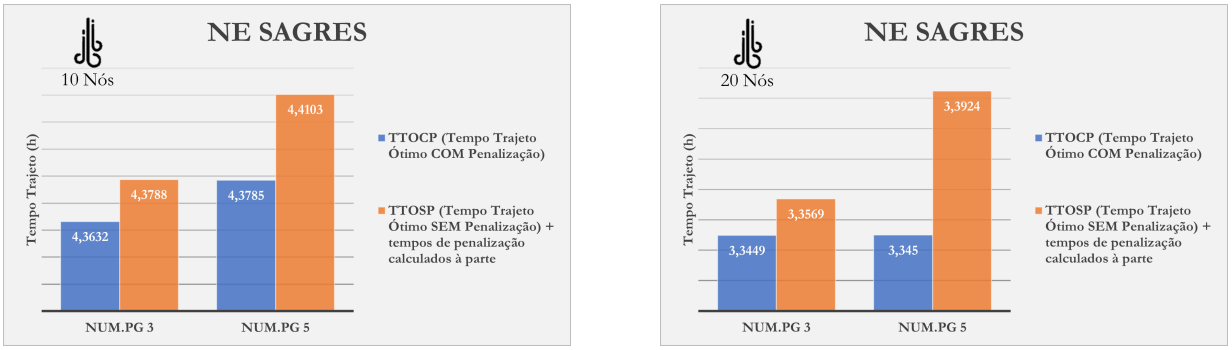
(D) Vento a 20 nós e 5 Pontos de Guinada

FIGURA 4.11: Trajetos do Veleiro NE Sagres realizados à bolina com penalização.

Nestas simulações podemos observar igualmente que os trajetos gerados com a função de penalização temporal (com uma exceção) apresentam pernadas colineares equivalentes a trajetos com um único ponto de guinada, necesssariamente mais realistas.

Gráficos COM penalização Vs SEM penalização

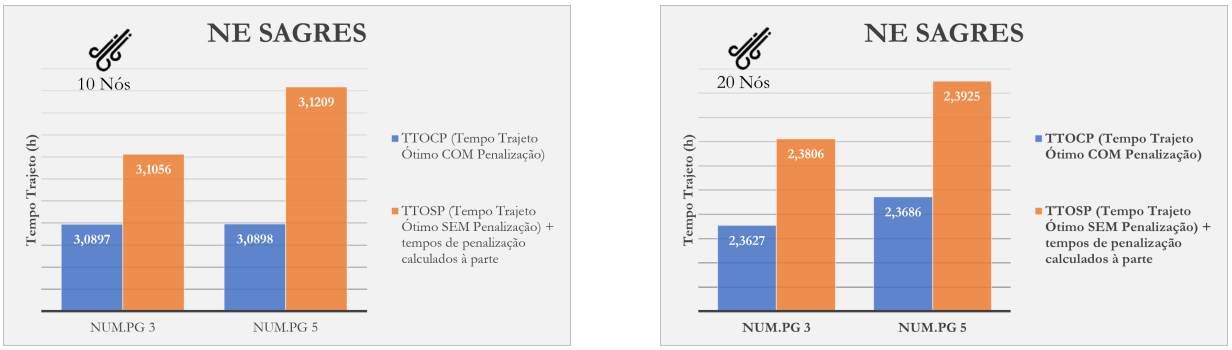
Vento Norte



(A) (B)

FIGURA 4.12: Tempo de trajeto em horas realizados pelo veleiro NE Sagres contra o vento a **10** nós no 1º gráfico e a **20** nós no 2º gráfico. Do lado esquerdo do gráfico estão representados os trajetos com três pontos de guinada e à direita cinco pontos de guinada. A cor azul encontra-se o Tempo de Trajeto Ótimo **com** penalização e a laranja encontra-se o Tempo de Trajeto Ótimo **sem** penalização mais os tempos de penalização calculados à parte.

Vento de 045



(A) (B)

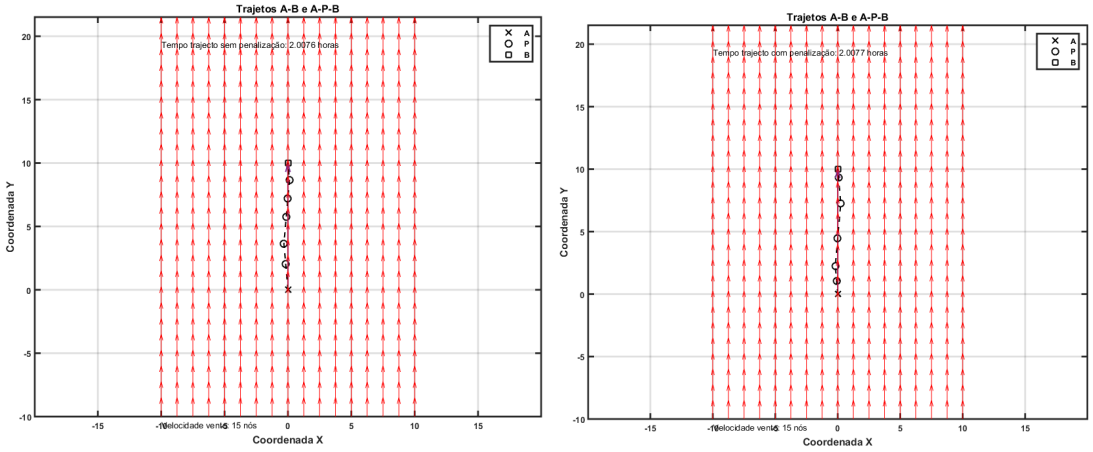
FIGURA 4.13: Tempo de trajeto em horas realizados pelo veleiro NE Sagres à bolina a **10** nós no 1º gráfico e a **20** nós no 2º gráfico. Do lado esquerdo do gráfico estão representados os trajetos com três pontos de guinada e à direita cinco pontos de guinada. A cor azul encontra-se o Tempo de Trajeto Ótimo **com** penalização e a laranja encontra-se o Tempo de Trajeto Ótimo **sem** penalização mais os tempos de penalização calculados à parte.

4.4 Trajetos com Popas e Largos

Neste sub capitulo, pretende-se simular trajetos com o veleiro a navegar com vento de popa e ao largo, respetivamente com vento de 000 e 315. Para tal, foi escolhido o Veleiro Alba.

Simulações com vento de 000

Como se pode observar pela Figura 4.14a e 4.14b tanto o trajeto com penalização e sem penalização apresentam valores semelhantes, respetivamente, 2,0076 e 2,0077 horas. Este facto resulta da não existência de mudanças de rumo penalizadas temporalmente. Com efeito, a função penalizadora temporal só aplica penalizações quando a mudança de rumo envolve mudança de bordo à bolina.



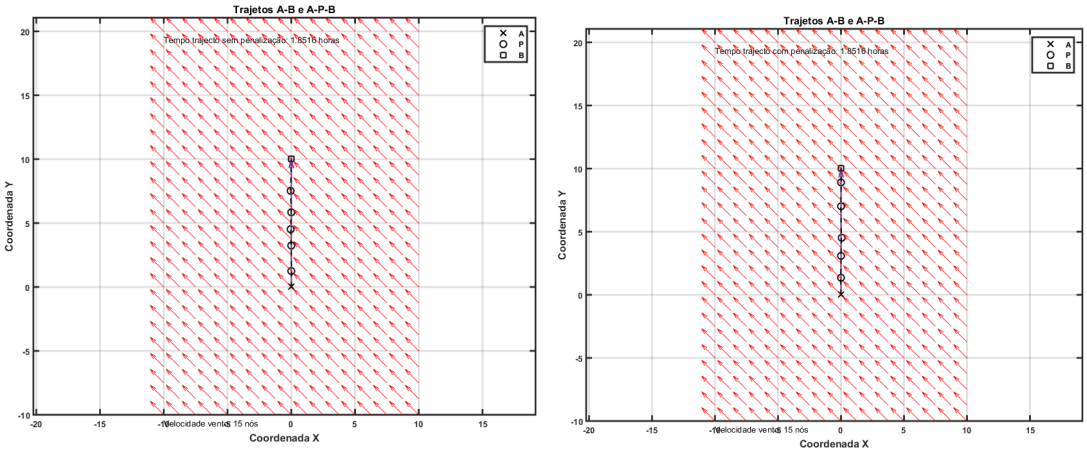
(A) **Sem penalização**,vento a 15 nós e 5 Pontos de Guinada.

(B) **Com penalização**,vento a 15 nós e 5 Pontos de Guinada.

FIGURA 4.14: Simulação do Veleiro Alba a realizar o trajeto com vento pela popa.

Simulações com vento de 315

O mesmo se aplica a quando a embarcação realiza um largo. Como ilustrado pela Figura 4.15a e 4.15b o trajeto ótimo corresponde ao trajeto direto de A para B, uma vez que seguindo o trajeto direto, o ângulo de entrada do vento no veleiro está muito próximo do ângulo para o qual a velocidade deste é máxima. À semelhança do anteriormente descrito, os tempos são semelhantes sendo mesmo iguais, 1.8516 horas. É importante realçar que, como nas simulações com vento pela popa, mais uma vez a função penalizadora temporal nestes trajetos não atua sendo ao largo que o veleiro executa o trajeto ótimo, segundo estas condições.



(A) **Sem penalização**,vento a 15 nós e 5 Pontos de Guinada.

(B) **Com penalização**,vento a 15 nós e 5 Pontos de Guinada.

FIGURA 4.15: Simulação do Veleiro Alba a realizar o trajeto ao largo.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões

Este trabalho teve como principais objetivos dotar os algoritmos desenvolvidos anteriormente (Fedorchuk, 2020) com capacidade para:

♢ Obter trajetos com mais de duas pernadas;

Relativamente à adição da capacidade para otimizar trajetos com mais de duas pernadas, o algoritmo desenvolvido permite que o utilizador defina como requisito um número arbitrário de pernadas. No entanto, esta funcionalidade em situações de vento uniforme (e estacionário) e na ausência de constrangimentos e restrições espaciais aos trajetos, não oferece vantagens, uma vez que os trajetos ótimos nestas circunstâncias podem ser obtidos com um único ponto de guinada nomeadamente se se considerar nos cálculos as penalizações associadas à execução de guinadas.

