João Miguel Ferreira Esteves

Licenciado em Ciências da Engenharia Eletrotécnica
e de Computadores

Desenvolvimento de um Sistema anticolisão para um veleiro com navegação autónoma

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: Professor Doutor Luís Filipe dos Santos Gomes,
Professor Associado com agregação,
Universidade Nova de Lisboa

Júri:

Presidente: Doutor Luís Augusto Bica Gomes de Oliveira

Arguentes: Doutor Tiago Oliveira Machado de Figueiredo Cardoso

Vogais: Doutor Luís Filipe dos Santos Gomes

Setembro, 2017

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Desenvolvimento de um Sistema anticolisão para um veleiro com navegação autónoma

Dedicado aos meus pais e avós que sempre souberam meter
o extra no ordinário e me ensinaram a ser um homem...
Dedicado também ao meu querido irmão
que já não se encontra entre nós...

Agradecimentos

Em primeiro lugar quero agradecer ao meu professor e orientador Luis Gomes por me ter apoiado ao longo desta dissertação e por ter estado sempre disponível a ajudar-me.

Quero também agradecer à professora Anikó Costa por possibilitar desenvolver nas suas aulas projetos que me ajudaram a fazer alguns testes práticos ao Hardware utilizado.

Também estou muito agradecido a ambos os professores pela energia positiva que me foram transmitindo ao longo deste trabalho tornando-o assim ainda mais entusiasmante.

À professora Ana Luisa Custódio devo um enorme agradecimento pela sua disponibilidade e prontidão na explicação de um problema matemático sobre o cruzamento entre vetores.

Não posso deixar de mencionar e agradecer a ajuda prestada pelo meu colega Hugo Marques na explicação de algumas partes do seu trabalho, trabalho esse que é referente ao veleiro em desenvolvimento nesta dissertação.

Gostava de agradecer também ao meu amigo Miguel Teixeira e à minha Mãe por me terem ajudado na revisão do texto escrito.

Agradeço aos meus pais que me ajudaram a chegar até aqui, pois sem eles este percurso não teria sido possível.

Dedico especialmente este meu trabalho ao meu querido irmão que foi forçado a abandonar-nos com 19 anos no dia 22 de Julho de 2017.

E quero oficializar neste documento a seguinte mensagem direcionada a ele:

Zé! Quero acreditar que me ouves e que vais sentir todos os momentos da minha vida como se fossem teus também, porque é a minha vontade que assim o faças, por isso, cada conquista minha será tua e cada momento meu será teu também!!

Dito isto, tenho uma mensagem para ti!

Neste mundo existem pessoas e existem PESSOAS com letra grande e não é muito fácil no meio de tanta pessoa encontrar muitas que se destaquem, mas tu eras sem dúvida nenhuma uma dessas PESSOAS! Não digo isto por seres meu irmão, mas por teres uma personalidade fortíssima e única. Eras daquelas pessoas que se não estivesses de bom humor não metias um sorriso falso! Eras exatamente aquilo que se via quando se olhava para ti e não usavas nenhuma "máscara"! Eras sem duvida alguma uma das pessoas mais humildes que conheci, uma pessoa com um bom coração, uma pessoa com o coração no sítio. Eras muito bem educado com toda a gente! Era impossível não ser teu amigo, não tinhas maldade nenhuma e apesar de saberes que tinhas um corpo que metia qualquer um K.O, não eras nada de confusões! Eras muito estimadinho com tudo o que tinhas! Qualquer coisa nas tuas mãos era nova durante anos! Não te preocupes que vou continuar a estimá-las.

Eras sem dúvida um jovem muito inteligente, muito desenrascado, muito trabalhador, engenhocas, tão esperto, criativo, com grande sentido de humor e muito brincalhão. Eras mesmo um desenrascado muito a cima da média, tinhas aquela capacidade para pensar "fora da caixa", ver mais além e conseguias resolver qualquer problema com relativa facilidade.

Eu sei que eras bom aluno e tinhas as cadeiras todas em dia e também sei da surpresa que querias fazer agora à mãe com as tuas boas notas! Mas não te preocupes, ela também já sabe.

Tinhas uma força enorme de viver e querias viver tudo ao máximo! Sei que eras um amigo com quem podiam sempre contar, eras leal e verdadeiro. Adoravas os teus amigos e querias ir sempre para o café ter com eles! Estavas numa fase tão feliz da tua vida e todo contente com as coisinhas novas que tinhas comprado! Tentavas sempre seguir os bons exemplos que vias em mim e apesar de não mostrares muito eu sei que ouvias sempre os meus conselhos e dicas!

Todas as tuas qualidades são uma junção das qualidades das excelentes pessoas que te educaram e que te ajudaram a fazer o homem que eras e como tal o ótimo resultado está à vista de todos os que te conheceram.

Como não podia deixar de ser, herdaste da família a veia de empreendedor! A mãe contava muitas vezes em forma de piada que aos 6 anos ias ao galinheiro da avó tirar os ovos das galinhas e vender às vizinhas de porta em porta! Aos 15 anos começaste a ser apicultor como passatempo, mas rápido se tornou na tua paixão! A enorme quantidade de horas que passavas de volta das tuas abelhinhas muitas vezes até às tantas da manhã, em fóruns na internet e a conversar com os mais velhos, aliadas ao teu grande perfecionismo e profissionalismo fizeram com que o mel que produziste fosse mesmo um produto biológico de alta qualidade, sabor e perfeição! Não te preocupes porque vou tomar bem conta das tuas abelhinhas e ainda te vais rir muito dos disparates que irei fazer até aprender!

Só mais uma coisa Zé, fica descansado que vou tomar conta do pai e da mãe!!

Mudando agora de assunto, como uma vez disseste, "Na vida encara-se tudo de braços abertos", e é o que nós, família e amigos vamos fazer! Vais continuar a viver nas memórias de cada um de nós e prometo-te que nunca serás esquecido!

Ate breve Zé…

Abstract

In this dissertation, a collision avoidance system intended to be used on an autonomous sailboat scale model is presented.

The boat has a sequence of route points in the controller memory which are followed sequentially, as for a typical regatta contest. The developed system is based on a camera device named “Pixy” and a low- cost Arduino board.

The image data obtained from Pixy is edited considering the inclination of the sailboat, wind speed, water current and sea ripple and the number of obstacles. The data is then used and considered in a function that change the next destination point in presence of a collision warning.

The methods and functions were validated using MATLAB simulation and some of them implemented on Arduino.

Keywords: Sailboat, autonomous navigation, autonomous sailboat, Arduino, Pixy, Collision avoidance, Simulator, MATLAB

Resumo

Nesta dissertação é apresentado um sistema anticolisão para um modelo em pequena escala de um veleiro com navegação autónoma.

O veleiro tem uma lista de pontos de rota na memoria do seu controlador que são seguidos sequencialmente, tal como numa típica competição de regata. O sistema desenvolvido é baseado num dispositivo ótico chamado "Pixy" e numa placa Arduino de baixo custo.

Os dados de imagem obtidos da Pixy são editados tendo em consideração a inclinação do veleiro, velocidade do vento, corrente e ondulação do mar e o número de obstáculos. Os dados são então usados e considerados numa função que altera o próximo ponto de destino caso exista um aviso de colisão.

Os métodos e funções desenvolvidos foram validados utilizando simulações em MATLAB e alguns deles implementados no Arduino.

Palavras-chave: veleiro, navegação autónoma, veleiro autónomo, Arduino, Pixy, evitar colisão, simulador, MATLAB

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas

A	Ampere
AIS	Automatic Identification System
bits/seg	Bits por segundo
DC	Corrente continua
EEPROM	Electrically-Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GPS	Global Positioning System
Hz	Hertz
I/O	Input/Output
IRSC	International Robotic Sailing Conference
IRSR	International Robotic Sailing Regatta
LED	Light Emitting Diode
m/s	Metros por segundo
PWM	Pulse Width Modulation
SRAM	Static random access memory
UDP	User Datagram Protocol
USB	Universal Serial Bus
V	Volt
Wh	Watt hora
Wp	Watt pico
WRSC	World Robotic Sailing Championship

CAPÍTULO 1

INTRODUÇÃO

1.1 Motivação

Este trabalho surge da necessidade de tornar mais eficiente a autonomia de navegação do veleiro em desenvolvimento na Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa de forma a que o mesmo consiga realizar uma navegação segura e livre de colisões no âmbito de uma das provas da competição WRSC (World Robotic Sailing Championship) [1].

1.2 Competições de Veleiros Autónomos

Existem algumas competições de veleiros autónomos, entre as quais se destacam três, a WRSC/IRSC (World Robotic Sailing Championship/International Robotic Sailing Conference), a SailBot IRSR (International Robotic Sailing Regatta) e a Microtransat. Em seguida é efetuada uma breve descrição sobre cada uma delas.

1.2.1 WRSC/IRSC

A "World Robotic Sailing Championship and International Robotic Sailing Conference", WRSC/IRSC [1], foi criada em 2008 e tem ocorrido todos os anos desde então. Consiste numa competição aberta a veleiros autónomos não tripulados com um comprimento até 4 metros, que utilizem somente a energia do vento para se deslocar. De acordo com o regulamento de 2016 [2] (ano em que foi realizada em Viana do Castelo, Portugal), a competição estava organizada em 4 provas (corrida, manutenção de estação, área *scanning* e evitar obstáculos), tendo sido efetuada uma prova distinta em cada dia, sendo que o primeiro dia foi reservado

para as equipas prepararem as embarcações e fazerem testes com a mesma na área da competição. Esta competição está aberta a embarcações movidas somente através da força do vento. É permitido ainda a inclusão de uma turbina eólica de forma a aproveitar a energia do vento e converter a mesma em energia mecânica ou elétrica para uso direto nas hélices da embarcação, sendo, no entanto, proibido o uso de qualquer energia (nas hélices) acumulada a bordo que não tenha sido gerada pela própria embarcação durante a prova.

As embarcações podem utilizar qualquer tipo de casco e uma ou mais velas.

Em 2016, esta competição dividiu-se em duas classes. A classe dos micro-veleiros (MS – micro-sailboats), em que se baseia em pequenos veleiros com um comprimento e peso não superiores a 1,5 metros e 100 kg respetivamente. E a classe de veleiros (S - sailboats), em que se trata de veleiros autónomos que não se encaixam na categoria MS e que não ultrapassem os 4 metros de comprimento e os 300 kg de peso total.

1.2.2 SailBot, IRSR

A "International Robotic Sailing Regatta", IRSR [3], é uma competição de barcos à vela autónomos realizada na América do Norte destinada a concursos entre estudantes. O primeiro evento ocorreu em 2006 em Kingston.

1.2.3 Microtransat

A Microtransat [4] é uma competição transatlântica para veleiros totalmente autónomos e tem como principal objetivo promover a partilha de conhecimento sobre o tema entre os participantes.

Esta competição foi inicialmente criada em 2005 pelo Dr. Mark Neal da "Aberystwyth University" e pelo Dr. Yves Briere do "Institut Supérieure de lÁéronautique et de l'Espace, ISAE" em Tolouse, França. A primeira competição transatlântica deveria ter ocorrido em Portugal em 2008, mas foi adiada para 2010 na Irlanda. Esta competição divide-se em duas classes. Uma das classes é a de vela que utiliza somente energia eólica para propulsão e onde o comprimento máximo da embarcação permitido era de 4 metros até 2017 e de 2,4 metros a partir desse ano. A outra classe permite embarcações com qualquer fonte de propulsão e um comprimente máximo da embarcação de 2 metros.

1.3 Objetivos da dissertação

De modo a cumprir as necessidades da competição WRSC, em particular da sua prova de evitar obstáculos, o objetivo será desenvolver um sistema que detete e permita evitar as boias que intercetem a rota do veleiro e que o permita realizar um percurso previamente definido de forma segura e autónoma.

1.4 Contribuição

A contribuição deste projeto para o veleiro em desenvolvimento na Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa é o desenvolvimento do protótipo de um sistema anticolisão com base no sensor ótico Pixy que em trabalhos futuros será integrado no controlador do veleiro. Através do sensor Pixy, é possível identificar obstáculos que estejam na rota do veleiro. No fim de implementado no controlador do veleiro, o sistema desenvolvido tem como base a informação proveniente da Pixy fazendo com que a embarcação se desvie das boias, efetuando assim uma navegação segura. Dos trabalhos desenvolvidos resultou uma publicação em conferência internacional de reconhecido prestígio [5].

1.5 Estrutura

A dissertação está dividida em cinco capítulos e está estruturada da seguinte forma. O capitulo um é a introdução. O capitulo dois é sobre os trabalhos relacionados. Neste capitulo é feita uma breve apresentação de algumas embarcações à vela relevantes e também sobre o veleiro em desenvolvimento; são também abordados os diferentes tipos de sensores de deteção de obstáculos e as plataformas sugeridas para integrar o sistema deste projeto na embarcação em trabalhos futuros. O capitulo três é sobre o sistema desenvolvido. Neste capitulo são explicados os métodos e algoritmos desenvolvidos tanto para integração no Arduino referente à receção da imagem com a Pixy como para as simulações em MATLAB do sistema. O capitulo quatro apresenta as validações feitas através de simulações em MATLAB, as validações laboratoriais referentes à receção de imagem da Pixy integrada com o Arduino. Finamente no capitulo cinco apresentam-se as conclusões e os trabalhos futuros.

CAPÍTULO 2

TRABALHOS RELACIONADOS

2.1 Conceitos básicos de Navegação à Vela

A navegação à vela consiste em navegar uma embarcação em função da força do vento que incide nas suas velas de modo a que a mesma se desloque em direção ao rumo pretendido. Para tal, o piloto pode manobrar as velas, o mastro, a quilha e o leme.

A Tabela 2.1 contem as principais características da constituição do veleiro utilizado neste projeto.

Tabela 2.1: Constituição básica de uma embarcação à vela

Nome	Definição
Vela mestra	Maior vela da embarcação
Mastro	Haste perpendicular que sustenta as velas
Proa	Extremidade dianteira da embarcação
Popa	Extremidade traseira da embarcação
Quilha	Montada ao centro na parte de baixo e auxilia a estabilidade da embarcação
Leme	Montado na traseira na parte de baixo e serve para controlar a direção da embarcação
Estai	Vela localizada na proa da embarcação
Bombordo	Lado esquerdo da embarcação
Estibordo	Lado direito da embarcação
Retranca	Haste que prende a parte inferior da vela
Escota	Cabo que controla a retranca

Na Figura 2.1 está representado o modelo da embarcação à vela relacionada com neste projeto.

Figura 2.1 - Veleiro utilizado neste projeto. Imagem base [6]

2.2 Projetos relacionados

Já foram desenvolvidos alguns projetos em veleiros autónomos, entre os quais se destacam o FASt, o Avalon e o ASV Roboat. Em seguida é feito um resumo dos dois primeiros visto que o FASt é um projeto português e o Avalon possui um sistema anticolisão. O FASt também tem como âmbito a navegação autónoma, mas não possui um arsenal de sensores tão composto como o Avalon.

É também feita em seguida uma análise e resumo do veleiro relacionado com este projeto.

2.2.1 Veleiro Autónomo FASt, FEUP

Este projeto [7] iniciou-se em Portugal em 2007 pelo professor João Carlos Alves com o apoio de um grupo de alunos de Engenharia Eletrotécnica e de Computação da Faculdade de Engenharia da Universidade do Porto (FEUP). FASt é um veleiro autónomo com um comprimento de 2,5 metros que teve como primeiro objetivo participar nas competições internacionais organizadas pela Microtransat [4]. Um ano após o seu inicio, o veleiro participou na primeira competição WRSC/IRSC [1] realizada em Breitenbrunn, Áustria. A segunda edição desta competição foi organizada pela FEUP em Matosinhos, Portugal. Em 2012 ganhou o primeiro lugar na mesma competição em Cardiff no Reino Unido e em 2014 em Galway, Irlanda conquistou o segundo lugar do campeonato do Mundo na corrida de resistência.

Figura 2.2 - Veleiro Autónomo FASt, FEUP [7]

Este veleiro possui ainda painéis solares de 45 Wp e duas baterias de iões de lítio com uma capacidade total de 190 Wh.