As vantagens de utilização de trajetos com numerosos pontos de guinada colocar-se-á na presença de campos de velocidade de vento não estacionários e ou considerando a existência de restrições espaciais aos trajetos permitidos (existência de obstáculos ou regiões não navegáveis). No entanto, o cálculo de trajetos complexos com numerosos pontos de guinada irá ter como preço a pagar o aumento exponencial do espaço de pesquisa utilizado no processo de otimização.

♢ Introduzir um fator de penalização temporal associado à realização de guinadas permitindo gerar trajetos ótimos mais realistas;

Esta funcionalidade permite obter trajetos mais próximos da realidade com a particularidade de ser ajustável a cada tipo de veleiro, isto é, como os parâmetros da função penalizadora são caraterísticas do veleiro faz com que para além de permitir obter o tempo de trajeto mínimo é única para cada veleiro.

Um fator importante para validação desta mesma função será o facto de testar a função no terreno, através de provas de conceito, por exemplo, para vários tipos de veleiro. O que não foi abordado neste trabalho.

♢ Traduzir os algoritmos desenvolvidos em *MATLAB*® para *Python*™ de forma a facilitar a sua utilização num futuro próximo em veleiros autónomos. Cujo hardware seja baseado em plataformas Raspberry Pi.

5.2 Trabalho Futuro

A breve trecho pretende-se:

- Implementar o algoritmo desenvolvido e já traduzido para *Python*™ numa plataforma Raspberry Pi adicionando a capacidade de recolha, leitura e processamento de dados em tempo real e fornecimento ao sistema de controlo de ajustes periódicos ao rumo ótimo calculado para chegada ao destino;
- Desenvolver os algoritmos desenvolvidos dotando-os da possibilidade de calcular tempos de trajeto mínimos em situação de campos de velocidade do vento não uniformes (mas estacionários);
- Desenvolver os algoritmos desenvolvidos dotando-os da possibilidade de calcular tempos de trajeto mínimos em situação de campos de velocidade do vento não uniformes nem estacionários

Bibliografia

A. Philpott, A. J. M. (2001). Optimising Yacht Routes under Uncertainty. *The 15th CHESAPEAKE SAILLING YACHT SYMPOSIUM*.

Afonso, P. M. R. (2016). *Desenvolvimento de um sistema para aferir a performance aerodinâmica de uma embarcação à vela baseado numa placa de prototipagem eletrónica Arduino* (tese de mestrado). Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

Almeida, W. R. M. (2004). *Anemómetro Baseaso no Método de Tempo de Transito:Estudo comparativo de arquiteturas, avaliação de incertezas e implementação* (tese de mestrado). Universidade Federal do Maranhão.

Anjo, A. J. B. (1994). *O Método de Simulação da Têmpera - Uma Abordagem Matemática*. Universidade de Aveiro.

Barreira, N. M. C. (2016). *Sistema Inteligente para Otimização de Rotas* (tese de mestrado). Universidade Nova de Lisboa.

Blum, C. & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*.

Casaca, J. & Henriques, M. J. (2004). O Principio de Fermat e a Refração Vertical. *Métodos Computacionais em Engenharia*.

Challenge, T. M. (s.d.). *About the Microtransat*. https://www.microtransat.org/

Dalang, R. C. (2010). Stochastic Optimization of Sailing Trajectories in an Upwind Regatta.

da Rocha, A. P. (2020). *Métodos da Geometria Diferencial no Estudo da Acústica Submarina na Aproximação Geométrica* (tese de mestrado). Escola Naval.

da Sila, M. O. (2011). *Campos potenciais modificados aplicados ao controle de múltiplos robôs* (tese de mestrado). Instituto de Ciências Matemáticas e de Computação - IMCM-USP.

da Silva, L. M. (2017). *Cadeias de Markov e Aplicações* (tese de mestrado). Universidade Federal Fluminense, Brasil.

Debski, R. (2016). An Adaptive Multi-Spline Refinement Algorithm in Simulation Based Sailboat Trajectory Optimization Using Onboard Multi-Core Computer Systems. *International Journal of Applied Mathematics and Computer Science*.

de Campos Bueno, V. & da Cunha Pires Soeiro, F. J. (1997). O Recozimento Simulado como Ferramenta de Otimização Global. *Revista Militar de Ciència e Tecnologia*.

de Carvalho, B. M. P. S. (2008). *Algoritmo de Dijkstra* [Departamento de Engenharia Informática Universidade de Coimbra, Portugal].

de Castro Honorato Silva, B. (2013). *Otimização de Rotas Utilizando Abordagens Heurísticas em um Ambiente Georreferenciado* (tese de mestrado). Universidade Estadual Do Ceará.

Dekkers, A. & Aarts, E. (1991). Global optimization and simulated annealing. *Mathematical Programming*.

El-Rabbany, A. (2022). *Introduction to GPS - The Global Positioning System*. Artech House.

Erckens, H., Busser, G.-A., Pradalier, D. C. & Siegwart, P. D. R. Y. (2010). Navigation Strategy and Trajectory Following Controller for an Autonomous Sailing Vessel. *IEEE Robotics & Automation Magazine*.

e Sá, N. B. (2005). Física Elementar da Navegação à Vela. *Gazeta de Física*.

Fedorchuk, M. (2020). *Otimização de trajetórias de veleiros* (tese de mestrado). Escola Naval.

Fernandes, B. A. (2013). *Otimização de Rotas de Distribuição Marítima de Curta Distância* (tese de mestrado). Universidade de Aveiro.

Fernandez, P. J. (2014). *Introdução aos Processos Estocásticos*. Colóquio Brasileiro de Matemática. Paços de Caldas, Minas Gerais, Brasil.

Ferreira, D. & Martins, J. A. (2002). *Navegador de Recreio* (Vol. 4). Dinalivro.

FOUNDATION, R. P. (s.d.). https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

Francisco, P. S. (s.d.). https://www.portalsaofrancisco.com.br/curiosidades/barco-a-vela

Gonzalez, O. R., Kuper, C., Jung, K., Naval, P. C. & Mendoza, E. (2007). Parameter estimation using Simulated Annealing for S-system models of biochemical networks. *BIOINFORMATICS*.

iNavX. (2019). https://inavx.com/

Inc., J. A. (2021). *iSailGPS*. https://isailgps.com/

Junior, I. R. S. & Cechin, A. L. (2006). Comparação entre Métodos Exatos e Heurísticos para tratar o Problema de Roteamento de Veículos em um Ambiente Fabril. *ENEGEP - Fortaleza, CE, Brasil*.

Júnior, D. P. F. & Júnior, V. V. (s.d.). *Concecitos e Simulações de Cadeias de Markov*. http://www.sbpcnet.org.br/livro/63ra/conpeex/pivic/trabalhos/DIVALDO_.PDF

Kimball, J. (2010). *Physics of Sailing*. CRC Press Taylor & Francis Group.

Koenig, R. (2018). *Marine Navigator*. https://www.visitmyharbour.com/what-is-marine-navigator.asp

Kulkarni, A. J., Abraham, A. & Krishnasamy, G. (2017). Introduction to Optimization. *Springer International Publishing Switzerland*.

Leitão, A. (2001). *Cálculo Variacional e Controle Ótimo*. Departamento de Matemática Universidade Federal de Santa Catarina. Florianópolis - SC, Brasil.

Ltd, P. M. (2014). *SeaNav*. https://pocketmariner.com/mobile-apps/seanavapp/

Ltd, W. V. (2010). *Transas iSailor*. http://www.isailor.us/

Marcin Życzkowski, R. S. (2017). MULTI-OBJECTIVE WEATHER ROUTING OF SAILING VESSELS. *POLISH MARITIME RESEARCH*, *24*, 10–17.

Meltemus. (2017). *qtVlm*. https://www.meltemus.com/index.php/en/

N, S. E., R, O. & H, C. K. M. C. K. A. N. (2016). Potential Field Methods and Their Inherent Approaches for Path Planning. *ARPN Journal of Engineering and Applied Sciences*, *11*.

Pessoa, P. F. P. (2011). *Opções de Conversão com Movimento de Reversão à Média com Saltos de Poisson: o Caso do Setor Sucroalcooleiro* (tese de mestrado). Pontifícia Universidade Catolica do Rio de Janeiro.

Pêtrès, C., Romero-Ramirez, M.-A. & Plumet, F. (2011). Reactive path planning for autonomous sailboat. *Int. Conf. on Advanced Robotics*, 112–117.

Rabaud, M. (2016). Optimal routing in sailing. *Proceeding of the conference Sports Physics,*

Ribeiro, M. P. F. (2013). *Algoritmos de optimização para redes ópticas de transporte* (tese de mestrado). Universidade de Aveiro.

Santos, J. F. B. (2019). *eVentos – Desenvolvimentos no veleiro autónomo Barlavento* (tese de mestrado). Escola Naval.

S.r.l., N. (2021). *Navionics*. https://www.navionics.com/fin/

Stelzer, R. (2012). *Autonomous Sailboat Navigation Novel Algorithms and Experimental Demonstration* (tese de mestrado). Montfort University, Leicester.

Venter, G. (2010). Review of Optimization Techniques. *Encyclopedia of Aerospace Engineering*.

Walther, L., Rizvanolli, A., Wendebourg, M. & Jahn, C. (2016). Modeling and Optimization Algorithms in Ship Weather Routing. *International Journal of e-Navigation and Maritime Economy*.

Weise, T. (2009). *Global Optimization Algorithms – Theory and Application –*. online in http://www.it-weise.de/.

Yang, X.-S. (2013). Optimization and Metaheuristic Algorithms in Engineering. *Metaheursitics in Water, Geotechnical and Transport Engineering*.

Zanchin, B. C. (2018). *Análise do Algoritmo A*(A estrela) no Planeamento de Rotas de Veículos Autónomos* (tese de mestrado). Universidade Tecnológica Federal do Paraná.

Apêndice A - Parametrização das constantes k_1 e k_2 para o Veleiro ALBA (30m de comprimento)

$\Rightarrow$ Quando V igual à velocidade máxima V_{max} do veleiro e α= 90º:

$$t = \frac{L}{1852 \times V_{max}} \tag{A.1}$$

,em que L representa o comprimento do veleiro em metros.

$\Rightarrow$ Quando $V = 0$ e α= 90º (penalização 20 vezes superior)

$$t = \frac{20 \times L}{1852 \times V_{max}} \tag{A.2}$$

Penalização em horas para $\alpha = 90$, L= 30m e $V = V_{max}$ =11 nós:

$$\frac{30}{1852 \times 11} = 1.4726 \times 10^{-3}(5s) \tag{A.3}$$

Penalização em horas para $\alpha = 90$, L= 30m e V =0 nós:

$$\frac{30 \times 20}{1852 \times 11} = 2.945 \times 10^{-2}(1m45s) \tag{A.4}$$

Determinação de k_1 e k_2 para um veleiro L=30m e V_{max}=11 (α=90º):

$$\begin{cases} k_1.e^{-11k_2} = \frac{30}{20372} \\ \\ k_1.e^{-k_2} = \frac{600}{20372} \end{cases} \tag{A.5}$$

Solução:

$$\Big\{[k_1 = 3.9739 \times 10^{-2}, k_2 = 0.29957]\Big\} \tag{A.6}$$

Modelo de penalização

$$t = 3.9739 \times 10^{-2}.e^{-0.29957V\frac{\alpha}{90}} \mathrm{e} m horas \tag{A.7}$$

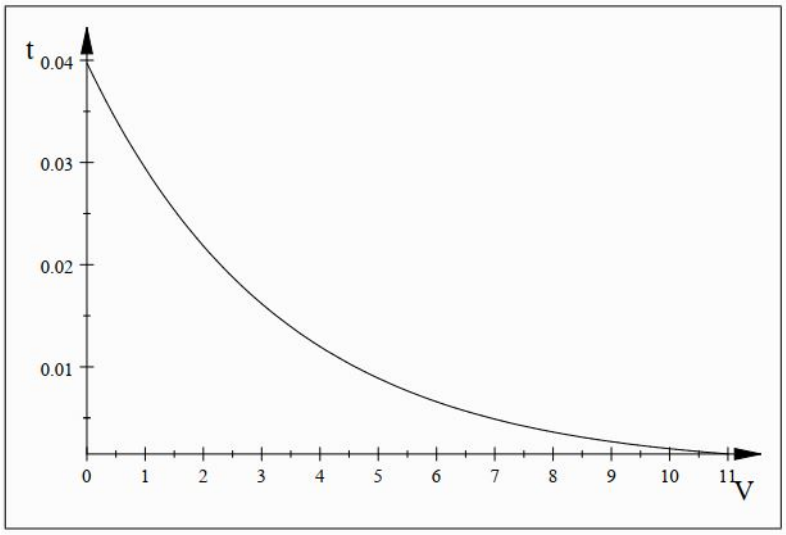


FIGURA A.1: Gráfico da penalização (em segundos de penalização associados a $\alpha = 90$ž)

Apêndice B - Parametrização das constantes k_1 e k_2 para o Veleiro NE SAGRES (89m de comprimento)

$\Rightarrow$ Quando V igual à velocidade máxima V_{max} do veleiro e α= 90º:

$$t = \frac{L}{1852 \times V_{max}} \tag{B.1}$$

,em que L representa o comprimento do veleiro em metros.

$\Rightarrow$ Quando $V = 0$ e α= 90º (penalização 20 vezes superior)

$$t = \frac{20 \times L}{1852 \times V_{max}} \tag{B.2}$$

Penalização em horas para $\alpha = 90$, L= 89m e $V = V_{max}$ =16 nós:

$$\frac{89}{1852 \times 16} = 3.0035 \times 10^{-3}(11s) \tag{B.3}$$

Penalização em horas para $\alpha = 90$, L= 89m e V =0 nós:

$$\frac{89 \times 20}{1852 \times 16} = 6.00701 \times 10^{-2}(3min36s) \tag{B.4}$$

Determinação de k_1 e k_2 para um veleiro L=89m e V_{max}=16 (α=90º):

$$\begin{cases} k_1.e^{-16k_2} = \frac{89}{29632} \\ \\ k_1.e^{-k_2} = \frac{445}{7408} \end{cases} \tag{B.5}$$

Solução:

$$\Big\{[k_1 = 7.3349 \times 10^{-2}, k_2 = 0.19972]\Big\} \tag{B.6}$$

Modelo de penalização

$$t = 7.3349 \times 10^{-2}.e^{-0.19972V\frac{\alpha}{90}} \mathrm{e} mhoras \tag{B.7}$$

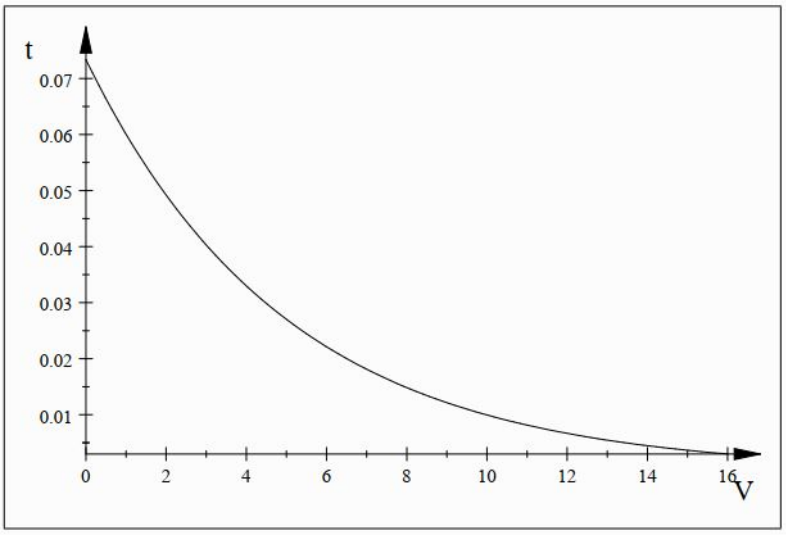


FIGURA B.1: Gráfico da penalização (em segundos de penalização associados a $\alpha = 90$ž)