Possui um pequeno computador de bordo de baixo consumo energético baseado numa FPGA (Field Programmable Gate Array) onde ocorre todo o processamento computacional principal do veleiro. Este sistema inclui um microprocessador 32-bit RISC que opera a uma frequência máxima de 50 MHz.

O sistema de comunicações inclui uma antena WiFi (LinkSys WRT54GC) para comunicações de pequeno alcance com um computador pessoal, um modem GSM (Siemens MC35) para transmissões de pequenos pacotes de informação a curto alcance, um modem IRIDIUM SBD (modelo 9601) para o mesmo efeito que o anterior mas para grandes distâncias (escala mundial, graças à comunicação via satélite) e um recetor de radio que permite uma condução controlada manualmente.

É munido de vários sensores, tais como anemómetro, GPS, bússola digital, cata vento, acelerómetro, sensor de ângulo da vela, sensor de posição do vento, inclinómetro, monitores de tensão, sensor de luz ambiente, termómetro e sensores de água.

Através de dois servo-motores, dois lemes são controlados de forma independente, e tanto a vela buja como a mestra são controladas simultaneamente por um motor DC.

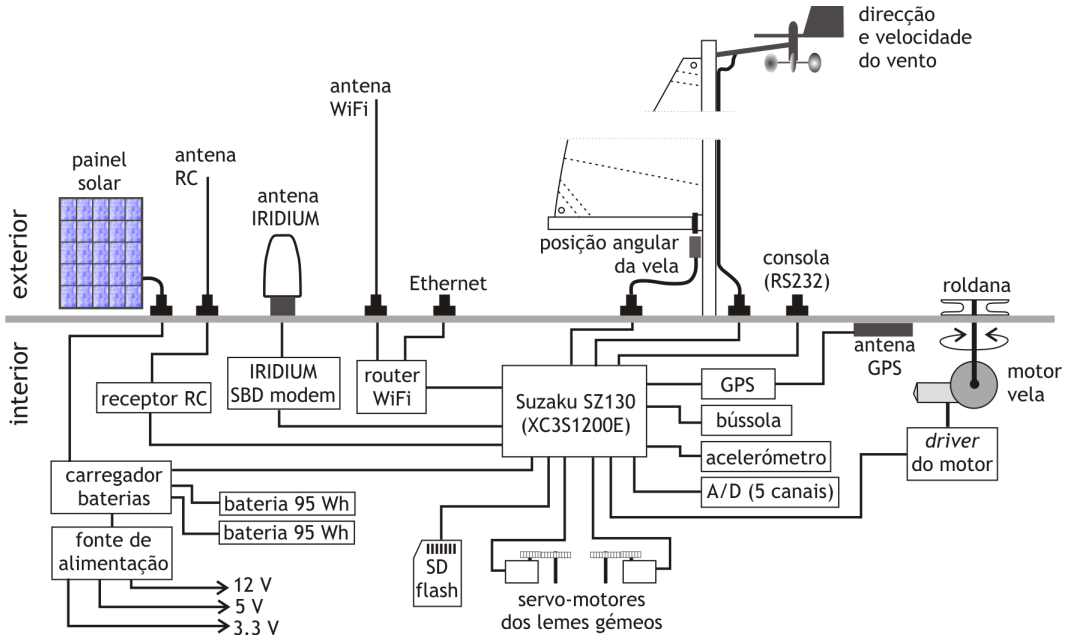

Figura 2.3 - Sistema eletrónico do FASt [8]

O sistema operativo utilizado pelo FASt no computador de bordo é o uCLinux que é uma versão do sistema operativo Linux e o desenvolvimento do software utilizado foi feito em linguagem C utilizando bibliotecas do Linux.

Este software de controle é formado por um conjunto de processos que comunicam entre si através de sockets UDP (User Datagram Protocol).

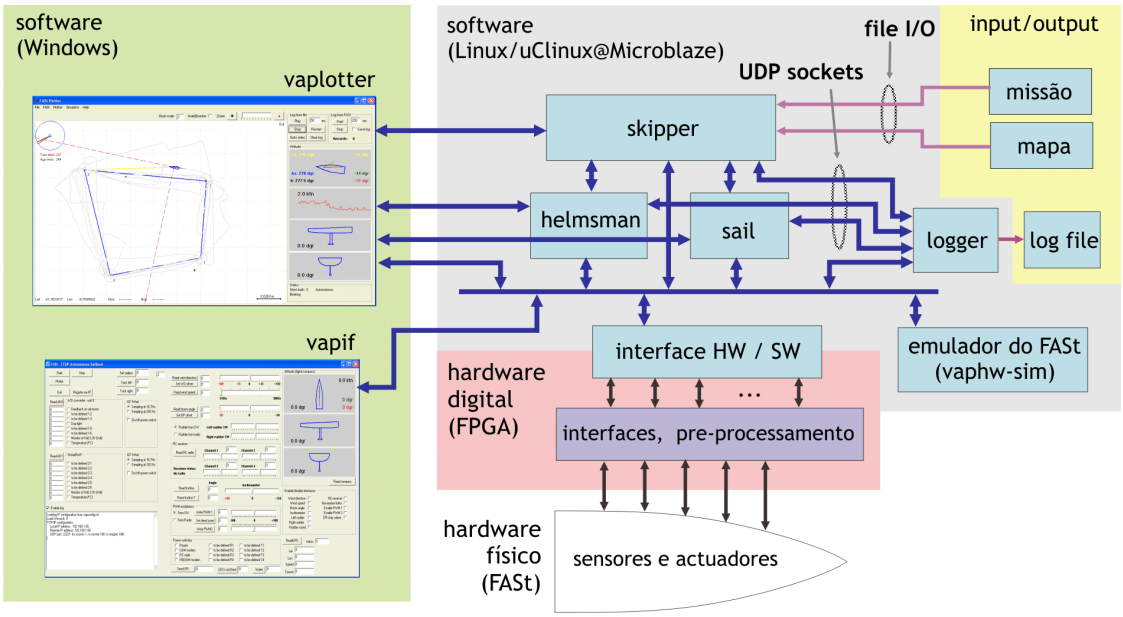

Figura 2.4 - Organização do software do FASt [8]

2.2.2 Avalon

O Avalon é um veleiro autónomo e também foi projetado para participar na competição Microtransat [9] de forma a não só possuir uma navegação autónoma como também a evitar tanto obstáculos estáticos como dinâmicos.

Figura 2.5 - Veleiro autónomo Avalon [9]

A parte principal do hardware é um computador de bordo MPC21 da DigitalLogic, que é um dispositivo que funciona a 500 MHz, possui uma RAM de 1024 MB e um disco rígido flash. Alem de outros sensores o Avalon possui um GPS, um sensor de ultrassom e um sensor AIS (Automatic Identification System). Este último sensor recebe os dados da posição e da velocidade dos barcos mais próximos em muito alta frequência (VHF) e é usado no sistema de anticolisão.

Para comunicações via satélite, a embarcação possui um modem IRIDIUM 9522-B. Também é equipado com painéis solares que produzem um total de 360 Wp e um conjunto de baterias com uma capacidade total de 2,4 KWh. Este veleiro, possui ainda um motor de 200 W para controlar a vela e dois motores de 150 W para controlar os lemes.

O software do Avalon baseia-se na DDX (Dynamic Data eXChange), que é executada no Linux.

2.2.3 Veleiro relacionado com este projeto

O veleiro relacionado com este projeto [10] foi inicialmente construído com o objetivo de participar na competição WRSC/IRSC, "World Robotic Sailing Championship and International Robotic Sailing Conference".

O seu hardware é composto por um microprocessador Arduino e diversos sensores, entre os quais uma Bússola, um cata-vento, um GPS, um recetor radio (que possibilita o controlo manual da embarcação) e uma antena WiFi de modo a trocar dados com um computador pessoal e auxiliar assim a navegação autónoma. Possui ainda dois motores servos que vão atuar na vela e no leme [11].

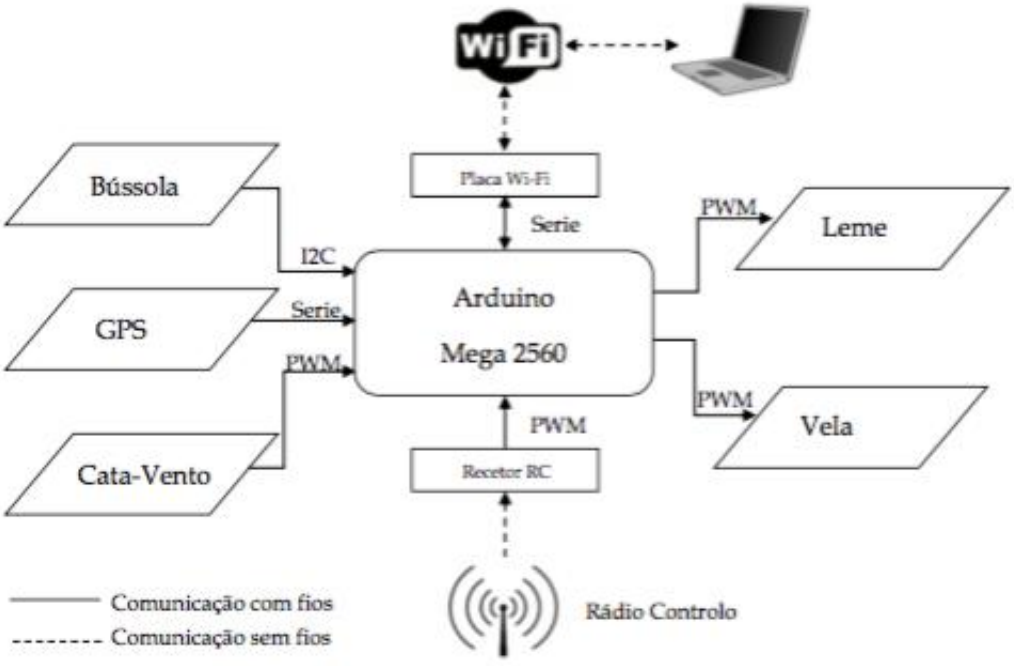

Figura 2.6 - Arquitetura de Hardware embutido [11]

O seu sistema de controlo é dividido em três níveis hierárquicos (Figura 2.7), capazes de atuar independentemente. O nível 1 é o que controla o leme e a vela. O nível 2 entra em ação quando o vento se encontra desfavorável, definindo a estratégia de navegação para cada segmento proveniente do nível 3, criando pontos intermédios entre cada segmento. No nível 3 é onde é definida a missão, este nível disponibiliza uma sequencia de segmentos de percurso aos outros níveis [10].

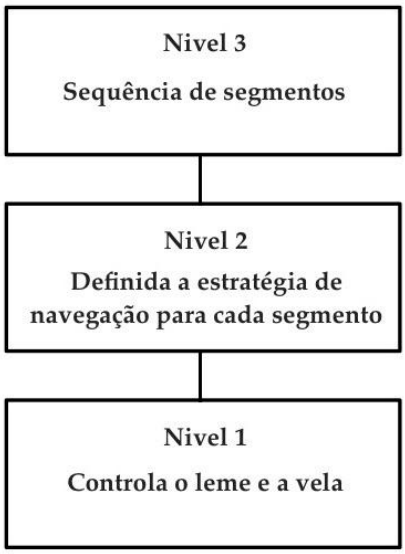

Figura 2.7 - Esquema dos níveis do software do veleiro em estudo

A informação proveniente dos sensores é analisada utilizando lógica difusa contribuindo assim esta análise na estratégia de navegação [12], [10], [13].

Nesta dissertação será desenvolvido e simulado em Matlab um sistema de anticolisão que em trabalhos futuros será acrescentado numa função no nível 2 e que irá fazer a verificação da existência de perigo de colisão. Caso exista perigo, essa função irá alterar a coordenada de destino. Assim que já não exista perigo de colisão o valor da coordenada de destino inicial será reposto.

2.2.3.1 Definição da estratégia de navegação

Considerando a geometria apresentada na Figura 2.8 onde várias regiões são consideradas, proposto em [13], cada segmento de navegação é caracterizado por um conjunto de pontos, nomeadamente o ponto inicial A, o ponto de destino B, bem como a linha da meta B1-B2. É importante notar que a distância entre B1 e B, e B e B2 (denominada d2 e d3, respetivamente, Figura 2.8) pode ser diferente. Esta caracterização permite uma adaptação fácil a diferentes requisitos, ou seja, quando a linha da meta B1-B2 representa a linha da meta final (e onde d2 e d3 seriam provavelmente iguais), bem como as metas intermediárias (como a que gira em torno de uma boia, onde o ponto de destino B não está longe da boia e B1 pode impor uma distância mínima para a boia, e onde B2 pode permitir alguma flexibilidade para se ajustar a restrições de navegação específicas).

A estratégia de navegação define uma área onde a navegação é possível (regiões 1 e 2 da Figura 2.8). Começando com a definição da trajetória ideal desejada (linha l0 na Figura 2.8), duas linhas paralelas igualmente espaçadas são definidas considerando a distância d1 a l0 (linhas l1 e l2 na Figura 2.8). As duas linhas (l4 e l5 na Figura 2.8) foram definidas com um ângulo específico em relação à linha l3 (que inclui a linha da meta), e permitindo a deteção de deslocamento para fora da área de navegação.

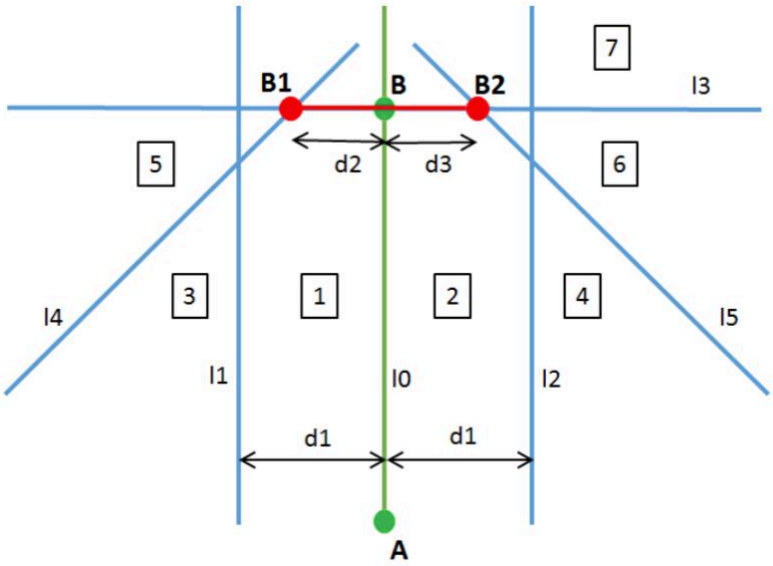

Figura 2.8 - Caracterização geométrica do corredor de navegação considerando a posição inicial A e a linha da meta B1-B2 [13].

Com base na caracterização geométrica apresentada na Figura 2.8 e na relação da direção do vento com a direção do objetivo, o modo de navegação adequado é escolhido de modo a evitar a zona de não-navegação (zonas 3, 4, 5 e 6 da Figura 2.8). A Figura 2.9 resume o comportamento do veleiro utilizando uma notação de diagrama de estados.

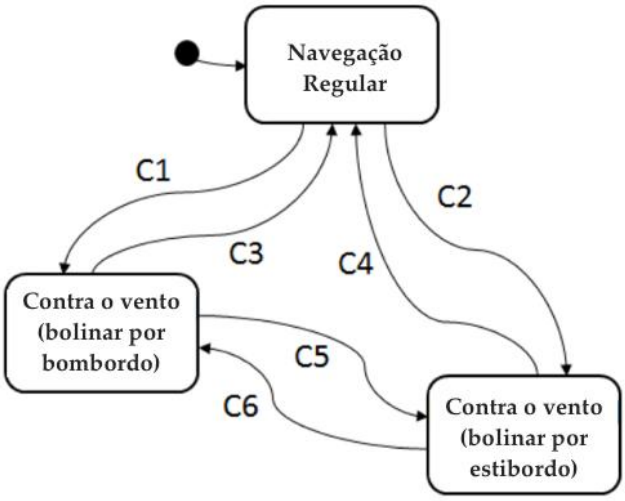

Figura 2.9 - Diagrama de estados que defino o modo de navegação [13]

Tabela 2.2 - Situações de navegação

C1	Entrando em situação de bolina (bolinar por bombordo) e o veleiro está nas zonas 1, 3 ou 5
C2	Entrando em situação de bolina (bolinar por estibordo) e o veleiro está nas zonas 2, 4 ou 6
C3	Movendo-se para navegação com vento favorável
C4	Movendo-se para navegação com vento favorável
C5	Mudança de bordo em situação de bolina e o veleiro está nas zonas 3 ou 5
C6	Mudança de bordo em situação de bolina e o veleiro está nas zonas 4 ou 6

Ao entrar em qualquer um dos três estados referidos, o novo ponto inicial e o ponto de destino (final) para a direção da navegação serão definidos de acordo com o seguinte:

• para o estado de navegação regular, o ponto inicial e o ponto final serão respetivamente A e B, como inicialmente definido na Figura 2.8 para o segmento;

• para o estado de navegação contra o vento em navegação por bombordo bem como para o mesmo em navegação por estibordo, o ponto inicial da direção

de navegação é ajustado para a posição atual do veleiro, enquanto o ponto de destino é configurado para um ponto pertencente a uma linha resultante de virar por davante o veleiro (fora da área de navegação), usando uma série de segmentos fechados para superar um curso em direção ao vento.

2.2.3.2 Controlador difuso de baixo nível

Segundo o artigo [13], as técnicas de controle difuso foram selecionadas para a implementação do controlador de baixo nível tanto para o leme como para a vela [14]. A motivação principal está associada ao fato de que seria mais fácil descrever dependências comportamentais usando um conjunto de regras. Um diagrama de blocos para o controlador de baixo nível é apresentado na Figura 2.10.

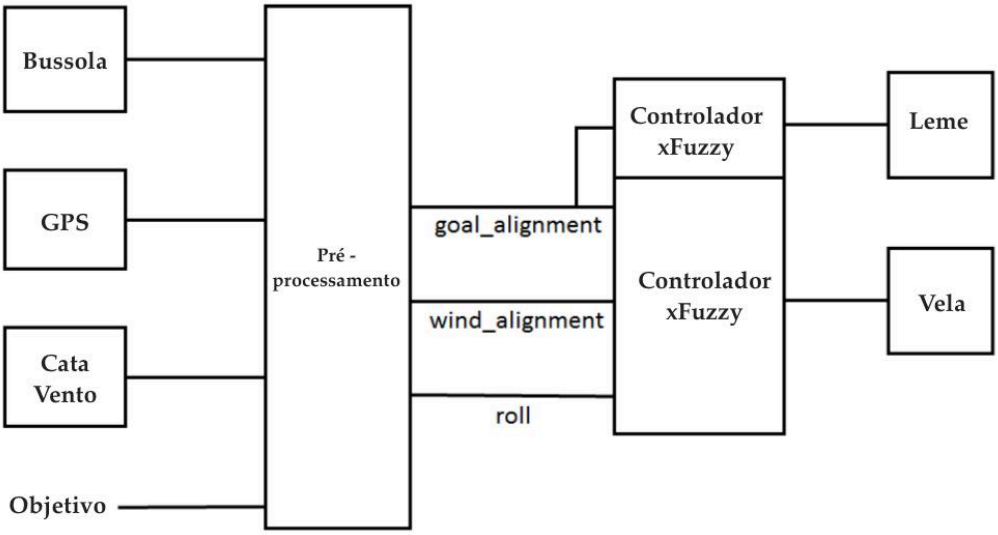

Figura 2.10 - Diagrama de blocos do controlador de baixo nível [13]

O bloco de pré-processamento é responsável por obter dados da bússola (orientação atual do veleiro, bem como atitude da inclinação (roll)), GPS (localização atual) e cata vento (direção do vento em relação à orientação do veleiro). As variáveis selecionadas para serem entradas dos controladores difusos são apenas três:

• *goal_alignment* (obtido pela diferença entre a orientação do veleiro e a direção desejada em direção ao objetivo); esta variável é usada por ambos os controladores de leme e vela;

• *wind_alignment* (obtido pela diferença entre a direção do vento e a orientação do veleiro); esta variável é usada apenas pelo controlador de vela;

• *roll* (obtido a partir do módulo da bússola, fornece uma informação indireta sobre a velocidade do vento); esta variável é usada apenas pelo controlador de vela.

No controlador do leme, a variável *goal_alignment* é a única entrada, caracterizada como ilustrado na Figura 2.11, um conjunto de regras difusas são definidas e apresentadas na Figura 2.12, onde a variável de saída é caracterizada usando cinco possíveis valores.

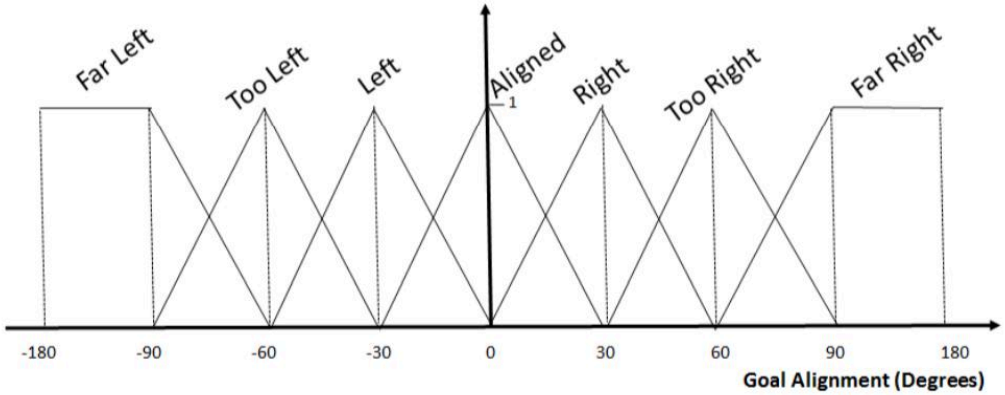

Figura 2.11 – Caracterização difusa da variável *gool_alignment* [13]

IF goal_alignment IS	THEN rudder IS
far_left	full_port
too_left	port
left	centered
aligned	centered
right	centered
too_right	starboard
far_right	full_starboard

Figura 2.12 – Regras difusas associadas ao controlador do leme [13]

Para o controlador da vela, seguiu-se uma abordagem semelhante, considerando três variáveis de entrada, nomeadamente o *goal_alignment* (mantendo os sete valores difusos apresentados na Figura 2.11), o *wind_alignment* (mantendo

três valores difusos: popa, inclinação e arco) e roll (mantendo dois valores difusos: *right* e *heel*), enquanto a variável difusa de saída mantém cinco possíveis valores (totalmente folgado, folgado, meio, caçado, totalmente caçado). O controlador é composto por um conjunto de 42 regras difusas. Uma descrição de uma versão anterior do controlador (apesar de satisfazer a mesma estrutura) pode ser encontrada em [10].

2.3 Sensores Anticolisão

Os sensores de anticolisão, como o próprio nome indica, têm a função de detetar objetos que representem perigo de colisão com os veículos onde estão incorporados, permitindo ao condutor ou ao sistema de navegação autónoma agir de modo a evitar a colisão.

Estes sensores são muito utilizados nos sistemas de estacionamento incorporados na maioria dos veículos automóveis mais modernos e numa grande parte dos veiculos autónomos utilizados na industria.

São normalmente instalados na dianteira ou traseira dos veículos.

Todos os sensores abordados em seguida, exceto o ótico e o GPS, funcionam com o principio básico de radar ou sonar.

Ou seja, o sistema possui um transmissor que emite uma onda e parte da sua energia será refletida e detetada pelo recetor caso exista uma superfície refletora ao seu alcance.

Os radares são ótimos para calcular a velocidade e a direção de um ou vários objetos em relação ao próprio radar.

2.3.1 Ultrassons

Este tipo de sensor utiliza ondas de pressão ultrassónicas para o calculo da distância do mesmo a um ou vários objetos. A sua estrutura incorpora um emissor e um recetor de ultrassom. O emissor, emite uma onda ultrassónica com uma frequência maior que 20 kHz (frequência máxima que um ser humano consegue ouvir) e parte da energia dessa onda, caso exista um objeto ao seu alcance é refletida e detetada pelo recetor. O tempo que decorre desde a emissão dessa onda até à sua receção é contabilizado e a distância ao objeto é facilmente calculada através da divisão desse tempo pela velocidade aproximada do som (340 m/s) [15] multiplicada por dois, visto que se tem de ter em conta que o sinal percorre o mesmo caminho duas vezes (ida e volta).

$$Distância = \frac{Tempo}{(Velocidade\ do\ som\ no\ ar) * 2} \quad (1)$$

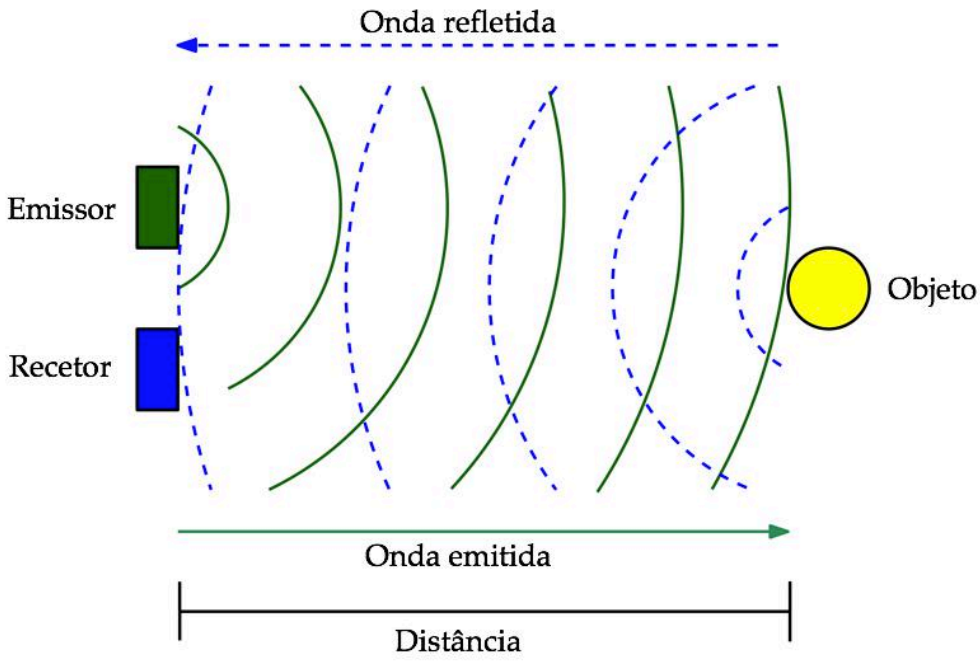

Figura 2.13 – Funcionamento de um sensor de ultrassons

Estes sensores têm um ângulo de visão relativamente razoável. Um exemplo disso é o sensor ultrassónico à prova de água modelo IM120712022 que deteta bem objetos que estejam dentro de um ângulo α de 75 +- 10 graus [16].

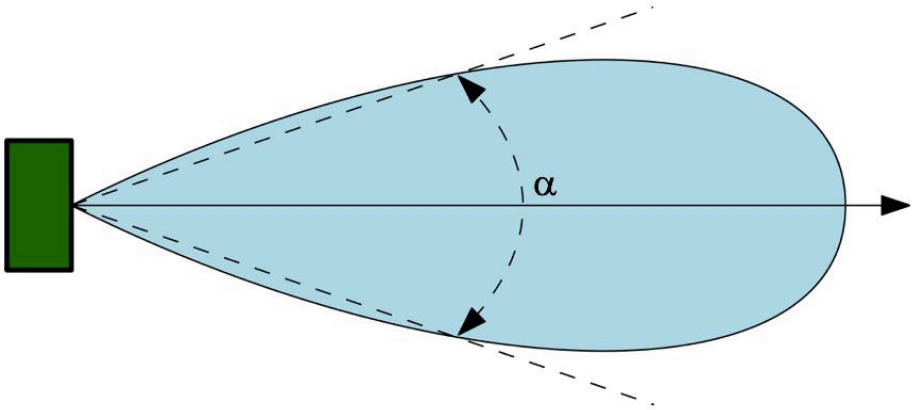

Figura 2.14 - Diretividade do sensor IM120712022

Este sensor é adequado para ser utilizado em objetos com baixas velocidades e curtas distâncias (aproximadamente até 5 metros). É uma má opção para distâncias longas e para grandes velocidades devido à sua sensibilidade ao vento (visto que a onda se propaga através do ar) e ao efeito Doppler [17].

2.3.2 Laser

Os sensores lasers são normalmente direcionados de acordo com o caminho do veiculo. Esta é uma tecnologia que funciona com objetos a altas velocidades e que evita o ruido de radio frequência. Um dos problemas são as leituras imprecisas caso o grau de reflexibilidade do obstáculo seja baixo.

Um sensor laser muito interessante é o LIDAR-Lite [18] da Pulsed Light. Este é um sensor de distância a laser de alto desempenho e o seu preço é muito competitivo comparado com sensores com um desempenho semelhante, custando cerca de 150 euros [19]. O seu alcance é de 40 metros, é pequeno, leve e gasta pouca energia. Este sensor é facilmente ligado ao Arduino, tendo mesmo uma biblioteca dedicada ao mesmo [20].

Figura 2.15 - Sensor LIDAR [19]

Uma outra opção dentro destes sensores seria modificar um sensor laser de mão utilizado normalmente pelos engenheiros, como por exemplo o UT390B da Uni-T [21] ilustrado na Figura 2.16 que custa cerca de 50 euros de forma a que possa ser ligado com o Arduino [22], obtendo desta forma um sensor capaz de medir distâncias até 40 metros com uma precisão de 2 mm.

Figura 2.16 - UT390B da Uni-T [21]

Outra forma de usar lasers para medir distâncias seria em conjunto com uma câmara fotográfica, ou seja, poderia ser colado à câmara na direção de captura de imagem um ou vários lasers muito simples e baratos. Quando esses lasers incidem num objeto o que a câmara capta são círculos de vários tamanhos dependendo da distância a que está de cada um dos objetos. Teria de se ter ainda

uma base de dados no sistema, de modo a receber o tamanho dos círculos captados pela imagem e a compara-los com essa mesma base, fazendo corresponder a cada tamanho uma distância equivalente.

Para o uso desta técnica teria ainda de ser usado um controlador com capacidade para processar imagens, tornando-a assim um pouco mais complexa comparada com um simples sensor laser medidor de distâncias.

2.3.3 Sensores LED

Nestes sensores são usados emissores de infravermelhos LED com comprimentos de onda na gama dos 880 nm. São principalmente usados a nível industrial, mas também integram alguns sistemas de estacionamento.

São usados para distâncias curtas (< 3 metros) e têm um mau funcionamento a altas temperaturas.

2.3.4 AIS

O AIS (Automatic Identification System) [23], como o próprio nome indica é um sistema de identificação automática utilizando ondas radio de muito alta frequência que fornece informações rápidas e precisas sobre própria embarcação (tais como a sua identificação, posição, curso e velocidade) a outras embarcações e às autoridades costeiras de forma a diminuir o risco de colisões.

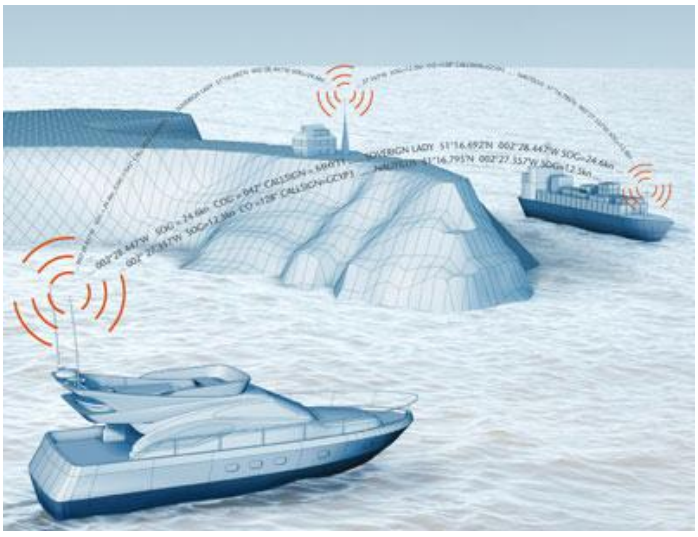

Figura 2.17 - Ilustração do AIS [23]

Este sistema pode manipular mais de 2000 relatórios por minuto e consegue atualizar a informação a cada dois segundos [24]. Existem duas frequências dedicadas a este sistema e cada uma delas é dividida em 2250 intervalos de tempo que são repetidos a cada 60 segundos.