Apêndice C - Tradução do programa para *Python*™

```python
import numpy as np

isave = 0  # Salvar os dados para posterior processamento
pen = 0  # Se pen = 1 aplica o modelo de penalizacao

x1: int = 0
y1: int = -30  # PONTO A
x2: int = 0
y2: int = 30  # PONTO B

# velocidade do vento
vento = 10  # n s
#  Imposicaoo do valor do Azimute Para Onde Sopra o Vento ("aposv") que
#  permitir  definir o  ngulo  theta correspondente ao rumo de cada pernada, (rAP e rPB) para entrada no diagrama polar das velocidades do
#  veleiro.

aposv = 135  # Azimute para onde sopra o vento

A = np.array([x1, y1])  # PONTO A
B = np.array([x2, y2])  # PONTO B

#  A vari vel "passos" definir  uma dist ncia de refer ncia "resolucao"
#  para gerar a resolucao espacial  m dia do espaco de pesquisa:
#  dAB/passos;
#  passos=5;  n mero de subdivis es de dAB (distancia de referencia)
passos = 5

#  Numero de pontos de guinada:
#  1 ponto de guinada corresponde a 2 pernadas;
#  2 pontos de guinada corresponde a 3 pernadas, etc.
numPG = 3

#               EPbase
#  Criacao das matrizes com as coordenadas X (MXep)
#  e coordenadas Y (MYep) dos pontos de guinada:
#
#  MXep: Em cada linha da coluna j encontra-se a coordenada  X
#  do ponto de guinada j.
```

```
#
#  MYep: Em cada linha da coluna j encontra-se a coordenada Y
#  do ponto de guinada j.
#  dAB-distancia de A a B
#  rAB-rumo de A a B (coordenadas polares [-180 a +180]

from EPbase import EPbase

MXep, MYep, dAB, rAB, resolucao, dimEP = EPbase(A, B, passos, numPG
    )
#  MXep
#  MYep
from trajectoPG import trajectoPG

#                  trajectoPG
#  A fun   o trajectoPG gera os trajectos (dist ncia e rumo)
#  dA_P1, ..., dPi_Pi+1,...,dPN_B
#
#   OBS: Estas matrizes incluem ja as informacoes (distancias e
    rumos)
#        com A e B
#  MdPG-Matriz das distancias (entre A, pontos de guinada e B)
#  MrPG-Matriz dos rumos (entre A, pontos de guinada e B)
#  Dimens es:
#   numero de linhas: dimEP (dimensao espa o de pesquisa)
#   numero de colunas:  numPG+1 (numero de pontos de guinada)

#  Obs: Os rumos  s  o calculados em
#  em coordenadas polares [-180 a +180]

#  MdPG
#  MrPG

MdPG, MrPG = trajectoPG(MXep, MYep, A, B)

#  Calculo dos correspondentes angulos theta de entrada no diagrama
     polar
#  para computacao das velocidades verificadas nas pernadas entre
#  os pontos de guinada.
from theta0P import theta0P

thetaAB, adosv = theta0P(rAB, aposv)

# thetaAB
# adosv
```

```
81 from thetaPG import thetaPG
82
83 MthetaPG = thetaPG(MrPG, aposv)
84
85 # MthetaPG
86
87 #                     gera_diag_polar
88 #
89 #  Gera_diag_polar com base nos dados deste tipo de diagrama
       fornecido pelo
90 #  veleiro. Este diagrama polar esta associado a velocidade do
       vento "vento"
91 #  e ser  utilizado em todos os c lculos feitos daqui para a
       frente.
92 #  diag_polar e uma fun  o polinomial (pp) definida por ramos de
       0 a 180 graus.
93 #  VARI VEIS:
94 #  --> x e y sao outputs meramente informativos: x-angulos de
       entrada do vento verdadeiro fornecidos pelo diagrama introduzido
       , y-velocidade do veleiro atingida (tendo em conta a velocidade
       do vento escolhida: vento)
95 #  --> diag_polar-(pp) definida por ramos de 0 a 180 graus (para
       theta fornece a
96 #  velocidade por interpolacao (spline ou linear)
97
98 from gera_diag_polar_sagres import gera_diag_polar_sagres
99
100 diag_polar, x, y = gera_diag_polar_sagres(vento)
101
102 # Calculo das velocidades em cada pernada de A a P a B
103
104 vel_AB = diag_polar(thetaAB)  # Trajecto directo
105
106 vel_PG = diag_polar(MthetaPG)  # Restantes trajectos com pontos de
       guinada
107
108 # Calculo dos tempos de percurso em cada pernada
109 temp_AB = dAB / vel_AB  # Trajecto directo
110
111 temp_PG = MdPG / vel_PG  # Restantes trajectos
112
113 #  Calulo da penalizacao  associada as guinadas
114 #  diferente de zero se pen = 1
115 from penalty import penalty
116
```

```
117 penalty_PG, BEV, BOL, BEVDIFF, BOLSUM, MATPENAL, ANGVAR, VELMED =
        penalty(MrPG, MthetaPG, vel_PG, aposv, pen)
118
119 #
120 #
121 #  Calculo dos tempos totais de percurso (horas) (distancia em
        milhas nauticas e velocidade em nos)
122 #
123 tempo_total_AB = temp_AB.copy()
124 tempo_total_APB = temp_PG.sum(axis=1) + penalty_PG.sum(axis=1)
125
126 # INICIO SIMULATING ANNEALING
127 # Colocar os valores do tempo total por ordem cresceste atraves da
        funcao
128 # "sort"
129 # MXep, MYep, tempo_total_APB
130
131 tempo_total_APB_ord = np.sort(tempo_total_APB)
132 index = np.argsort(tempo_total_APB)
133
134 # Reordenar as linhas das matrizes MXep e MYep de acordo com a nova
        ordenacao
135 #  do vector tempo_total_APB
136
137 MXep_ord = MXep[index]  # ;  MXep_ord-Matriz MXep ordenada por
        tempos de trajecto crescentes
138 MYep_ord = MYep[index]  # ;  MYep_ord-Matriz MYep ordenada por
        tempo de trajectos crescentes
139
140 #
141 # N mero de eleitos: numELEITOS
142 # so 50% dos melhores trajectos preservados para a epoca seguinte
143
144 numELEITOS = np.int(dimEP / 2)  # dimEP    par!!
145
146 tempo_total_APB_1 = tempo_total_APB_ord[:numELEITOS]
147 tempo_total_APB_1 = np.append(tempo_total_APB_1, tempo_total_APB_1)
148 MXep_1 = MXep_ord[:numELEITOS, :]
149
150 MXep_1 = np.concatenate([MXep_1, MXep_1])
151 MYep_1 = MYep_ord[:numELEITOS, :]
152 MYep_1 = np.concatenate([MYep_1, MYep_1])
153
154 #
155 # Guardar o melhor resultado
```

```
156 melhor_tempo = tempo_total_APB_1[0]
157 melhor_X = MXep_1[0]
158 melhor_Y = MYep_1[0]
159
160 # Amplitude perturbadora base: dAB
161
162 amp_per = dAB.copy()  # sempre copy para que nao fiquem ligadas
163
164 #
165 tempo_min = []  # lista com os tempos minimos obtidos em cada epoca
166 #
167 ciclos = 1000  # numero de ciclos de arrefecimento
168
169 ### Inicio do loop
170
171
172 for i1 in range(ciclos):
173     # Guardar o melhor tempo da geracao anterior e as
    correspondentes
174     # coordenadas dos pontos de guinada
175     total_tempo_tmp = melhor_tempo.copy()
176     melhor_X_tmp = melhor_X.copy()
177     melhor_Y_tmp = melhor_Y.copy()
178     #
179
180     amp_per = amp_per / 1.01
181     #
182     #
183     #
184     # Criacao das matrizes que vao fazer a perturbacao em MXep,
    MYep
185     MXrandom = -amp_per + 2 * amp_per * np.random.rand(dimEP, numPG
    )  #
186     MYrandom = -amp_per + 2 * amp_per * np.random.rand(dimEP, numPG
    )  #
187
188     # Perturbacao:
189
190     MXep_1 = MXep_1 + MXrandom
191     MYep_1 = MYep_1 + MYrandom  # Alguma razao para nao ter +
192
193     MdPG, MrPG = trajectoPG(MXep_1, MYep_1, A, B)
194     MthetaPG = thetaPG(MrPG, aposv)
195     vel_PG = diag_polar(MthetaPG)
196     temp_PG = MdPG / vel_PG
```

```
197
198     #  Calculo da penalizacao temporal se pen = 1
199     #  diferente de zero se pen = 1
200     penalty_PG, BEV, BOL, BEVDIFF, BOLSUM, MATPENAL, ANGVAR, VELMED
        = penalty(MrPG, MthetaPG, vel_PG, aposv, pen)
201     tempo_total_APB = np.sum(temp_PG, axis=1) + np.sum(penalty_PG,
    axis=1)
202     tempo_total_APB_ord = np.sort(tempo_total_APB)
203     index = np.argsort(tempo_total_APB)
204     MXep_ord = MXep[index]
205     MYep_ord = MYep[index]
206
207     #  Constituicao do espaco de pesquisa com os 50% melhores
    trajectos
208     tempo_total_APB_ord = tempo_total_APB_ord[:numELEITOS]
209     tempo_total_APB_ord = np.append(tempo_total_APB_ord,
    tempo_total_APB_ord)
210     MXep_ord = MXep_ord[:numELEITOS, :]
211     MXep_ord = np.concatenate([MXep_ord, MXep_ord])
212     MYep_ord = MYep_ord[:numELEITOS, :]
213     MYep_ord = np.concatenate([MYep_ord, MYep_ord])
214
215     # Substituir o ultimo valor desta geracao pelo primeiro da
    geracao
216     # anterior
217     tempo_total_APB_ord[-1] = total_tempo_tmp
218     MXep_ord[-1] = melhor_X_tmp
219     MYep_ord[-1] = melhor_Y_tmp
220
221     tempo_total_APB_ord = np.sort(tempo_total_APB_ord)
222     index = np.argsort(tempo_total_APB_ord)
223     MXep_1 = MXep_ord[index]  # MXep_ord-Matriz MXep ordenada por
    tempos de trajecto crescentes
224     MYep_1 = MYep_ord[index]  #
225     #
226     #
227     # Guardar o melhor resultado desta geracao para ser usado na
    geracao
228     # seguinte
229     melhor_tempo = tempo_total_APB_ord[0]
230     melhor_X = MXep_1[0]
231     melhor_Y = MYep_1[0]
232     tempo_min.append(melhor_tempo)  # Criado para ver a evolucao do
        melhor tempo de gera  o para geracao
233
```

```
234 import matplotlib.pyplot as plt
235
236 plt.figure(0)
237 plt.plot(tempo_min)
238
239 #     TESTE FUN   O PENALTY
240
241
242 MrPG_temp = MrPG[index]
243 MthetaPG_temp = MthetaPG[index]
244 vel_PG_temp = vel_PG[index]
245 penalty_PG, BEV, BOL, BEVDIFF, BOLSUM, MATPENAL, ANGVAR, VELMED =
    penalty(MrPG_temp[0, :], MthetaPG_temp[0],
246
          vel_PG_temp[0], aposv, pen)
247 # Penalizacao em segundos para melhor analise
248 # COMPARAR COM O TRA ADO DO TRAJECTO OPTIMO
249
250
251 #          GRAVACAO FICHEIRO .mat com dados dos calculos
252 #          para processamento futuro
253 import datetime
254
255 if isave == 1:
256     ref_h = datetime.datetime.now()
257     fName = 'otv_' + str(ref_h.date()) + '_hora_' + str(ref_h.hour)
    + '_' + str(ref_h.minute)
258     import shelve
259
260     my_shelf = shelve.open(fName, 'n')  # 'n' para novo
261     for key in dir():
262         try:
263             my_shelf[key] = globals()[key]
264         except TypeError:
265             #
266             # __builtins__, my_shelf, and imported modules can not
    be shelved.
267             #
268             print('ERROR shelving: {0}'.format(key))
269     my_shelf.close()
270 # Para recuperar
271 # my_shelf = shelve.open(filename)
272 # for key in my_shelf:
273 #     globals()[key]=my_shelf[key]
274 # my_shelf.close()
```

```
275
276 # PROCESSAMENTO GRAFICO PARA VISUALIZACAO RESULTADOS
277
278 # Plot para ver a evolucao do tempo
279 # Reparar que o pyplot j  foi importado
280
281 plt.style.use(['dark_background'])
282 plt.figure(1)
283 plt.plot(tempo_min, '-k', color='b', linewidth=2.5)
284 plt.xlabel('Ciclos')
285 plt.ylabel('Tempo(h)')
286 plt.title('Evolucao do tempo m nimo com o decorrer dos ciclos')
287 plt.legend()
288
289 iprint = 1
290
291 if iprint == 1:
292     ref_h = datetime.datetime.now()
293     fName = 'otv_' + str(ref_h.date()) + '_hora_' + str(ref_h.hour)
    + '_' + str(ref_h.minute)
294     plt.savefig('fname' + 'tempo_min')
295
296 # Plot dos trajectos A-B e A-P-B
297
298 plt.figure(2)
299 plt.scatter(A[0], A[1], marker='s', label='A')
300 plt.annotate('A', (A[0], A[1]))
301 pll1 = np.concatenate([np.array([A[0]]), melhor_X, np.array([B[0]])
    ])
302 pll2 = np.concatenate([np.array([A[1]]), melhor_Y, np.array([B[1]])
    ])
303 plt.plot(pll1, pll2, 'o-', label='P')
304 # plt.plot(A[0], A[1], 'x', melhor_X, melhor_Y, 'o--', B[0], B[1],
    's')
305 plt.scatter(B[0], B[1], marker='v', label='B')
306 plt.xlabel('Coordenada X')
307 plt.ylabel('Coordenada Y')
308 plt.title('Trajetos A-B e A-P-B')
309 # ax2.quiver(A[0],A[1],B[0]-A[0],B[1]-A[1],155)
310 plt.arrow(A[0], A[1], B[0] - A[0], B[1] - A[1], color='w',
311           head_width=1, length_includes_head=True, linestyle='
    dashed',
312           overhang=1, head_length=8)
313
314 #  VENTO
```

```
315
316 px = np.sin(np.radians(aposv))
317 py = np.cos(np.radians(aposv))
318
319 from malhabase import malhabase
320
321 Xbase, Ybase, dAB, rAB, resolucao = malhabase(A, B, passos)
322 plt.quiver(Xbase, Ybase, 0 * Xbase + px, 0 * Ybase + py, color='r')
323 plt.legend()
324 plt.show()
325 # text(Xbase(1,1),Ybase(1,1)-resolucao/2,['Velocidade vento: ',
      num2str(vento),' n s'])
326 # if pen == 0
327 # text(Xbase(1,1),Ybase(end,end)-resolucao/2,['Tempo trajecto sem
      penaliza o: ',num2str(melhor_tempo),' horas'])
328 # else
329 # text(Xbase(1,1),Ybase(end,end)-resolucao/2,['Tempo trajecto com
      penaliza o: ',num2str(melhor_tempo),' horas'])
330 # end
331 # axis equal
332
333
334 #  if iprint==1;
335 #  %Guardar o gr fico
336 #  fName2=['ano_',num2str(ref_h(1)),'_mes_',num2str(ref_h(2)),'
      _dia_',num2str(ref_h(3)),'_hora_',num2str(ref_h(4)),'_min_',
      num2str(ref_h(5))];
337 #  fig5 = ['TRAJ_OPTIM_FIG_PS_',num2str(fName2)]; print(gcf, '-dpsc
      ', fig5 )
338 #  fig6 = ['TRAJ_OPTIM_FIG_DMETA_',num2str(fName2)]; print(gcf, '-
      dmeta', fig6)
339 #  fig7 = ['TRAJ_OPTIM_FIG_EPS_',num2str(fName2)]; print(gcf, '-
      depsc', fig7)
340 #  fig8 = ['TRAJ_OPTIM_FIG_MAT_',num2str(fName2)]; savefig(fig8)
```