O sistema AIS é constituído em parte por um transcetor (sistema emissor/recetor) de muito alta frequência (VHF), como por exemplo um LORAN-C [25] ou por um recetor GPS. Possui ainda outros sensores, tais como indicadores de velocidade, bússola giroscópica e indicador de velocidade de rotação e de direção.

2.3.5 Imagem

Estes sensores consistem em câmaras e num sistema capaz de processar a imagem recebida. Na embarcação relacionada com este projeto um grande candidato a ser integrado será o sistema ótico denominado de Pixy. A Pixy possui um processamento de imagem integrado. É possível ensinar a Pixy a identificar centenas de objetos em simultâneo. Esta câmara é ideal para navegação diurna e má para navegação noturna uma vez que funciona mal com pouca quantidade de luz. Um outro ponto a ter em consideração na aplicação da Pixy na embarcação é a ondulação, pois com forte ondulação a imagem pode tornar-se instável e de difícil processamento.

Figura 2.18 - Câmara Pixy [26]

A Pixy assim que sai da caixa está preparada para se conectar com o Arduino e enviar-lhe blocos de informação a 1 Mbits/seg, isto significa que consegue enviar mais de 6000 objetos detetados por segundo ou 135 objetos por frame, visto que tem uma capacidade de processamento de 50 frames por segundo.

Uma outra alternativa seria ligar uma simples câmara de ação, estilo GoPro, mas de baixo custo. Essa câmara poderia ser ligada por exemplo a um Raspberry Pi onde seria feito o processamento de imagem, visto que o Arduino não tem capacidade para o fazer.

2.3.6 Radar de Comprimento de Onda Milimétrico

A maioria dos carros modernos usam este tipo de sensores. Estes radares funcionam com frequências de onda muito altas (30 a 300 GHz, o que corresponde a um comprimento de onda entre 1 e 10 mm [27]) e a distância a um objeto é calculada através do tempo que a onda demora a incidir e a ser refletida pelo mesmo. Esta tecnologia consegue detetar objetos até 40 metros [28], funciona quase à velocidade da luz e têm energia suficiente para evitar o ruido de radio frequência [29].

2.4 Plataformas

O veleiro relacionado com este trabalho utiliza a plataforma Arduino, mas pode ser ou não vantajoso alterar para uma plataforma com mais capacidade de processamento quando o sistema anticolisão desenvolvido for integrado no veleiro em trabalhos futuros. Se for o caso, uma outra plataforma considerada poderá ser o Raspberry Pi.

2.4.1 Plataforma Arduino Mega 2560

Este projeto é desenvolvido com o apoio de uma placa Arduino mega 2560.

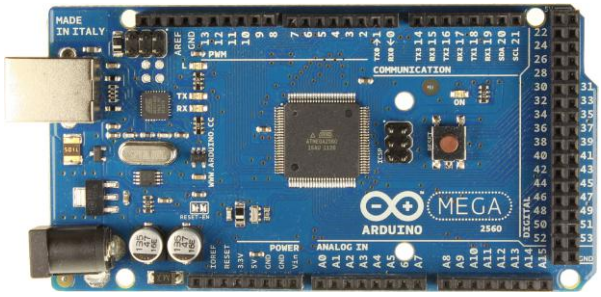

Figura 2.19 -Arduino Mega 2560 [30]

Esta é uma placa do Arduino baseada no microcontrolador ATmega2560. Esta placa contem 54 pinos de entradas e saídas digitais onde 15 destas podem ser usadas como saídas PWM. Contem ainda 16 entradas analógicas, algumas portas serie e portos digitais. Esta placa possui tudo o que é necessário para controlar o microcontrolador e é facilmente controlada pelo computador através de um cabo USB e do software do Arduino [31].

Tabela 2.3: Especificações do Arduino mega 2560 [31]

Microcontrolador	ATmega2560
Tensão operacional	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limite)	6-20V
Digital I/O pins	54 (dos quais 15 fornecem saída PWM)
Pinos de entrada analógicos	16
Corrente DC por Pin I/O	20 mA
Corrente DC para 3.3V Pin	50 mA
Memória Flash	256 KB dos quais 8 KB são usados pelo bootloader
Velocidade do relógio	16 MHz

2.4.2 Plataforma Raspberry Pi

O Raspberry Pi [32] é um computador básico de baixo custo que foi originalmente criado para promover o interesse dos estudantes do ensino secundário [33]. Mas devido ao seu tamanho e preço acessível foi rapidamente abraçado por fabricas e por entusiastas de projetos eletrónicos que requeressem um poder de computação superior a um microcontrolador básico como o Arduino.

Esta plataforma funciona com o sistema Linux, é mais lenta que um computador normal e tem um baixo consumo de energia. Como é capaz de correr um sistema operativo tem um mundo de funções muito maior que o Arduino.

Figura 2.20 - Raspberry Pi 3 [34]

Na Tabela 2.4 pretende-se mostrar a capacidade desta plataforma enumerando algumas das suas características.

Tabela 2.4 - Características do Raspberry Pi 3 [34]

CPU	Cortex A53 Quad Core
Velocidade do processador	1.2 GHz
RAM	1 GB
Portas	4 Usb, HDMI, Ethernet, interface serial da câmara (CSI), interface de exibição (DSI), conector de áudio e vídeo analógico
Pinos GPIO	40

CAPÍTULO 3

SISTEMA DESENVOLVIDO

3.1 Escolha do Hardware a considerar

Para a escolha dos sensores a considerar foram utilizados vários critérios. Os critérios considerados mais relevantes estão detalhados na Tabela 3.1.

Um dos critérios utilizados foi "Deteção de objetos". Este critério diferencia o sistema AIS dos restantes sensores, uma vez que o AIS só funciona para embarcações que também o possuam enquanto que os outros sensores conseguem detetar objetos nas suas proximidades.

Outro critério tido em conta foi o grau de reflexibilidade de cada objeto uma vez que a robustez dos sensores que têm como base de funcionamento a deteção de objetos através da emissão e receção de ondas (ondas essas que incidem nos objetos, são refletidas pelos mesmos e recebidas pelos sensores) é influenciada pela mesma.

Foi também feita uma comparação das distâncias de funcionamento de cada sensor e dos seus ângulos de deteção.

Tabela 3.1 - Análise de vários critérios dos sensores anticolisão

Sensor	Deteção de objetos	Depende do grau de reflexibilidade do objeto	Distâncias	Ângulo de deteção
Ultrassons (de pequenas dimensões)	Sim	Sim	Curtas	Médio
Laser	Sim	Sim	Médias	Mau
LED	Sim	Sim	Curtas	Mau
AIS	Não	Não	Longas	-
Imagem	Sim	Não	Longas	Bom

Através da análise da Tabela 3.1, na escolha dos sensores a considerar para o sistema em desenvolvimento, chega-se à conclusão que o sistema AIS está de fora, uma vez que não deteta qualquer objeto, apenas outras embarcações que também utilizem AIS. Os sensores Laser e LED também ficam de foram uma vez que têm um mau ângulo de visão, dependem do grau de reflexibilidade do objeto são suscetíveis a ruido ou erros e no caso do LED só servem para curtas distâncias.

O sensor de imagem é o mais adequado, uma vez que não depende do grau de reflexão dos obstáculos, funciona até longas distâncias e tem um ângulo de visão bom. Foi então por isso escolhido o sensor Pixy descrito anteriormente.

Um sensor de ultrassons ainda foi ponderado para deteção de obstáculos a curtas distâncias, servindo assim como um sistema de emergência caso o sistema baseado no sensor de imagem falhasse. No entanto essa ideia foi abandonada após alguns testes com o modelo IM120712022 e com o modelo IM120628006 uma vez que esses testes demonstraram a grande suscetibilidade não só ao grau de reflexibilidade dos obstáculos como também ao ruido causado pelo ambiente em redor. Os testes também demonstraram que os sensores só funcionavam como se pretendia quando os obstáculos estavam quase encostados ao emissor, sendo que quando se afastavam o sinal recebido começava a ser extremamente fraco, não servindo assim para uma utilização fiável.

A plataforma aconselhada para a implementação do sistema desenvolvido neste projeto em trabalhos futuros é o Arduino uma vez que não é necessária mais capacidade de processamento e que todos os algoritmos efetuados em teses anteriores que tornam o veleiro autónomo podem ser utilizados sem qualquer alteração.

3.2 Algoritmos de Perigo

Neste ponto irão ser abordadas algumas situações em função das condições ambientais. Estas situações, presentes na Tabela 3.2, têm em conta a inclinação do veleiro, a força do vento, a corrente e a ondulação. A situação 3.2.1 é considerada a situação ideal, onde existe vento, mas fraco (não provocando assim inclinação do veleiro). Nesta situação também não é considerada a existência de corrente nem de ondulação. Foi por essas razões que o algoritmo base foi desenvolvido nesta situação. Todas as outras situações têm o mesmo algoritmo integrado, mas com o acréscimo de algoritmos extra.

Na situação 3.2.2 resolve-se o problema da receção de imagem quando o veleiro está inclinado. Na situação 3.2.3 resolve-se o problema de quando existe corrente, ou seja, quando a corrente influencia a direção do veleiro, fazendo com que este não navegue apenas com um vetor de velocidade na direção para a qual está apontado, mas sim, com a soma desse vetor com um vetor de velocidade a apontar para um dos lados, ou seja, quando o ângulo entre o sentido para o qual a Pixy está apontada e o sentido em que o veleiro está a navegar é diferente de zero graus. Na situação 3.2.4 resolve-se o problema da receção de imagem quando existe ondulação. E por fim, na situação 3.2.5 quando a Pixy visualiza vários obstáculos é só considerado o que está mais próximo.

Tabela 3.2 - Conjunto de situações abordadas em função das condições ambientais.

Situação	Inclinação	Vento	Corrente	Ondulação	Obstáculos
3.2.1 Simplificada	Inexistente	Fraco	Inexistente	Inexistente	1
3.2.2 Com vento moderado	Moderada	Mode-rado	Inexistente	Inexistente	1
3.2.3 Com corrente	-	-	Existe	Inexistente	1
3.2.4 Com ondulação	-	-	-	Moderada	1
3.2.5 Com múltiplos obstáculos	-	-	-	-	Vários

No Arduino, após a receção dos dados da Pixy, as correções desses dados serão efetuadas tendo em conta as situações da Tabela 3.2 com a seguinte ordem:

1º - 3.2.4 – É tida em conta a ondulação.

2º - 3.2.5 – É feita a verificação do número de obstáculos que existem e é escolhido o mais próximo.

3º - 3.2.3 – É feita a correção da zona de perigo considerada.

4º - 3.2.2 – É feita a rotação da imagem caso o veleiro esteja inclinado.

5º - 3.2.1 – Por último e depois de corrigir as situações anteriores, é feita a análise da existência ou não de perigo e do lado em que se encontra o obstáculo caso exista perigo.

3.2.1 Situação simplificada

Como referido anteriormente, inicialmente para construir o algoritmo base deste sistema ótico de anticolisão não foram consideradas algumas condições de navegação como por exemplo a inclinação do veleiro, a forte ondulação, a direção do veleiro não coincidir com o sentido da navegação (sem corrente) e a possibilidade de aparecer mais do que um obstáculo de risco.

Então para este algoritmo base foi só considerado que iria aparecer um e um só obstáculo de risco, que o barco navega estável, sem inclinação e no sentido para o qual está orientado.

O algoritmo é composto por vários passos. Um dos primeiros passos é a análise da imagem que vem da Pixy e verificação da existência ou não de um obstáculo de risco (boias a vermelho/laranja/amarelo). Para este passo foi criada uma função que recebe da Pixy três parâmetros, tais como o centro geométrico do objeto (em relação ao referencial interno da Pixy e que está ilustrado na Figura 3.1), a largura e a altura do mesmo. Esta função retorna dois parâmetros, onde um deles é a existência de perigo ou não, e o outro, o lado em que o obstáculo se encontra em relação ao centro da imagem.

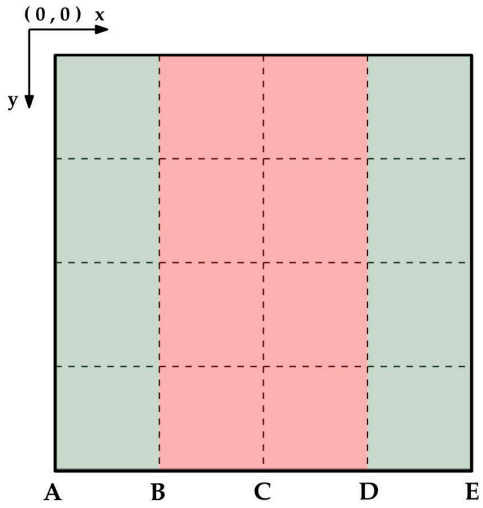

Figura 3.1 - Plano de imagem proveniente da Pixy.

Sempre que é detetado um obstáculo na imagem é enviado da Pixy para o Arduíno a informação desse mesmo obstáculo. Através da largura e da altura do obstáculo na imagem é calculada a sua área na mesma. De notar que esta área não é medida em nenhuma unidade real, sendo assim só uma indicação da área ocupada pelo obstáculo na imagem.

É considerado perigo de colisão sempre que existe um obstáculo com uma área superior à AREA_MAXIMA e que se situe entre as linhas B e D (Figura 3.1).

Como foi mencionado anteriormente é feita uma análise do lado em que se encontra o obstáculo. Caso o centro do obstáculo seja igual ao centro do referencial C, o obstáculo encontra-se centrado, caso o centro do obstáculo esteja à esquerda do referencial C, encontra-se no lado esquerdo e caso esteja à direita desse referencial, encontra-se no lado direito. Estas comparações são feitas com o algoritmo representado na Figura 3.2.

```
if (x < C){
      lado = 0; }          // lado esquerdo
if (x >= C){
      lado = 1; }          // lado direito
```

Figura 3.2 - Código do Arduíno que devolve o lado do obstáculo.

Caso a área do obstáculo seja inferior a AREA_MAXIMA, não existe perigo, caso contrário terá de se recorrer a mais alguns passos.

Sabendo o lado em que o obstáculo se encontra, sabendo também o seu centro geométrico e a sua largura, foram calculadas as coordenadas das suas extremidades visto que se pretende manter fora da zona de perigo todas as coordenadas do mesmo.

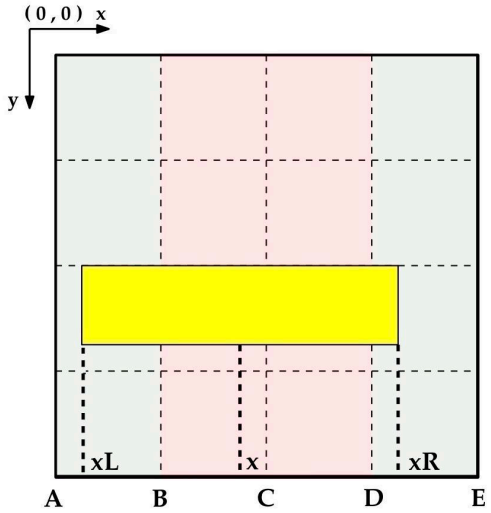

Figura 3.3 - Exemplo de obstáculo na tela da Pixy.

As coordenadas em x das extremidades do obstáculo são calculadas da seguinte forma:

$$\text{Extremidade direita,} \qquad \text{xR} = x + \frac{largura}{2} \tag{2}$$

$$\text{Extremidade esquerda,} \qquad \text{xL} = x - \frac{largura}{2} \tag{3}$$

Obtidas estas coordenadas resta verificar se alguma parte do obstáculo se encontra na zona de perigo. Para isso é feita uma verificação se a distância de cada uma das coordenadas em x dos pontos x, xR e xL (equações (4)**(5)(6)(7)**) ao ponto C é menor que a distância do ponto B ao ponto C. Desta forma consegue-se saber que o obstáculo se encontra na zona de perigo.

$$D1 = \text{Distância de x a C} \quad (4)$$

$$D2 = \text{Distância de xR a C} \quad (5)$$

$$D3 = \text{Distância de xL a C} \quad (6)$$

$$D4 = \text{Distância de B a C} \quad (7)$$

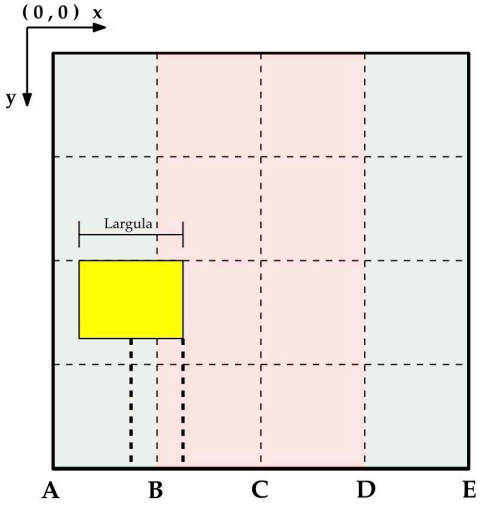

Figura 3.4 - Exemplo de outro obstáculo na tela da Pixy.

Na Figura 3.5 está escrito o código implementado no Arduino que verifica a existência de perigo utilizando a lógica anteriormente descrita.

```
if (area > AREA_MAXIMA){
   if (D1 ≤ D4 || D2 ≤ D4 || D3 ≤ D4){
                perigo = 1;}              // Há perigo
   else perigo = 0; } // Não há perigo
```

Figura 3.5 - Código do Arduino que verifica a existência de perigo.

3.2.2 Situação com vento Moderado

É considerado que o veleiro se encontra sem inclinação como se observa na Figura 3.6, quando o ângulo θ é de 90º (ângulo obtido através da bussola). Na Figura 3.7 o veleiro encontra-se inclinado θ graus para a esquerda (bombordo) e na Figura 3.8, θ graus para a direita (estibordo). O eixo do horizonte está representado com uma linha ondulada ou uma reta a azul claro nas figuras 3.6 a 3.8.

Figura 3.6 - Ângulo θ quando o veleiro não está inclinado

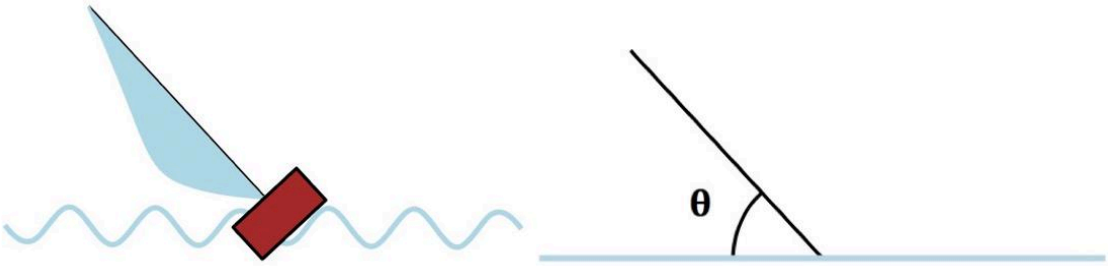

Figura 3.7 - Ângulo θ quando o veleiro está inclinado para a esquerda

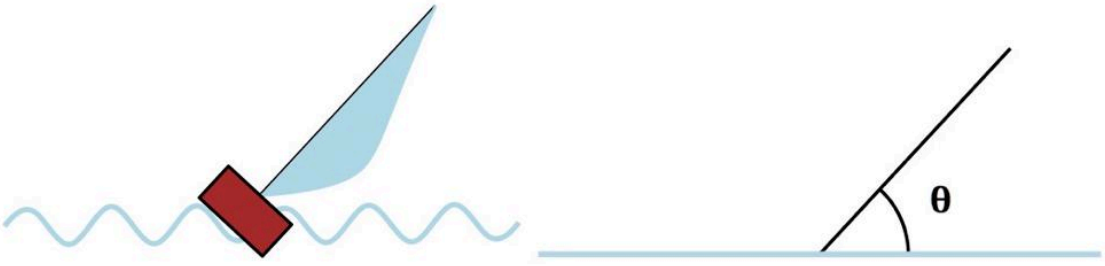

Figura 3.8 - Ângulo θ quando o veleiro está inclinado para a direita

Como se pode observar pela Figura 3.9, quando o veleiro se encontra inclinado (situação (a)), a imagem recebida pela Pixy (situação (b)), é também referente a uma imagem com horizonte inclinado, ou seja, o eixo do horizonte na imagem não faz um ângulo de zero graus como acontece na realidade. Por isso, é necessário fazer a rotação da imagem (situação (c)), para que o eixo do horizonte faça o ângulo de zero graus, tal como acontece na realidade (situação (a)).

A rotação é proporcional ao ângulo θ feito entre a linha do horizonte e o mastro do veleiro.

Quando o veleiro está inclinado θ graus para a esquerda (situação (a)), a linha do horizonte na imagem obtida pela Pixy (situação (b)), está inclinada (90 - θ) graus para a direita. Então tem de se corrigir essa inclinação fazendo a rotação de todos os pontos utilizados em (90-θ) graus para a esquerda.

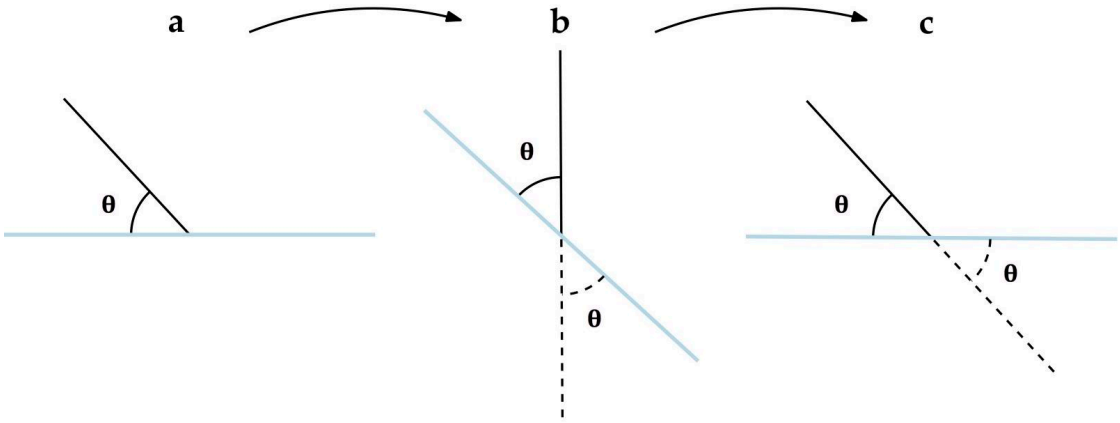

Figura 3.9 – (a) Horizonte real, (b) Horizonte visto pela Pixy, (c) Horizonte após rotação

Não é praticável fazer a rotação de imagem completa, uma vez que o processamento da imagem é feito automaticamente na Pixy.

No entanto, o Arduino recebe da Pixy as coordenadas do centro do obstáculo bem como o seu comprimento e largura. Então em vez de se fazer uma rotação de todos os pixéis da imagem, faz-se apenas a rotação do ponto que corresponde ao centro do obstáculo e aos vértices limite da imagem (permitindo assim atribuir novos valores aos pontos limite da Pixy, pontos A e E da Figura 3.4).

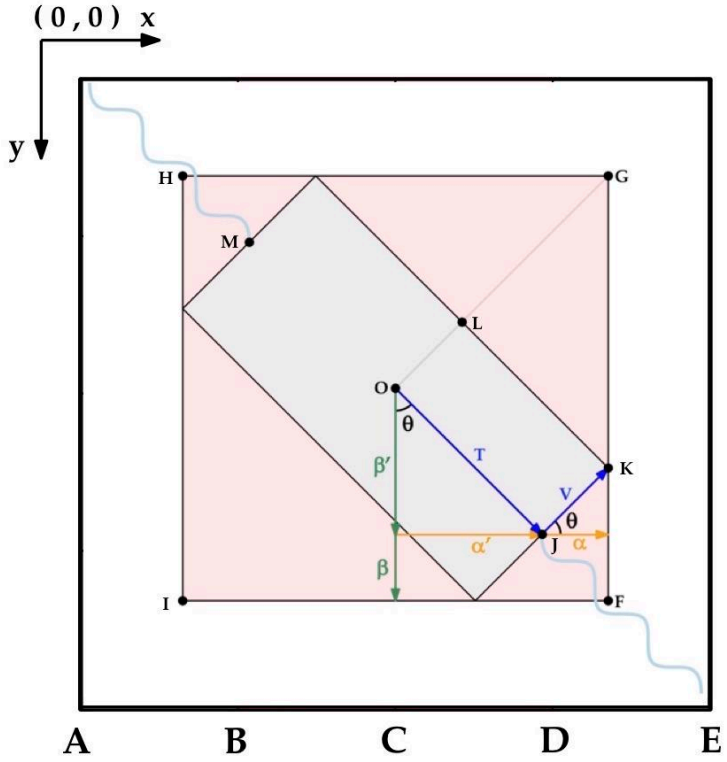

Figura 3.10 - Imagem recebida pela Pixy quando o veleiro está inclinado θ graus para a esquerda

Analisando mais em detalhe as situações (b) e (c) da Figura 3.9, em que a situação (a) dessa figura é referente à Figura 3.7, caso em que o veleiro se encontra inclinado θ graus para a esquerda. O retângulo a cinzento representa o obstáculo real e o retângulo a vermelho representa o obstáculo considerado pela Pixy.

Quando o veleiro está inclinado, a altura (β+β') e a largura (α+α') que a Pixy devolve corresponde ao retângulo a vermelho e por isso a altura e largura real do obstáculo (retângulo a cinzento) não são conhecidos. Então estes valores foram calculados para o pior caso, que é quando a altura do obstáculo, (segmento $\overline{OL}$) é tão pequena que o comprimento do mesmo, (segmento $\overline{JM}$) se aproxima do valor da diagonal do retângulo a vermelho, (segmento $\overline{FH}$).

Deste modo tem-se o valor máximo da largura do obstáculo como,

$$largura\_max = 2 * \sqrt{(\beta + \beta')^2 + (\alpha + \alpha')^2} \quad (8)$$

e é esse valor que o algoritmo utiliza sempre que o veleiro está inclinado, dando assim uma margem de segurança ainda mais confortável.

Como foi mencionado anteriormente, a situação (b) corresponde à imagem recebida pela Pixy quando o veleiro está inclinado para a esquerda θ graus (situação (a)). É necessário então proceder à rotação dos pontos utilizados em (90 - θ) graus para a esquerda (situação (c)).

A rotação do ponto central do obstáculo é feita da seguinte forma:

$$x'_O = x_O * \cos(90 - \theta) + y_O * sen(90 - \theta) \quad (9)$$

$$y'_O = -x_O * \text{sen}(90 - \theta) + y_O * cos(90 - \theta) \quad (10)$$

sendo x'_O e y'_O o novo ponto a ser considerado.

O mesmo procedimento é feito para os vértices da imagem obtida pela Pixy, uma vez que os valores dos pontos A e E da Figura 3.4 serão referentes às coordenadas em x desses mesmos vértices. Sendo o ponto A correspondente à coordenada x mais pequena entre todos os vértices, e o ponto E à coordenada maior. É pertinente realçar que quando o veleiro não está inclinado o ponto A tem o valor igual a zero.

Nesta fase já são conhecidos quase todos os pontos necessários, estando em falta apenas as coordenadas laterias do obstáculo. Então, para a coordenada x da parte lateral esquerda é feita uma subtração à coordenada x do ponto central do obstáculo pela *largura_max* anteriormente calculada e para a coordenada lateral direita é feita a soma dessa *largura_max* à coordenada x do ponto central do obstáculo.

Se o veleiro em vez de estar inclinado para a esquerda, estiver inclinado para a direita, terá de ser aplicada a rotação aos pontos utilizados da imagem, não para a esquerda, mas sim para a direita. O calculo da *largura_max* será feito da mesma forma, pois é independente do lado para o qual está inclinado, visto que o resultado é idêntico.

3.2.3 Situação com corrente

Como se pode observar na Figura 3.11, o veleiro da situação (1) só é influenciado pelo vetor força do vento, querendo isso dizer que nesta situação não existe corrente e por isso o sentido do vetor deslocamento $\vec{D}$ é igual ao sentido do vetor resultante do vento $\vec{V}$. De realçar que o sentido para o qual o veleiro está virado é igual ao sentido do vetor resultante do vento.

Na mesma imagem, mas na situação (2), existe corrente e o veleiro já sofre a incidência da força de dois vetores, sendo estes o do vento $\vec{V}$ e da corrente $\vec{C}$ e por isso o vetor deslocamento $\vec{D}$ é o vetor resultante da soma destes dois vetores anteriores. Então, nesta situação o sentido para o qual o veleiro está apontado já não coincide com o sentido de navegação.

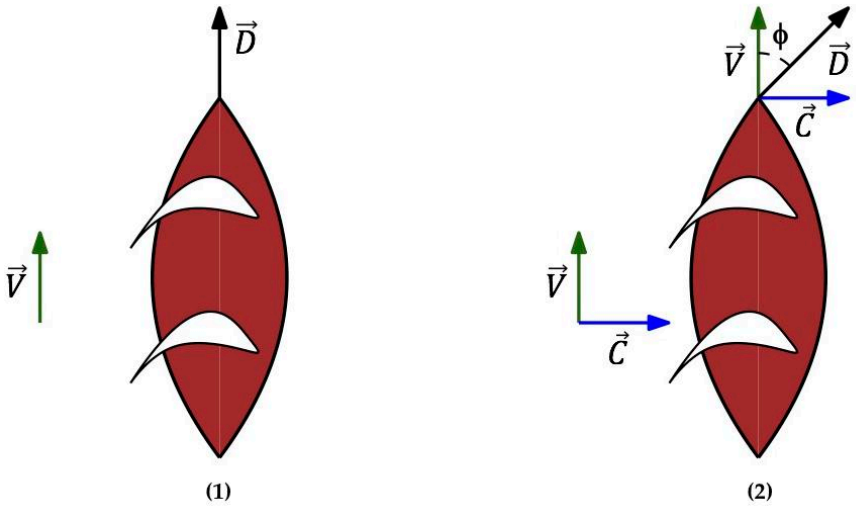

Figura 3.11 - Vetores força do vento e da corrente exercidos no veleiro

Deparou-se então com um problema na situação (2) da Figura 3.11, uma vez que a Pixy está apontada com sentido igual ao sentido para o qual o veleiro está apontado e deveria estar apontada no sentido do vetor deslocamento $\vec{D}$, ou seja, ϕ graus para a direita.

Para resolver este problema, tem de se deslocar a zona de perigo em ϕ graus para a direita, como está exemplificado na situação (2) da Figura 3.12.

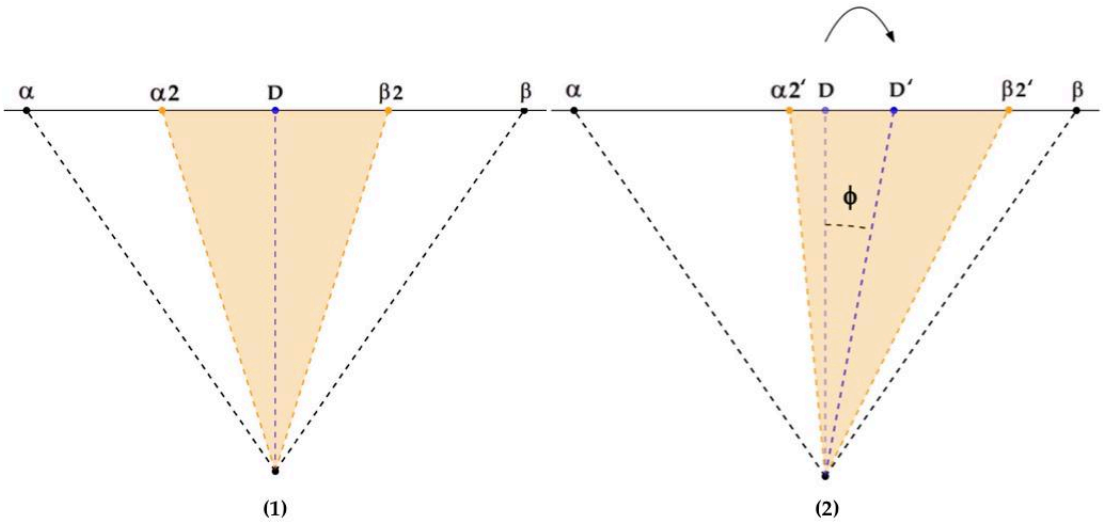

Figura 3.12 - Deslocamento da zona de perigo considerada em função do ângulo ϕ.