Código Fonte C.1: OTV.py

```
from math import *
import numpy as np
import re

# fazer matriz: np.array([[linha1], [linha2], ...)
# multiplicar matrizes np.matmult(esq, dir)
# mult por escalar: k * A

A = np.array([0, -30])
B = np.array([0, 30])

passos: int = 5   # numero de subdivisoes de dAB (distancia de referencia)
numPG: int = 3

def EPbase(A, B, passos, numPG):
    """
    Criacao das matrizes com as coordenadas X (MXep)
    e coordenadas Y (MYep) dos pontos de guinada:

    Em cada linha da coluna j encontra-se a coordenada  X
    do ponto de guinada j.

     Em cada linha da coluna j encontra-se a coordenada Y
    do ponto de guinada j.

    dAB- distancia de A a B

    rAB- rumo de A a B (coordenadas polares [-180 a +180]
    """

    dAB = dist(A, B)

    rAB = degrees(atan2(B[1]-A[1], B[0]-A[0]))

    resolucao = dAB/passos

    dimEP = 2 * (passos**2)  #Dimensao do espa o de pesquisa (numero par)

    menorX = min(A[0], B[0])
    maiorX = max(A[0], B[0])
    menorY = min(A[1], B[1])
    maiorY = max(A[1], B[1])
```

```
43
44      MXep = [[] for _ in range(dimEP)]
45      MYep = [[] for _ in range(dimEP)]
46      for i in range(0, numPG):
47
48          Xbase = (menorX - dAB) + ((maiorX + dAB)-(menorX - dAB)) *
    np.random.random([dimEP, 1])
49          Ybase = (menorY - dAB) + ((maiorY + dAB) - (menorY - dAB))
    * np.random.random([dimEP, 1])
50          MXep = np.hstack([MXep, Xbase])
51          MYep = np.hstack([MYep, Ybase])
52
53      return MXep, MYep, dAB, rAB, resolucao, dimEP
54
55  print(EPbase(A, B, passos, numPG))
```

Código Fonte C.2: Epbase.py

```
import numpy as np

def malhabase(A,B,passos):
#  dAB- distancia de A a B
# rAB- rumo de A a B (coordenadas polares [-180 a +180])
    dAB = np.linalg.norm(B-A)
    rAB = np.degrees(np.arctan2(B[1]-A[1],B[0]-A[0]))
    resolucao = dAB/passos
    menorX=np.min([A[0],B[0]])
    maiorX=np.max([A[0],B[0]])
    menorY=np.min([A[1],B[1]])
    maiorY=np.max([A[1],B[1]])

    vectorX=np.arange(menorX-dAB,maiorX+dAB,resolucao)
    vectorY=np.arange(menorY-dAB,maiorY+dAB,resolucao)

    Xbase, Ybase = np.meshgrid(vectorX,vectorY)
    return Xbase, Ybase, dAB, rAB, resolucao
```

Código Fonte C.3: Malhabase.py

```python
import numpy as np


def penalty(MrPG, MthetaPG, vel_PG, aposv, pen):
    # Quando MrPG vem um vetor 4x1, MrPG.shape aparece como (4,) e isso frita o n
    # Se tamanho MrPG < 2 (pq (4,) tem tamanho 1) entao passa m, n = 4, 1
    if len(MrPG.shape) < 2:
        MrPG = np.array([MrPG])

    if len(vel_PG.shape) < 2:
        vel_PG = np.array([vel_PG])

    (m, n) = MrPG.shape

    k1 = 20 / 1852
    k2 = np.log(20) / 20
    aposv = np.radians(aposv)

    aposv_cp = np.degrees(np.arctan2(np.cos(aposv), np.sin(aposv)))

    if aposv_cp >= 0:
        adosv = aposv_cp - 180
    else:
        adosv = aposv_cp + 180

    adosv = np.ones((m, n)) * adosv
    TEMP = MrPG - adosv
    BEV = TEMP.copy()
    # logic1 = np.logical_and(BEV >= -360, BEV <= -180)
    # logic2 = np.logical_and(BEV >= -180, BEV <= 0)
    # logic3 = np.logical_and(BEV >= 0, BEV <= 180)
    # logic4 = np.logical_and(BEV >= 180, BEV <= 360)
    #
    # BEV[logic1] = 1
    # BEV[logic2] = -1
    # BEV[logic3] = 1
    # BEV[logic4] = -1

    np.where(np.logical_and(BEV >= -360, BEV <= -180), 1, BEV)
    np.where(np.logical_and(BEV >= -180, BEV <= 0), -1, BEV)
    np.where(np.logical_and(BEV >= 0, BEV <= 180), 1, BEV)
    np.where(np.logical_and(BEV >= 180, BEV <= 360), -1, BEV)
```

```
43
44      BEVDIFF = np.abs(np.diff(BEV) / 2)
45
46      BOL = MthetaPG.copy()
47      # logic_bol = np.logical_and(BOL >= 0, BOL <= 95)
48      # BOL = BOL * 0
49      # BOL[logic_bol] = 1
50      np.where(np.logical_and(BOL >= 0, BOL <= 95), 1, 0)
51      if BOL.ndim == 1:
52          BOL1 = BOL[1:]
53          BOL2 = BOL[:-1]
54      else:
55          BOL1 = BOL[:, 1:]
56          BOL2 = BOL[:, :-1]
57
58      BOLSUM = BOL1 + BOL2
59      np.where(np.logical_and(BOLSUM >= 1.5, BOLSUM < 2.5), 1, BOLSUM
        )
60
61      MATPENAL = np.floor((BOLSUM + BEVDIFF) / 2)
62
63      MrPG_temp1 = MrPG[:, 1:]
64      MrPG_temp2 = MrPG[:, :-1]
65
66      ANGVAR = np.floor(np.abs(MrPG_temp1 - MrPG_temp2) / 2.5)
67
68      vel_PG_temp1 = vel_PG[:, 1:]
69      vel_PG_temp2 = vel_PG[:, :-1]
70
71      VELMED = (vel_PG_temp1 + vel_PG_temp2) / 2
72
73      penalty_PG = k1 * np.exp(-k2 * VELMED) * MATPENAL * ANGVAR / 90
74      if pen == 0:
75          penalty_PG = penalty_PG * 0
76          BEV = BEV * 0
77          BOL = BOL * 0
78          BEVDIFF = BEVDIFF * 0
79          BOLSUM = BOLSUM * 0
80          MATPENAL = MATPENAL * 0
81          ANGVAR = ANGVAR * 0
82          VELMED = VELMED * 0
83      return penalty_PG, BEV, BOL, BEVDIFF, BOLSUM, MATPENAL, ANGVAR,
        VELMED
```

Código Fonte C.4: Penalty.py

```
import numpy as np

def theta0P(rAB,aposv):
#  rAB rumo de A para B em coordenadas polares [-180 a +180]
#  aposv-Azimute Para Onde Sopra o Vento
#  adosv-Azimute De Onde Sopra o Vento
#  thetaAB angulos de entrada no diagrama polar
#  Conversao de "aposv" (0[N] a 360) em "adosv" [-180 0[E] +180[
    aposv = np.radians(aposv)
    rAB = np.radians(rAB)
    aposv_cp = np.degrees(np.arctan2(np.cos(aposv),np.sin(aposv)))#
    ;% Azimute aposv em coordenadas polares

    if aposv_cp >= 0:
        adosv = aposv_cp-180
    else:
        adosv = aposv_cp+180
    adosv_2 = np.radians(adosv)

    thetaAB = np.degrees(np.arccos(np.cos(rAB)*np.cos(adosv_2)+np.
    sin(rAB)*np.sin(adosv_2)))

    return thetaAB, adosv
```