Em termos práticos o que será feito é o seguinte:

1º - Será calculado o ângulo ϕ através das coordenadas GPS da posição real do veleiro.

2º - Será feito o arrastamento das linhas limite da zona de perigo. Na pratica será feita uma equivalência entre esse ângulo ϕ e a distância a que as linhas serão arrastadas. Explicado pela Figura 3.1, o que acontecerá é que a área a vermelho será arrastada essa distância para a direita.

3.2.4 Situação com ondulação

Quando não existe ondulação, a posição do obstáculo é fixa na imagem, mas quando existe alguma ondulação, a sua posição vai variando, podendo até sair da imagem recebida pela Pixy.

No entanto esta situação não provoca nenhum problema uma vez que a função de perigo desenvolvida só entrará em ação quando algum objeto é detetado na imagem. Isto quer dizer que se existir um obstáculo e de repente o mesmo deixe de estar presente na imagem, nada irá acontecer e o algoritmo continua

com o último valor de perigo registado. Um exemplo é o caso em que um obstáculo se encontra na zona de perigo e o valor de perigo registado é perigo=1. Este valor só passará para perigo=0 quando o obstáculo estiver presente na imagem, mas fora da zona de perigo, querendo isto dizer que no caso de forte ondulação, em que o objeto pode sair da imagem, o valor de perigo continua a ser perigo=1. No instante seguinte o valor volta a estar presente na imagem e o algoritmo volta ao normal funcionamento.

3.2.5 Situação de Múltiplos obstáculos

Como se pode observar pela Figura 3.13, é possível que durante a competição apareçam vários obstáculos na imagem recebida pela Pixy.

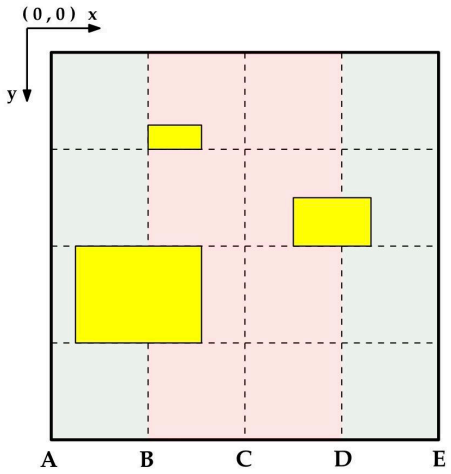

Figura 3.13 - Presença de vários obstáculos na imagem recebida pela Pixy

Mas considerando que todos os obstáculos são boias com aproximadamente o mesmo tamanho e que não estão colocadas muito próximo umas das outras, nesta situação, só é considerado o obstáculo com maior área ocupada na imagem.

3.3 Métodos desenvolvidos para o Simulador do comportamento do Veleiro

Nesta etapa foi desenvolvida a lógica do sistema anticolisão no programa Matlab.

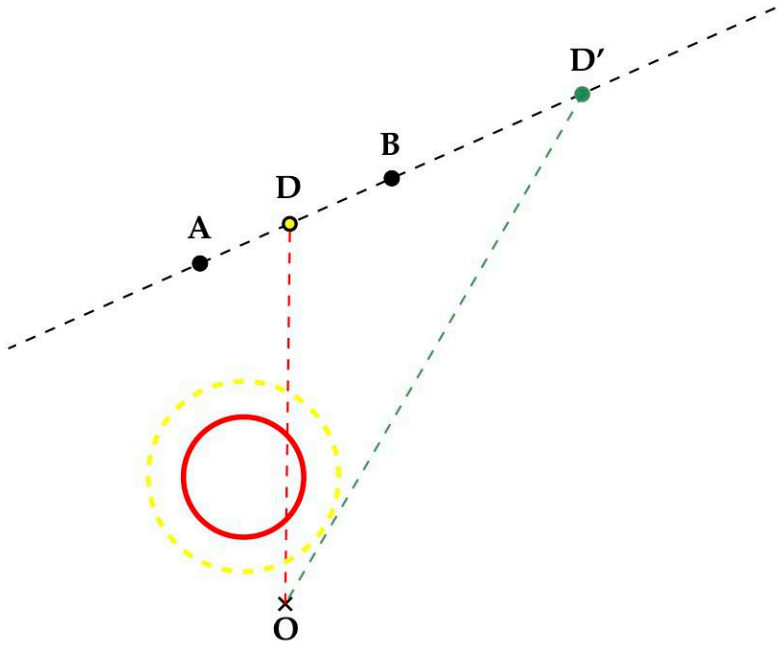

Figura 3.14 - Ilustração de uma situação em que o veleiro encontra um obstáculo.

Tendo em conta os pontos da meta A e B representados na Figura 3.14, foi calculado o ponto destino D, situado inicialmente no centro do segmento de reta $\overline{AB}$. Foi também calculada a equação da reta que passa por esses dois pontos.

O ponto O é referente à posição do veleiro. O ponto D' será o novo ponto de destino caso o veleiro se encontre em rota de colisão com um obstáculo (ou seja, se a informação de perigo proveniente da Pixy for perigo=1). Este ponto D' volta a ser o ponto de destino inicial D quando já não existir perigo.

De notar que se está a considerar que se conhecem os pontos A e B, pertencendo esses dois pontos à meta. Isto é valido nesta competição em especifico. Mas para uma situação mais geral, caso só seja conhecido o ponto D, os pontos A e B serão calculados para que o algoritmo funcione de igual forma. A forma como são calculados está explicada mais à frente e é referente à Figura 3.22.

Para simulação do algoritmo, como referido anteriormente, foi inserido um obstáculo cujas suas coordenadas são conhecidas ao contrário do que acontece na realidade. Este facto serve apenas para simulação dos dados provenientes da Pixy, e não vai alterar o funcionamento do algoritmo em relação a uma situação real.

Esta simulação em Matlab divide-se em duas partes (3.3.1 e 3.3.2). A primeira parte (3.3.1) é referente à simulação da Pixy e esta parte não será integrada no Arduino. A segunda parte (3.3.2) é referente à simulação do sistema anticolisão desenvolvido.

3.3.1 Funções para Simular a Pixy

Inicialmente para simular os dados provenientes da Pixy foi criada uma função mais simples tendo como base um único segmento de reta $\overline{OD}$. Esta função serviu apenas como base para o desenvolvimento do resto do algoritmo. Nesta parte deu-se especial importância ao algoritmo de anticolisão e por isso a estrutura da função que verifica se existe perigo de colisão não é crucial para o desenvolvimento do mesmo, apenas o seu output.

Depois de concluído o algoritmo de anticolisão foi então desenvolvida uma função mais complexa e mais real para simular a Pixy. Esta função está descrita na secção 3.3.1.2. De notar que a função mais simples funciona com base num segmento de reta $\overline{OD}$ e considera que não existe perigo quando esse segmento deixa de intercetar o obstáculo. Isso irá fazer com que o veleiro faça um percurso tangente ao obstáculo e por essa razão nesta função se o raio do obstáculo for r (circulo vermelho Figura 3.15) a função considera que r é igual r mais uma distância de segurança (circulo amarelo Figura 3.15) o que fará com que o veleiro faça um percurso tangente não ao obstáculo, mas sim ao circulo de segurança.

Os dados provenientes da Pixy serão se existe ou não perigo (perigo = 1 ou perigo = 0) e o lado em que se encontra o obstáculo caso exista perigo (lado = 0 ou lado = 1) lado esquerdo ou direito respetivamente.

3.3.1.1 Função mais simples para Simular a Pixy

Para a função mais simples que simula a Pixy foram utilizados alguns métodos.

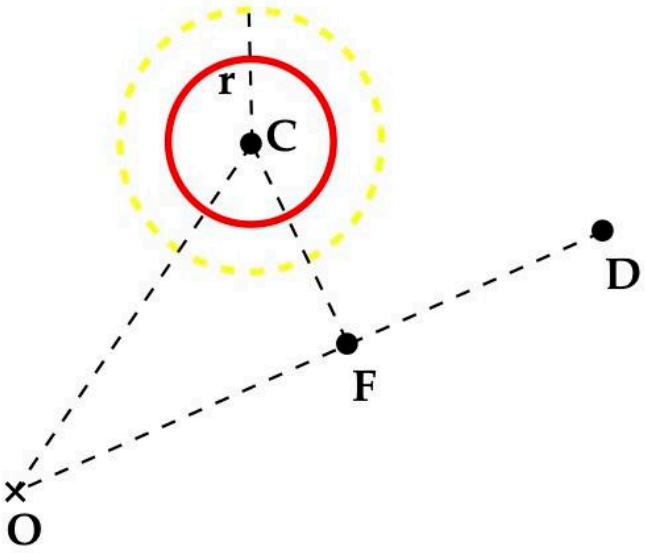

Figura 3.15 – Simulação da Pixy através de uma reta que intercepta ou não um obstáculo

O primeiro método foi para perceber se existe ou não perigo e para isso foi calculada a equação da reta que passa pelos pontos O e D.

Calculou-se o declive *mod* dessa reta,

$$mod = \frac{(yd - yo)}{(xd - xo)} \quad (11)$$

e com este declive calculou-se o parâmetro *bod*,

$$bod = yo - mod * xo \quad (12)$$

sendo a equação da reta a seguinte:

$$y = mod * x + bod \quad (13)$$

Posteriormente também foi calculada a equação da reta que passa pelos pontos C e F que tem como declive

$$mxc = -\frac{1}{mod} \quad (14)$$

significando assim que os segmentos de reta $\overline{OD}$ e $\overline{CF}$ são perpendiculares.

A equação desta reta será então a seguinte:

$$y = mxc * x + bxc \quad (15)$$

sendo que,

$$bxc = yc - mxc * xc \quad (16)$$

Depois foi calculada a coordenada do ponto F através da equação da interceção dessas duas retas.

$$y = \frac{\frac{bod}{mod^2} + bxc}{1 + \frac{1}{mod^2}} \quad (17)$$

$$x = \frac{y - bod}{mod} \quad (18)$$

Foi então calculada a distância entre o ponto C e o ponto F e comparada com a distância de segurança (raio r do circulo amarelo Figura 3.16).

$$d = \sqrt{(x - xc)^2 + (y - yc)^2} \quad (19)$$

Caso $d>r$ não há perigo, caso $d<r$ há perigo e caso d=r significa que a rota é tangente à circunferência de segurança de raio r e é considerado que há perigo.

O segundo método foi para perceber de que lado existe o perigo (direita ou esquerda).

Para isso foi analisado se existe interceção entre os dois segmentos

de reta $\overline{AC}$ e $\overline{OD}$ representados na Figura 3.16. Caso exista significa que o obstáculo se encontra à direita do segmento $\overline{OD}$, caso não exista significa que se encontra à esquerda do mesmo.

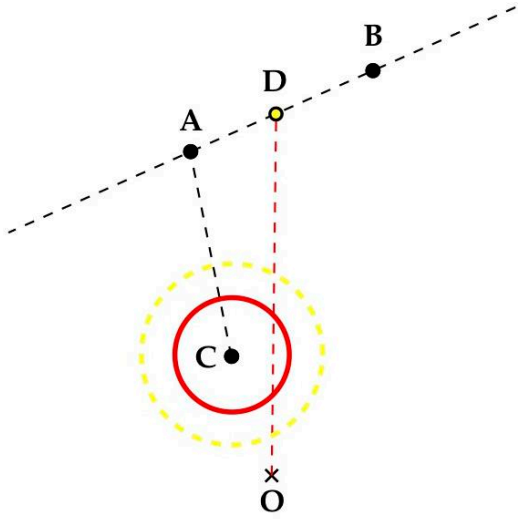

Figura 3.16 - Ilustração dos segmentos $\overline{AC}$ e $\overline{OD}$

$$det = (xd - xo) * (yc - ya) - (yd - yo) * (xc - xa) \quad (20)$$

Se *det* for 0 não há interceção.

$$ss = \frac{(xd - xo) * (yo - ya) - (yd - yo) * (xo - xa)}{det} \quad (21)$$

$$tt = \frac{(xc - xa) * (yo - ya) - (yc - ya) * (xo - xa)}{det} \quad (22)$$

Se *ss* e *tt* estiverem entre 0 e 1 o obstáculo encontra-se à direita, caso contrário encontra-se à esquerda. Estas formulas têm como base a teoria de vetores que podem ser visualizadas no documento [35].

3.3.1.2 Função mais complexa para Simular a Pixy

De forma a simular mais corretamente o comportamento da Pixy, foi criada uma simulação em Matlab que dependendo da posição do obstáculo devolve o mesmo output da função real implementada no Arduino.

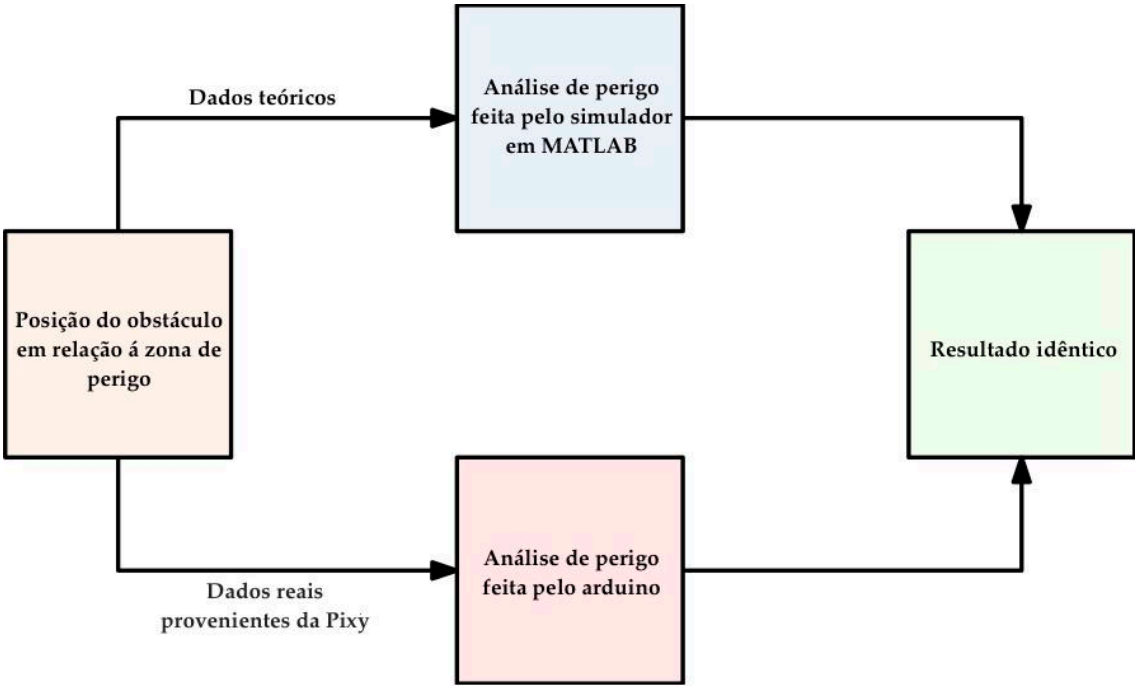

Figura 3.17 – Comparação entre a função perigo implementada no Arduino e a simulação em Matlab

Explicando por outras palavras e recorrendo ao esquema da Figura 3.17, a função que foi implementada no Arduino, (que recebe os dados depois da Pixy fazer o processamento da imagem e depois desses dados serem corrigidos tendo em conta as situações da Tabela 3.2), decide se existe perigo ou não através da análise da posição do obstáculo em relação à zona de perigo. Caso algum ponto do obstáculo esteja dentro da zona de perigo é considerado que existe perigo.

Esta simulação faz a mesma análise, e devolve o mesmo resultado.