Código Fonte C.5: theta0P.py

```
import numpy as np

def thetaPG(MrPG,aposv):
    (m,n) = MrPG.shape
    aposv_cp = np.arctan2(np.cos(np.radians(aposv)),np.sin(np.
   radians(aposv)))
    aposv_cp = np.degrees(aposv_cp)

    if aposv_cp >= 0:
        adosv = aposv_cp-180
    else:
        adosv = aposv_cp+180

    adosv = np.ones((m,n))*adosv

    MthetaPG = np.arccos(np.cos(np.radians(MrPG))*np.cos(np.radians
   (adosv))+np.sin(np.radians(MrPG))*np.sin(np.radians(adosv)))
    MthetaPG = np.degrees(MthetaPG)

    return MthetaPG
```

Código Fonte C.6: thetaPG.py

```
from scipy.interpolate import CubicSpline, interp1d
import numpy as np

def gera_diag_polar(vento, t='spline'):
    x = np.array([0, 45, 60, 75, 90, 105, 120, 135, 150, 175, 180])
    if (0<= vento < 6):
        y6=np.array([0.0 , 1.0, 1.2, 1.6, 2.0, 1.8, 1.6, 1.4, 1.3,
   1.2, 1.0])
        y=y6*vento/6
    elif (6 <= vento < 8):
        y6=np.array([0.0,1.0,1.2,1.6,2.0,1.8,1.6,1.4,1.3,1.2,1.0])
        y8=np.array([0.0,2.0,3.0,3.2,3.2,3.0,2.8,2.6,2.4,2.2,2.2])
        y=y6+(vento-6)*(y8-y6)/2
    elif (8 <= vento < 10):
        y8=np.array([0.0,2.0,3.0,3.2,3.2,3.0,2.8,2.6,2.4,2.2,2.2])
        y10=np.array([0.0,3.0,4.0,4.7,5.0,4.8,4.0,3.8,3.6,3.2,3.0])
        y=y8+(vento-8)*(y10-y8)/2
    elif (10 <= vento < 12):
        y10=np.array([0.0,3.0,4.0,4.7,5.0,4.8,4.0,3.8,3.6,3.2,3.0])
        y12=np.array([0.0,3.2,4.5,5.4,5.5,5.4,5.2,4.8,4.4,4.2,4.0])
        y=y10+(vento-10)*(y12-y10)/2
    elif (12 <= vento < 14):
        y12=np.array([0.0,3.2,4.5,5.4,5.5,5.4,5.2,4.8,4.4,4.2,4.0])
        y14=np.array([0.0,3.5,5.0,6.0,6.0,5.6,5.4,5.0,4.8,4.7,4.5])
        y=y12+(vento-12)*(y14-y12)/2
    elif (14 <= vento < 16):
        y14=np.array([0.0,3.5,5.0,6.0,6.0,5.6,5.4,5.0,4.8,4.7,4.5])
        y16=np.array([0.0,3.6,5.3,6.3,6.5,6.3,6.0,5.8,5.5,5.3,5.0])
        y=y14+(vento-14)*(y16-y14)/2
    elif (16 <= vento < 20):
        y16=np.array([0.0,3.6,5.3,6.3,6.5,6.3,6.0,5.8,5.5,5.3,5.0])
        y20=np.array([0.0,4.0,5.5,6.5,7.0,6.8,6.5,6.2,5.8,5.6,5.5])
        y=y16+(vento-16)*(y20-y16)/4
    elif (20 <= vento <= 25):
        y20=np.array([0.0,4.0,5.5,6.5,7.0,6.8,6.5,6.2,5.8,5.6,5.5])
        y25=np.array([0.0,4.2,5.8,6.8,7.2,7.2,7.0,6.7,6.4,6.2,6.0])
        y=y20+(vento-20)*(y25-y20)/5
    else:
        print('error: Number between 0 and 25') # ou este ou os que
    est o abaixo!!!
    # else vento < 0:
    #     vento = 0
    #     y6 = np.array([0,0,0,0,0,0,0,0,0,0,0])
    # else:
```

```
43      #      y = np.array([0,0,0,0,0,0,0,0,0,0]) # reparar este
        problem
44
45
46      if t == 'spline':
47          diag_polar = CubicSpline(x,y)
48      elif t == 'linear':
49          diag_polar = interp1d(x,y,'linear')
50      # Para fun   o sem scipy http://www.astropython.org/snippets/
        interpolation-without-scipy90/
51
52      return diag_polar, x, y
```

Código Fonte C.7: gera$_d$*iag*$_p$*olar.py*

```
import numpy as np

def trajectoPG(MXep,MYep,A,B):
    # function [MdPG,MrPG]=trajectoPG(MXep,MYep,A,B)
    # A fun ao trajectoPG gera trajectos (distancia e rumo)
    #  dA_P1, ..., dPi_Pi+1,...,dPN_B
    #
    #  MdPG-Matriz das distancias
    #  MrPG-Matriz dos rumos

    #  Obs: Os rumos sao calculados em
    #  em coordenadas polares [-180 a +180]
    (m,n) = MXep.shape
    # m-numero de trajectos (dimensao do espa o de pesquisa)
    # n-numero de pontos de guidada
    Amx = np.ones(m)*A[0].reshape(1,-1) # % Vectores coluna com as
   coordenadas de A e B
    Amy = np.ones(m)*A[1].reshape(1,-1)
    Bmx = np.ones(m)*B[0].reshape(1,-1)
    Bmy = np.ones(m)*B[1].reshape(1,-1)

    MXep = np.hstack((Amx.T,MXep,Bmx.T))
    MYep = np.hstack((Amy.T,MYep,Bmy.T))

    MdPG = np.zeros((m,n+1))
    MrPG = np.zeros((m,n+1))

    for i in range(0,n+1,1):
        dPG = np.sqrt(np.square(MXep[:,i]-MXep[:,i+1])+np.square(
   MYep[:,i]-MYep[:,i+1]))
        MdPG[:,i]=dPG
        rPG = np.arctan2(MYep[:,i+1]-MYep[:,i],MXep[:,i+1]-MXep[:,i
   ])
        MrPG[:,i]=np.degrees(rPG)
    return MdPG , MrPG
```

Código Fonte C.8: trajectoPG.py

Apêndice D - Gráficos para análise de robustez

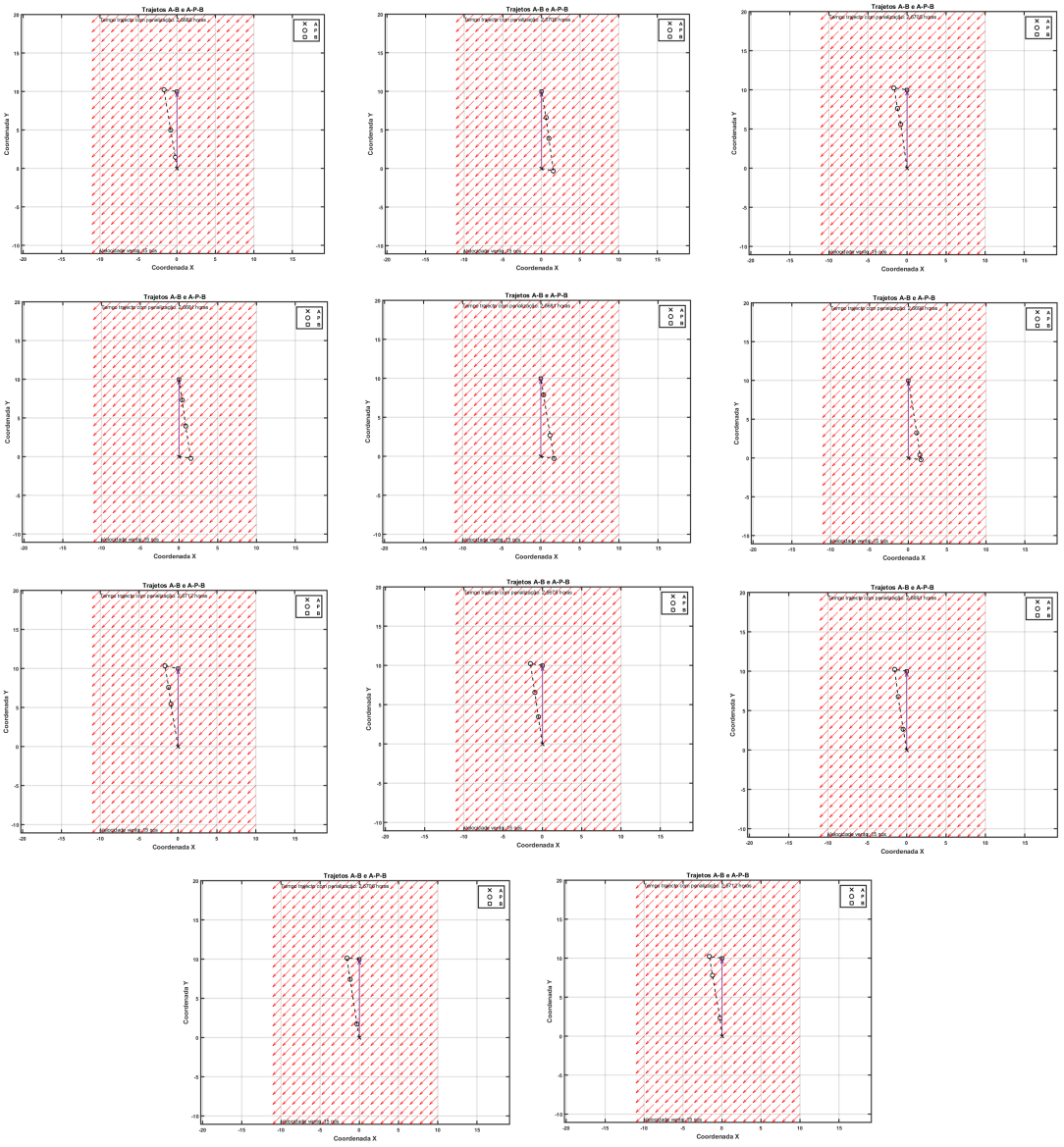


FIGURA D.1: Gráficos para análise de robustez do Programa com vento de 045 a 15 nós com três pontos de guinada.

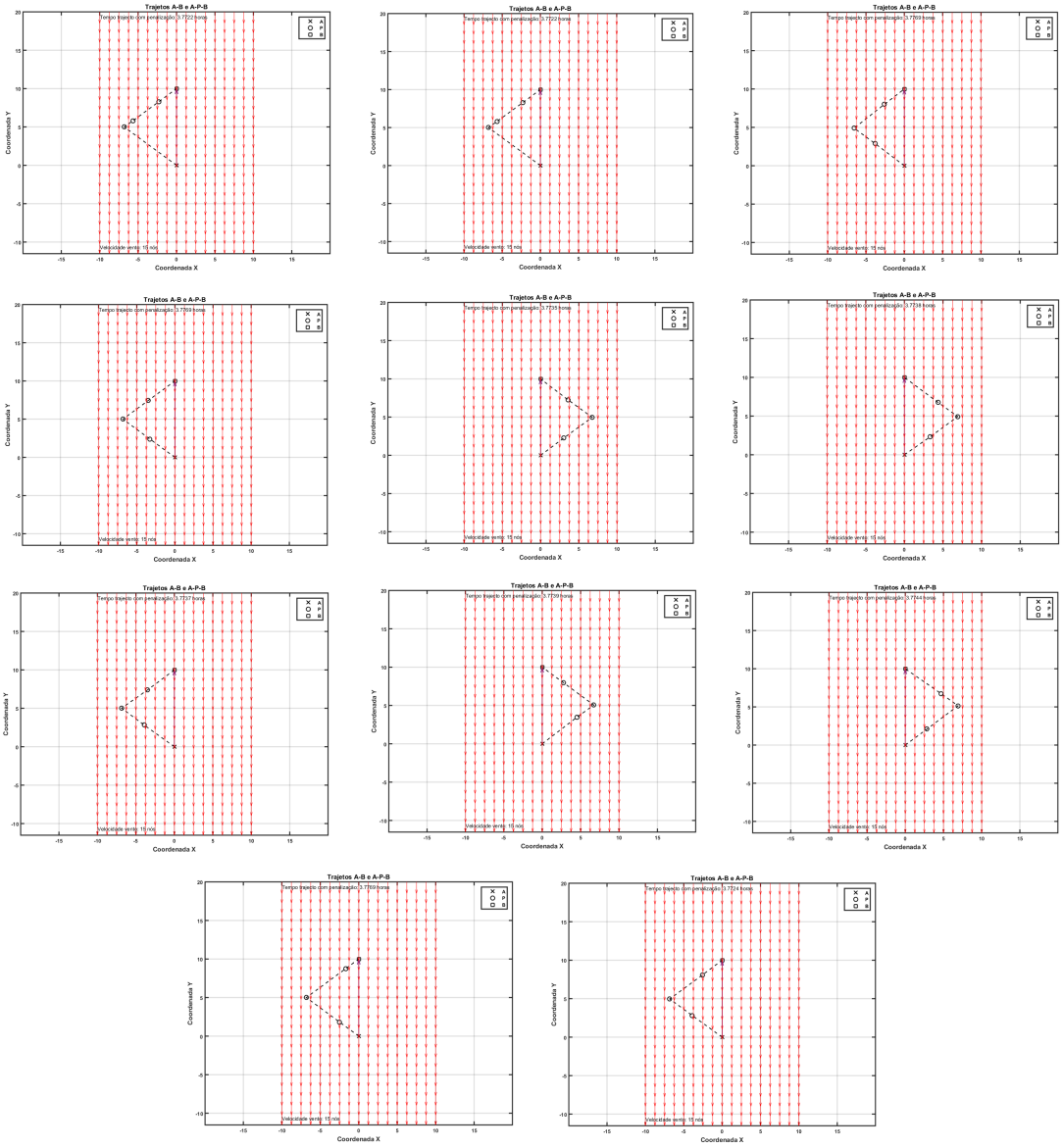


FIGURA D.2: Gráficos para análise de robustez do Programa com vento de Norte a 15 nós com três pontos de guinada.

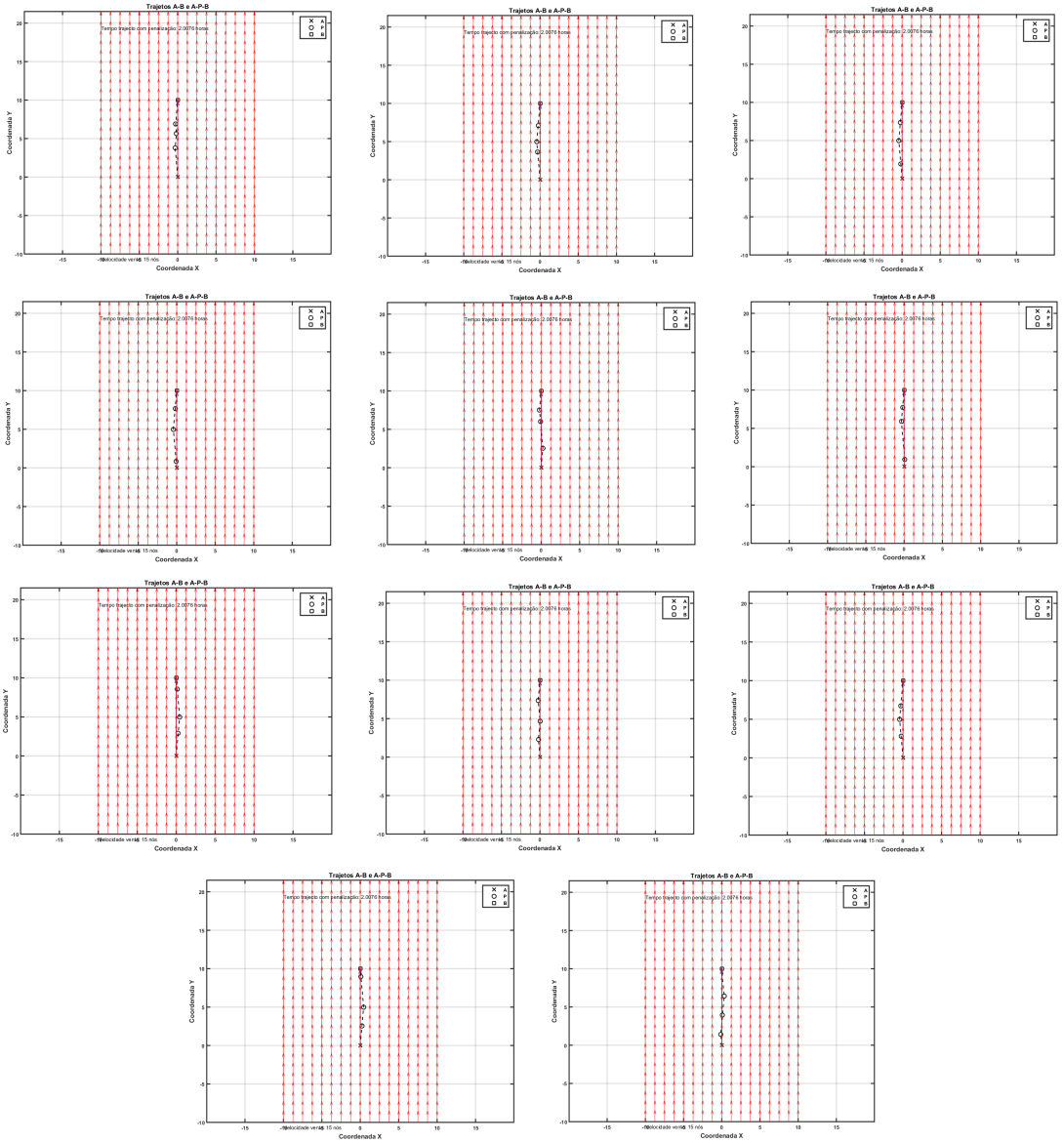


FIGURA D.3: Gráficos para análise de robustez do Programa com vento de Popa a 15 nós com três pontos de guinada.