Relativamente à Figura 3.18 o ângulo entre as retas tracejadas a preto representa o ângulo de visão da Pixy. O ângulo entre as retas tracejadas a laranja representa o ângulo de perigo, isto é, se o objeto estiver entre essas duas retas existe perigo de colisão. A reta tracejada a azul representa a direção da visão da Pixy e

as retas tracejadas a vermelho representam as tangentes do ponto O com o obstáculo bem como a reta que passa pelo ponto O e pelo ponto C (centro do obstáculo).

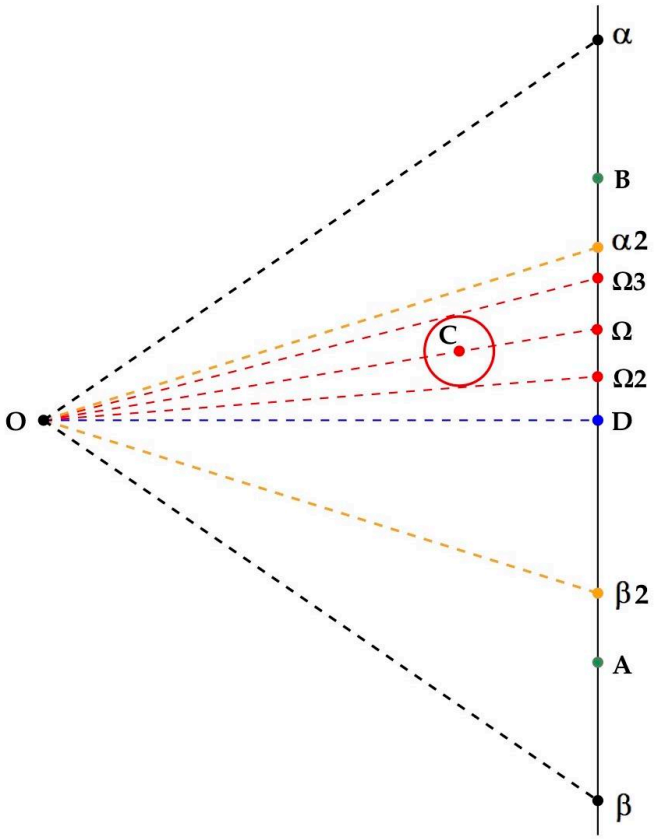

Figura 3.18 - Ilustração de todas as linhas necessárias aos cálculos nesta função.

Sendo o ângulo de visão da Pixy denominado horizontal_angle e igual a 75º, para a construção das linhas a preto utilizou-se o ângulo:

$$angulo\_visao = \frac{horizontal\_angle}{2} \quad (23)$$

e para as linhas a laranja o ângulo:

$$angulo\_visao2 = \frac{horizontal\_angle}{4} \quad (24)$$

No Matlab estes valores são parâmetros de entrada em funções que utilizam radianos e não graus, sendo por isso necessário multiplicar o valor dos ângulos

em graus por π e dividir por 180 passando-os desta forma para valores em radianos.

Considerou-se que o segmento de reta $\overline{OD}$ possui um declive igual a zero e, portanto, as outras retas foram construídas com base neste referencial.

Inicialmente foram calculadas as equações de todas as retas, sendo o declive das linhas a preto:

$$mo\_\alpha = \tan(angulo\_mod + angulo\_visao) \quad (25)$$

$$mo\_\beta = \tan(angulo\_mod - angulo\_visao) \quad (26)$$

e os seus pontos de interceção com o eixo dos "yy":

$$bo\_\alpha = \text{yo} - \text{mo\_}\alpha * \text{xo} \quad (27)$$

$$bo\_\beta = \text{yo} - \text{mo\_}\beta * \text{xo} \quad (28)$$

Os declives das linhas a laranja serão:

$$mo\_\alpha 2 = \tan(angulo\_mod + angulo\_visao2) \quad (29)$$

$$mo\_\beta 2 = \tan(angulo\_mod - angulo\_visao2) \quad (30)$$

e os seus pontos de interceção com o eixo dos "yy":

$$bo\_\alpha 2 = \text{yo} - \text{mo\_}\alpha 2 * \text{xo} \quad (31)$$

$$bo\_\beta 2 = \text{yo} - \text{mo\_}\beta 2 * \text{xo} \quad (32)$$

Já para a reta a azul o declive será:

$$moc = \frac{yc - yo}{xc - xo} \quad (33)$$

e o seu ponto de interceção com o eixo dos "yy":

$$boc = \text{yc} - \text{moc} * \text{xc} \quad (34)$$

As retas a vermelho Ω2 e Ω3 têm respetivamente um declive:

$$mo\_\Omega 2 = \tan\left(angulo\_moc + angulo\_\Omega\right) \quad (35)$$

$$mo\_\Omega 3 = \tan\left(angulo\_moc - angulo\_\Omega\right) \quad (36)$$

e os seus pontos de interceção com o eixo dos "yy":

$$bo\_\Omega 2 = \mathrm{yo} - \mathrm{mo\_}\Omega 2 * \mathrm{xo} \quad (37)$$

$$bo\_\Omega 3 = \mathrm{yo} - \mathrm{mo\_}\Omega 3 * \mathrm{xo} \quad (38)$$

sendo que,

$$angulo\_\Omega = \mathrm{asin}\left(\frac{r}{dist\_o\_c}\right) \quad (39)$$

$$angulo\_moc = \mathrm{atan}\left(moc\right) \quad (40)$$

$$dist\_o\_c = \sqrt{(xo - xc)^2 + (yo - yc)^2} \quad (41)$$

Posteriormente foram calculados os pontos de interceção de todas essas retas com a reta da meta. As equações utilizadas foram semelhantes para todos os pontos, por isso nas equações (42) e (43) onde está 'X', basta substituir por cada um dos pontos α, $\alpha 2$, β, $\beta 2$, Ω, Ω2 e Ω3.

Ponto $'X'$:

$$x\_'X' = \frac{bab - bo\_'X'}{mo\_'X' - mab} \quad (42)$$

$$y\_'X' = \mathrm{mo}'X' * \mathrm{x\_}'X' + \mathrm{bo\_}'X' \quad (43)$$

Sabendo a localização de todos estes pontos e de forma a tornar este simulador o mais real possível foi aplicada a lógica do ponto 3.2.1, mais precisamente a lógica da Figura 3.5 de forma a perceber se alguma parte do obstáculo se situa entre as retas a laranja, ou seja, para perceber se existe perigo de colisão. Foi então calculada a distância de cada um dos pontos Ω, Ω2, Ω3 e β2 ao ponto D e aplicado o algoritmo.

$$dist\_\Omega\_\mathrm{d} = \sqrt{(x\_\Omega - xd)^2 + (y\_\Omega - yd)^2} \quad (44)$$

$$dist\_\Omega2\_\mathrm{d} = \sqrt{(x\_\Omega2 - xd)^2 + (y\_\Omega2 - yd)^2} \quad (45)$$

$$dist\_\Omega3\_\mathrm{d} = \sqrt{(x\_\Omega3 - xd)^2 + (y\Omega3 - yd)^2} \quad (46)$$

$$dist\_\beta2\_d = \sqrt{(x\_\beta2 - xd)^2 + (y\_\beta2 - yd)^2} \quad (47)$$

$$\text{Extremidade esquerda,} \qquad \mathrm{xL} = x - \frac{largura}{2} \quad (48)$$

O obstáculo pode encontrar-se em varias posições, estando alguns exemplos nas figuras seguintes.

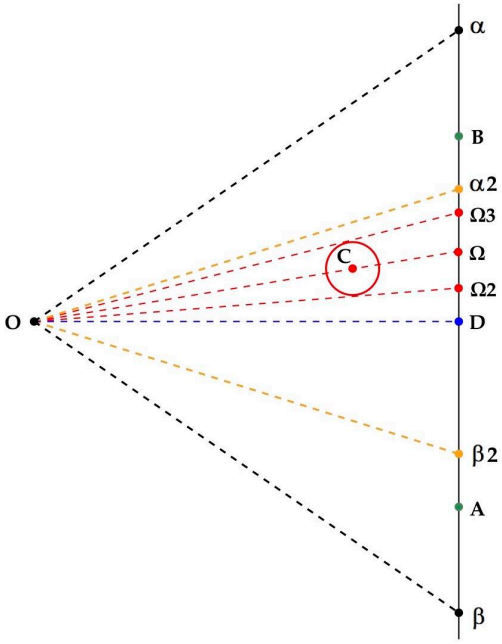

Figura 3.19 - Posição do obstáculo 1.

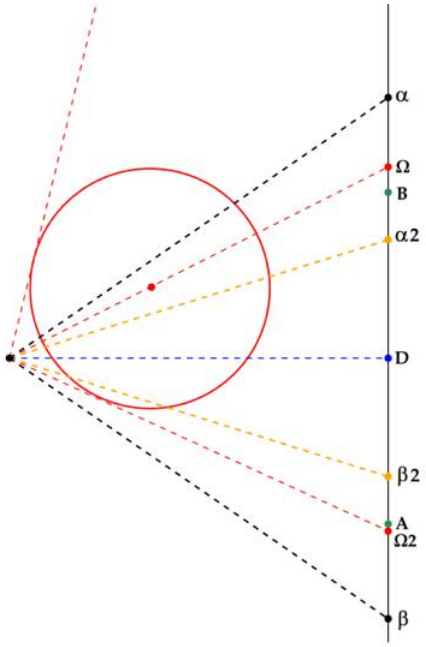

Figura 3.20 - Posição do obstáculo 2

Como se pode observar no exemplo da Figura 3.20 o obstáculo não está totalmente dentro do campo de visão do simulador da Pixy. Então para tornar o simulador o mais real possível, a extremidade esquerda do obstáculo (ponto Ω3) em vez de ficar fora do ângulo de visão fica no limite, isto é, neste caso, fica igual ao ponto α (que é o que acontece na realidade). A extremidade direita mantem-se igual, ou seja, no ponto Ω2, e o ponto Ω tem de ser corrigido, de forma a ficar

entre os pontos Ω2 e Ω3 e que o ângulo entre as 3 novas retas vermelhas seja igual. Para isso, através das novas equações das retas das extremidades, é calculada a reta bissetriz que intercepta a linha da meta e o novo ponto Ω é calculado através dessa interceção.

Um exemplo da explicação anterior pode ser observado na Figura 3.21, figura esta proveniente das simulações. Nesta figura existe uma reta a vermelho a mais, o que é justificado pelo facto dessa reta representar o limite da zona de perigo e, sendo a sua cor original o amarelo, na figura a reta está a vermelho pois foi definido que sempre que exista perigo numa determinada zona as retas ficam a vermelho.

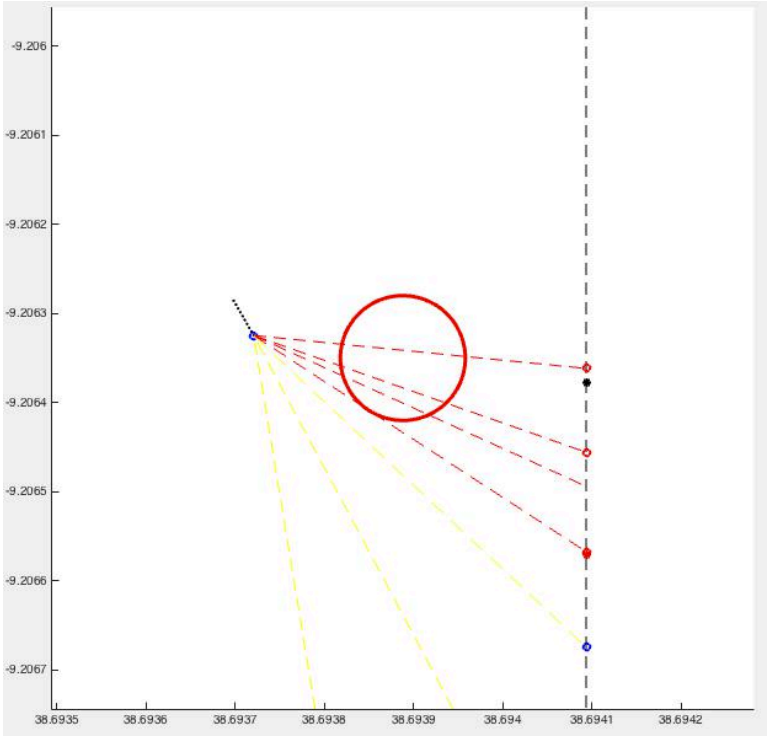

Figura 3.21 - Simulador da Pixy em Matlab

3.3.2 Função que simula o andamento do Veleiro

Para tornar o processo semelhante à realidade foi também criada uma função já mencionada anteriormente que tem como objetivo simular o andamento do veleiro. Este método faz com que o veleiro caso não exista perigo se desloque X metros a partir do ponto O ao longo do segmento $\overline{OD}$ ou $\overline{OD'}$ até chegar ao destino D ou D'. O valor do X é ajustável e variável conforme se deseje, mas tem de se ter em conta que se o valor for muito pequeno o tempo para o algoritmo correr será muito grande, por outro lado se X for muito grande não estará a simular corretamente a realidade pois significaria velocidades de navegação muito elevadas. As equações usadas neste processo são as seguintes:

$$xo' = \mathrm{X} * \mathrm{xd} + (1 - \mathrm{X}) * \mathrm{xo} \quad (49)$$

$$yo' = \mathrm{X} * \mathrm{yd} + (1 - \mathrm{X}) * \mathrm{yo} \quad (50)$$

3.3.3 Algoritmo Anticolisão

Com as funções que simulam a Pixy e o andamento do veleiro feitas, está-se então em condições para o desenvolvimento do algoritmo para evitar a colisão.

Voltando agora à Figura 3.14 caso exista perigo o objetivo será arrastar o ponto de destino D ao longo da reta da meta até que deixe de existir perigo. O novo ponto destino D' será então o ponto calculado no instante após em que o perigo deixou de existir.

Caso o obstáculo se situe do lado direito, o ponto destino será arrastado para a esquerda, caso contrário será arrastado para a direita. Com esta condição consegue-se otimizar um pouco o trajeto do veleiro, encurtando assim o mesmo.

O processo de arrastamento do ponto D é semelhante ao processo que simula o andamento do veleiro. Mas ao contrário deste último processo (em que o ponto O variava entre o segmento $\overline{OD}$), teremos o ponto D a variar ao longo da reta da meta, reta esta que convém ser infinita pois pode ser necessário o ponto D ser arrastado para muito longe. Como não se consegue trabalhar com valores infinitos, foi atribuído um valor a um ponto na reta da meta muito distante do ponto D (xabmax, yabmax). Com este ponto já se consegue usar as equações aplicadas a um segmento de reta, sendo essas equações as seguintes:

$$xd' = \mathrm{k} * \mathrm{xabmax} + (1 - \mathrm{k}) * \mathrm{xd} \quad (51)$$

$$yd' = \mathrm{k} * \mathrm{yabaux} + (1 - \mathrm{k}) * \mathrm{yd} \quad (52)$$

Tendo concluído todos estes processos resta apenas aplicar um pouco de lógica entre eles.

Então o principio de funcionamento relativo Figura 3.14 é o seguinte:

Fase 1: O veleiro encontra-se no ponto O e está em rota de colisão com o obstáculo encontrando-se este à sua esquerda.

Fase 2: O ponto D é então arrastado varias vezes para a direita e o seu valor final D′ é definido assim que o veleiro deixar de estar em rota de colisão.

Fase 3: É então chamado o processo que simula o andamento do veleiro ao longo do segmento $\overline{OD'}$. De notar que é dado um determinado intervalo de tempo para o veleiro avançar M metros na direção de D′ até que seja novamente verificado se existe ou não perigo de colisão. Esta distância percorrida M é crucial para o funcionamento do algoritmo uma vez que se M fosse zero significaria que o ponto O iria fazer uma tangencia ao obstáculo com uma distância ao mesmo a tender para zero. Isto funcionaria se o veleiro tivesse uma largura a tender para zero, mas como isso não acontece tem de se ter em conta a largura do mesmo, bem como garantir uma distância de segurança razoável ao obstáculo. Neste algoritmo em Matlab o valor de M é ajustado em função da distância de O a D, mas quando for implementado na realidade poderá ser necessário algum ajuste.

O valor de M não é inserido diretamente no algoritmo, este valor vai ter em consideração o valor de X da função que simula o andamento do veleiro. O valor de M será então igual a X multiplicado por N (sendo N o número de vezes que se chama a função que faz andar o veleiro antes de o algoritmo voltar a calcular um novo ponto de destino D′).