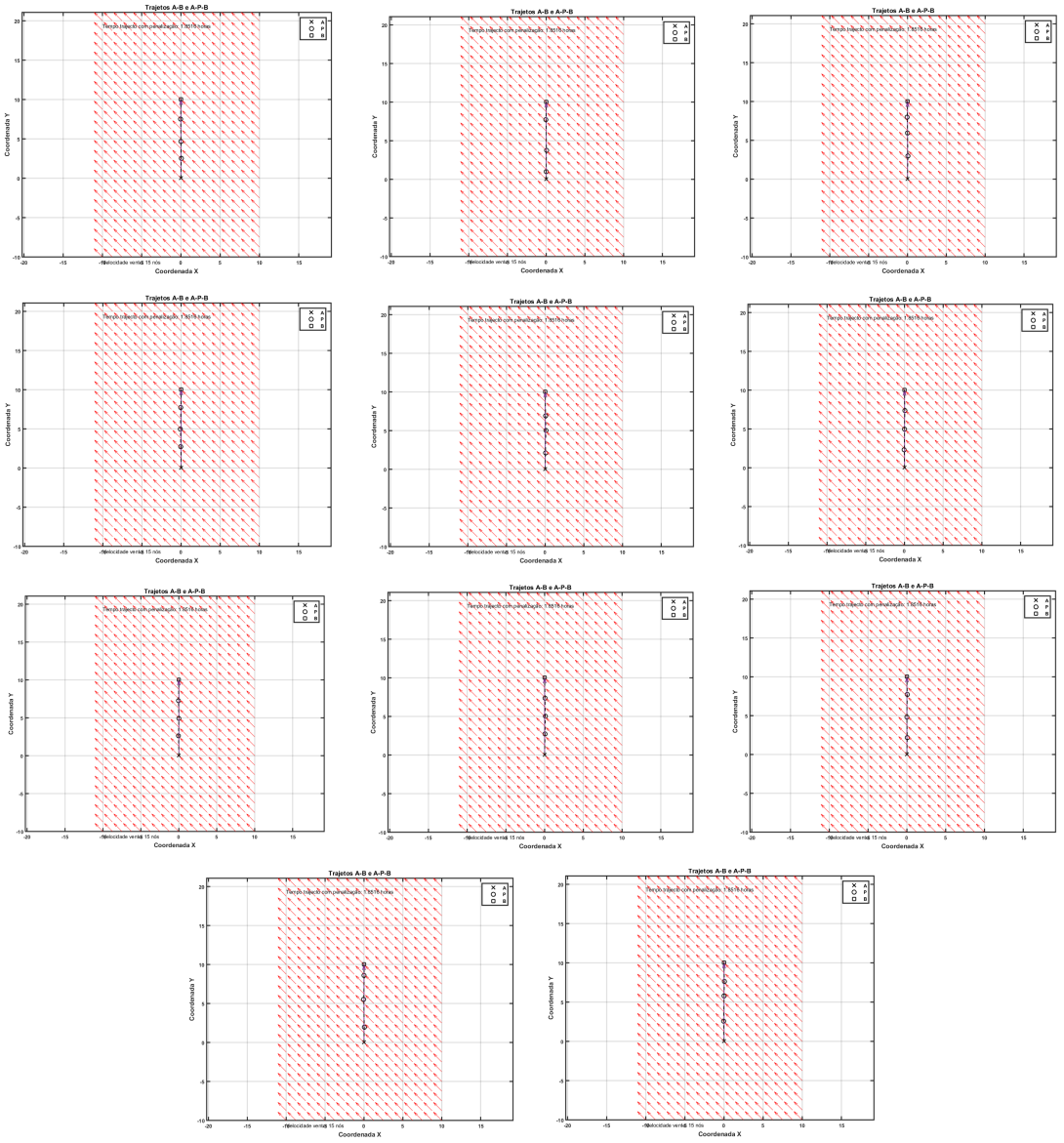


FIGURA D.4: Gráficos para análise de robustez do Programa com vento de 315 a 15 nós com três pontos de guinada.

Apêndice E - Valores para o Cálculo do Tempo de CPU em segundos

CPU (s) para 3 pontos de guinada - COM penalização - 15 nós										
Vento		**Vento**		**Vento**		**Vento**		**Vento**	
Norte	0,0926	**de 045**	0,9615	**de 090**	0,9173	**de 135**	0,7278	**de 180**	0,6597
	0,887		0,9374		0,8886		0,6879		0,662
	0,9183		1,032		0,8991		0,7417		0,6604
	0,8577		0,9753		0,9382		0,7882		0,6698
	0,902		0,9558		0,9186		0,7173		0,6679
	0,9019		1,0001		0,9423		0,7229		0,6524
	0,9016		1,0162		0,9084		0,6935		0,6529
	0,8947		1,0093		1,0118		0,7077		0,6612
	0,8994		0,9532		0,9235		0,79		0,645
	0,8981		0,9554		0,9159		0,7068		0,6256
Média:	**0,8153**		**0,9796**		**0,9264**		**0,7284**		**0,6557**

CPU (s) para 3 pontos de guinada - SEM penalização - 15 nós										
Vento		**Vento**		**Vento**		**Vento**		**Vento**	
Norte	0,4385	**de 045**	0,4369	**de 090**	0,461	**de 135**	0,4621	**de 180**	0,4607
	0,4763		0,4715		0,4402		0,46		0,4418
	0,4466		0,4523		0,4456		0,4277		0,4566
	0,4524		0,4464		0,4485		0,4791		0,4482
	0,456		0,4481		0,4473		0,438		0,4709
	0,4734		0,4826		0,4341		0,4526		0,4479
	0,4581		0,4441		0,4507		0,4249		0,4538
	0,4313		0,4332		0,4875		0,4505		0,4851
	0,4683		0,4499		0,4495		0,4501		0,4456
	0,4704		0,5175		0,4233		0,481		0,4539
Média:	**0,4571**		**0,4583**		**0,4488**		**0,4526**		**0,4565**

CPU (s) para 5 pontos de guinada - COM penalização - 15 nós										
Vento		**Vento**		**Vento**		**Vento**		**Vento**	
Norte	1,5946	**de 045**	1,4911	**de 090**	1,5455	**de 135**	1,0656	**de 180**	0,9388
	1,4564		1,4516		1,5298		1,0619		0,9008
	1,4306		1,5105		1,5342		1,0582		0,9117
	1,4454		1,5017		1,5098		1,073		0,9216
	1,4317		1,5472		1,5243		1,0428		0,9265
	1,4504		1,649		1,5372		1,0467		0,8905
	1,3942		1,476		1,5235		1,0676		0,8959
	1,486		1,4585		1,4949		1,0475		0,8888
	1,5014		1,4806		1,4907		1,0371		0,9118
	1,4567		1,4973		1,5172		1,0413		0,9258
Média:	**1,4647**		**1,5064**		**1,5207**		**1,0542**		**0,9112**

CPU (s) para 5 pontos de guinada - SEM penalização - 15 nós									
Vento		**Vento**		**Vento**		**Vento**		**Vento**	
Norte	0,5176	**de 045**	0,5201	**de 090**	0,4872	**de 135**	0,4953	**de 180**	0,508
	0,5077		0,5319		0,4962		0,5015		0,5227
	0,5145		0,5098		0,4976		0,5153		0,5151
	0,5365		0,5199		0,4982		0,4976		0,5119
	0,5233		0,5073		0,5001		0,5101		0,5149
	0,5162		0,5181		0,5326		0,497		0,5047
	0,5265		0,5483		0,5048		0,5216		0,5013
	0,5108		0,5286		0,4886		0,497		0,5054
	0,5326		0,5136		0,5018		0,5014		0,542
	0,5065		0,535		0,4776		0,4963		0,5425
Média:	**0,5192**		**0,5233**		**0,4985**		**0,5033**		**0,5169**

Anexo I - Fluxograma do Algoritmo A*

Após declarar as variáveis iniciais e vetores:

S - Posição ou nó inicial

Abertos - lista de nós ou posições que ainda não foram avaliadas e ordenadas

Fechados - lista de nós ou posições que já foram avaliadas e ordenadas

N - nó ou posição atual (determina se o objetivo foi alcançado ou é necessário prosseguir)

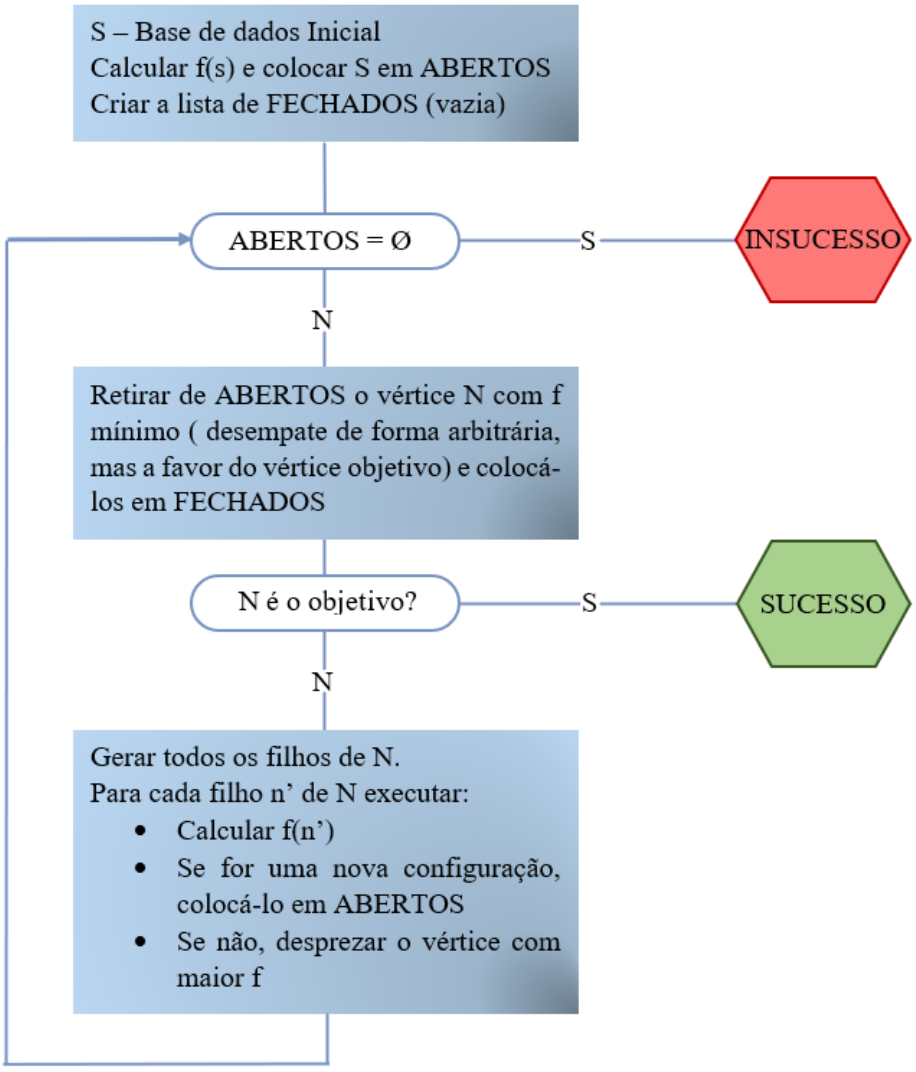


FIGURA I.1: Adaptado de (Zanchin, 2018).