Terminado este tempo o algoritmo inicia e volta à fase 1, com a única variante do ponto O, estando este situado um pouco mais perto do destino inicial.

3.3.3.1 Calculo dos pontos A e B caso não sejam conhecidos

Como foi dito anteriormente todos estes processos são válidos quando os pontos A e B são conhecidos (é o que acontece nesta competição), mas como não acontece em todas as competições, para o algoritmo funcionar de forma semelhante, teriam de se fazer algumas aproximações.

De notar que é vantajoso a utilização de um segmento de reta para o destino ao invés de um único ponto, pois pode acontecer que o obstáculo se situe exatamente em cima do ponto de destino e nesse caso o objetivo deixaria de ser passar exatamente por esse ponto, mas sim um pouco ao lado. Nessas situações esse segmento pode ser útil, mas também em situações em que o vento ou a corrente não estejam favoráveis sendo assim necessário que o ponto D esteja situado mais próximo ou do ponto A ou do ponto B.

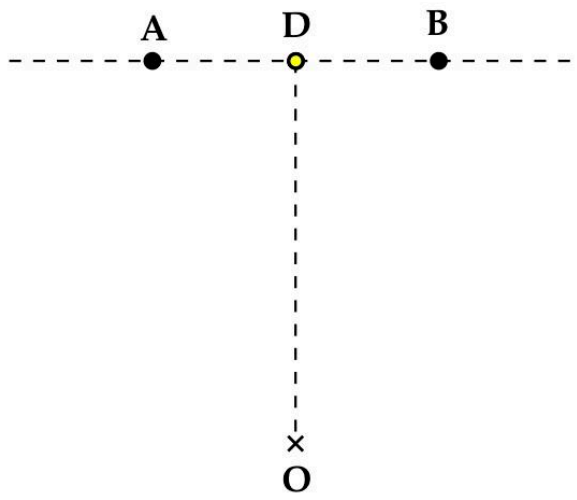

Figura 3.22 - Ilustração auxiliar ao calculo dos pontos A e B.

Se os pontos D e O forem conhecidos, e o objetivo seja criar uma reta auxiliar para integrar os pontos A e B, a solução mais fácil será ter a reta à qual os pontos A e B pertencem perpendicular ao segmento de reta $\overline{OD}$, deste modo o declive do segmento e da reta será o inverso negado um do outro.

Ou seja, para o segmento de reta $\overline{AB}$ e $\overline{OD}$ serem perpendiculares, se o declive do segmento $\overline{OD}$ for dado por *mod*, o declive do segmento $\overline{AB}$ tem de ser -1/*mod*.

Sabendo o declive da reta que contem os pontos A e B, para obter o resto da sua equação é só substituir na equação geral da reta pelo ponto D e calcular o ponto em que a reta corta o eixo do y.

Deste modo tem-se:

$$mab = -\frac{1}{mod} \quad (53)$$

$$bab = yd - mab * xd \quad (54)$$

A partir desta equação já se pode retirar as coordenadas dos pontos A e B, sendo utilizado o mesmo algoritmo que foi utilizado anteriormente no arrastamento do ponto D.

CAPÍTULO 4

VALIDAÇÕES E RESULTADOS

4.1 Validações através de simulação

Neste ponto será validada a solução proposta para a situação com vento moderado (4.1.1) pois é a que apresenta uma maior complexidade a nível de implementação, sendo que para as outras situações será apenas necessário alterar as variáveis de entrada.

Serão também feitas validações e simulações gráficas através de Matlab do sistema total com duas variantes (4.1.2). A primeira utilizando uma simulação do comportamento da Pixy simplificada (4.1.2.1) e a segunda utilizando uma simulação do comportamento da Pixy mais real (4.1.2.2). A versão simplificada serviu apensas para ir testando o algoritmo nas fazes iniciais.

4.1.1 Validação da solução proposta na situação com Vento Moderado

Para comprovar todos os procedimentos do ponto 3.2.2 foi feito um simulador da Pixy em Matlab que simula todas as situações da Figura 3.9.

Para a simulação cujo resultado gráfico é visível na Figura 4.1, o valor considerado para a inclinação do veleiro foi de θ igual a 60º.

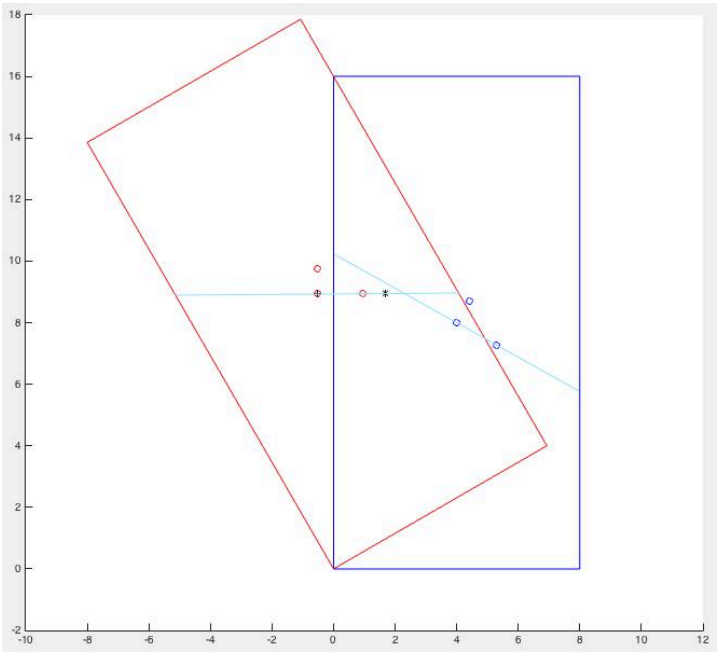

Figura 4.1 - Simulação em Matlab da Pixy quando o veleiro está inclinado

O retângulo e os pontos a vermelho da Figura 4.1, representam a situação (a) da Figura 3.9, ou seja, a imagem real em que a linha do horizonte está com zero graus de inclinação. O único ponto a vermelho que não interceta nenhuma reta representa o limite superior do obstáculo, o ponto a vermelho que contem um sinal "+" a preto no seu interior representa o centro do mesmo, já o ponto a vermelho mais à direita marca o seu limite lateral direito. Nesta imagem o horizonte é a reta a azul claro, que é paralela à reta que contem o ponto central do obstáculo e o ponto lateral direito do mesmo.

O retângulo e os pontos a azul representam a situação (b) da Figura 3.9, ou seja, a imagem recebida pela Pixy da situação anterior. Como se pode observar o eixo do horizonte está inclinado (90º - 60º), ou seja 30º para a direita.

Nesta situação, com o auxilio da Figura 3.10, foi simulado o valor da altura e da largura que a Pixy iria devolver caso a situação fosse real e com esses valores, foi calculada a *largura_max* que o obstáculo poderá ter.

Estes cálculos foram feitos da seguinte forma:

Sendo,

$$T = \sqrt{(x_A - x_O)^2 + (y_A - y_O)^2} \quad (55)$$

$$V = \sqrt{(x_C - x_O)^2 + (y_C - y_O)^2} \quad (56)$$

$$\alpha' = T * sen(\theta) \quad (57)$$

$$\alpha = V * cos(\theta) \quad (58)$$

$$\beta' = V * sen(\theta) \quad (59)$$

$$\beta = T * cos(\theta) \quad (60)$$

A largura que a Pixy iria devolver será então (α+α') e a altura (β+β').

Então,

$$largura\_max = 2 * \sqrt{(\beta + \beta')^2 + (\alpha + \alpha')^2} \quad (61)$$

Tendo calculado o ponto central do obstáculo (ponto a azul mais à esquerda) e o valor máximo da largura que o obstáculo poderá ter (*largura_max*), foi feita então a rotação desse ponto e de todos os vértices em (90º – 60º) graus para a esquerda, ou seja, de 30º.

Pode então visualizar-se na Figura 4.1 o ponto central corretamente rodado, estando representado por um sinal "+" a preto circundado por um circulo a vermelho (querendo isto dizer que foi parar ao sitio correto). Na mesma figura pode também observar-se um sinal "*" a preto, correspondendo este ao ponto lateral direito do obstáculo. A coordenada em y deste ponto é igual à coordenada em y do ponto central do obstáculo depois de rodado, e a coordenada em x é calculada somando à coordenada em x do ponto central depois de rodado a variável *largura_max* calculada anteriormente.

Como se pode observar, esta simulação comprova a proposta inicial que visa resolver a situação da inclinação do veleiro.

O ponto lateral direito do obstáculo não é exato, mas não há problema, pois só erra por excesso, querendo isto dizer que a largura admitida é um pouco superior à largura real do obstáculo. Este facto só faz com que a distância de segurança entre o obstáculo e o trajeto do veleiro seja um pouco mais confortável.

4.1.2 Simulação gráfica do algoritmo Desenvolvido

Com o algoritmo construído fizeram-se algumas simulações com coordenadas reais. De notar que se fizeram dois tipos de simulação diferentes, uma com a função de perigo mais simples e outra com a função mais real. A sequencia de imagens serve para ilustrar o algoritmo em funcionamento à medida que o veleiro avança no percurso.

Tabela 4.1 Coordenadas das simulações

Secção	Coordenadas			Raio do circulo a vermelho
4.1.2.1	xa=38.694093421370 ya=-9.206571131944 xd=xd_inicial yd=yd_inicial	xb=38.694093431370 yb=-9.206378012895 xo=38.69369566469 yo=-9.206281453371	xd_inicial=(xa+xb)/2 yd_inicial=(ya+yb)/2 xc=38.69388778449 yc=-9.206350128955	r=0.0001
4.1.2.2	xa=38.694093421370 ya=-9.206571131944 xd=xd_inicial yd=yd_inicial	xb=38.694093431370 yb=-9.206378012895 xo=38.69369566469 yo=-9.206281453371	xd_inicial=(xa+xb)/2 yd_inicial=(ya+yb)/2 xc=38.69388778449497 yc=-9.20635002895585	r=0.00004

4.1.2.1 Simulação gráfica com a função de perigo mais simples

Na Figura 4.2 e na Figura 4.3 pode-se observar o resultado gráfico da simulação da função de perigo mais simples desenvolvida anteriormente. Função esta que tem como base a verificação da interceção da reta que contem os pontos de origem e destino com o obstáculo.

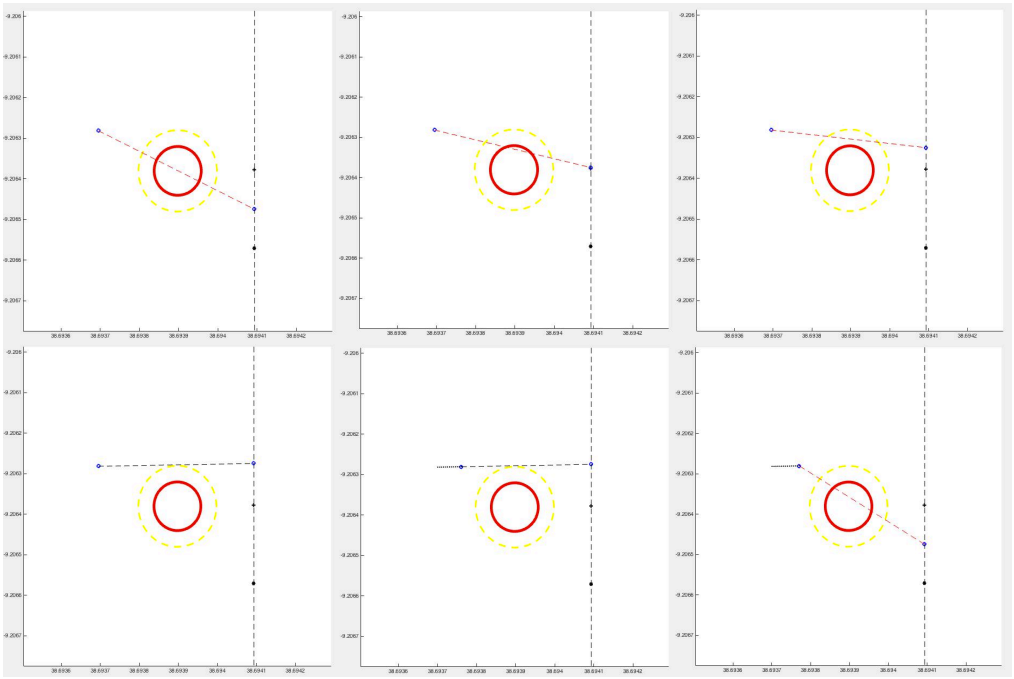

Figura 4.2 - Simulação gráfica com a função de perigo mais simples (a)

A Figura 4.2 é composta por uma sequencia de 6 imagens. Inicialmente pode ver-se que a reta que contem o ponto de origem e o ponto de destino intercepta o obstáculo e por isso está representada a vermelho. Nesta situação o simulador arrasta o ponto de destino ao longo da reta da meta e é feita uma constante verificação da existência de perigo.

Quando deixa de existir perigo o veleiro avança durante um certo tempo em direção ao novo ponto de destino e passado esse tempo o ponto de destino volta a ter o valor inicial. O mesmo processo é repetido até ao fim do percurso.

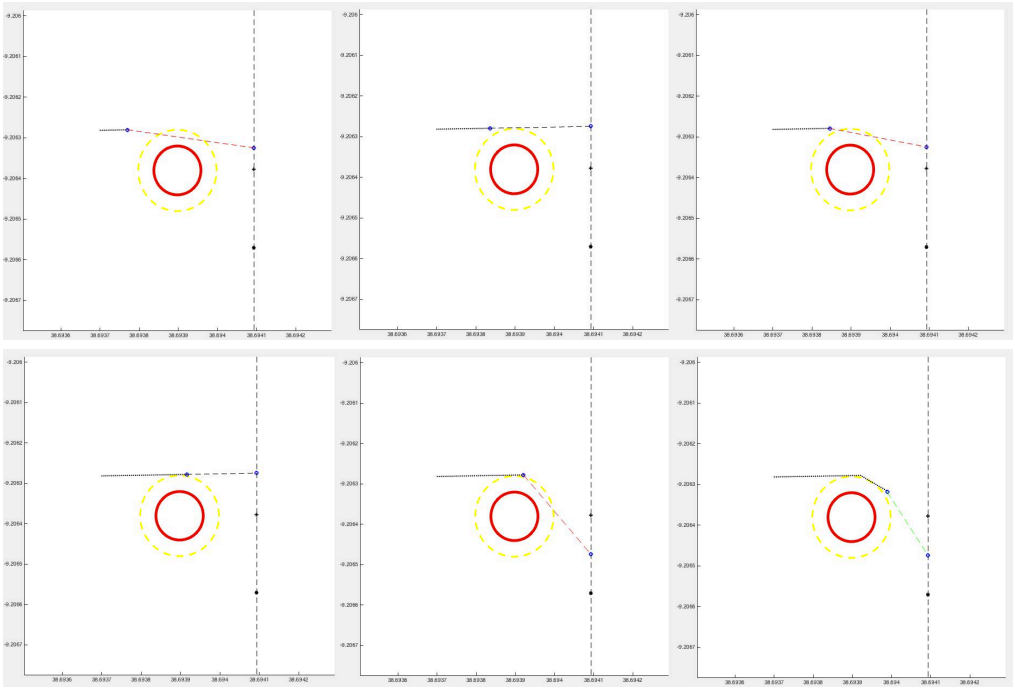

Figura 4.3 - Simulação gráfica com a função de perigo mais simples (b)

Pode concluir-se então que a função de perigo mais simples funciona como esperado. Após a validação desta função através de simulação, está-se em condições de melhorar a maneira como a função perigo funciona tornando-a assim mais próxima à realidade. Na próxima secção (4.2.2) são apresentados os resultados da simulação dessa função mais complexa.

4.1.2.2 Simulação gráfica com a função de perigo mais complexa

Neste ponto são apresentados os resultados da simulação gráfica da função de perigo mais complexa, função essa que pretende simular a Pixy de uma forma mais realista. Como se pode observar pelas figuras 4.4 e 4.5 o "veleiro" deteta corretamente o "obstáculo" e o algoritmo anticolisão entra imediatamente em ação, alterando a coordenada de destino ciclicamente e efetuando um percurso sem perigo de colisão.

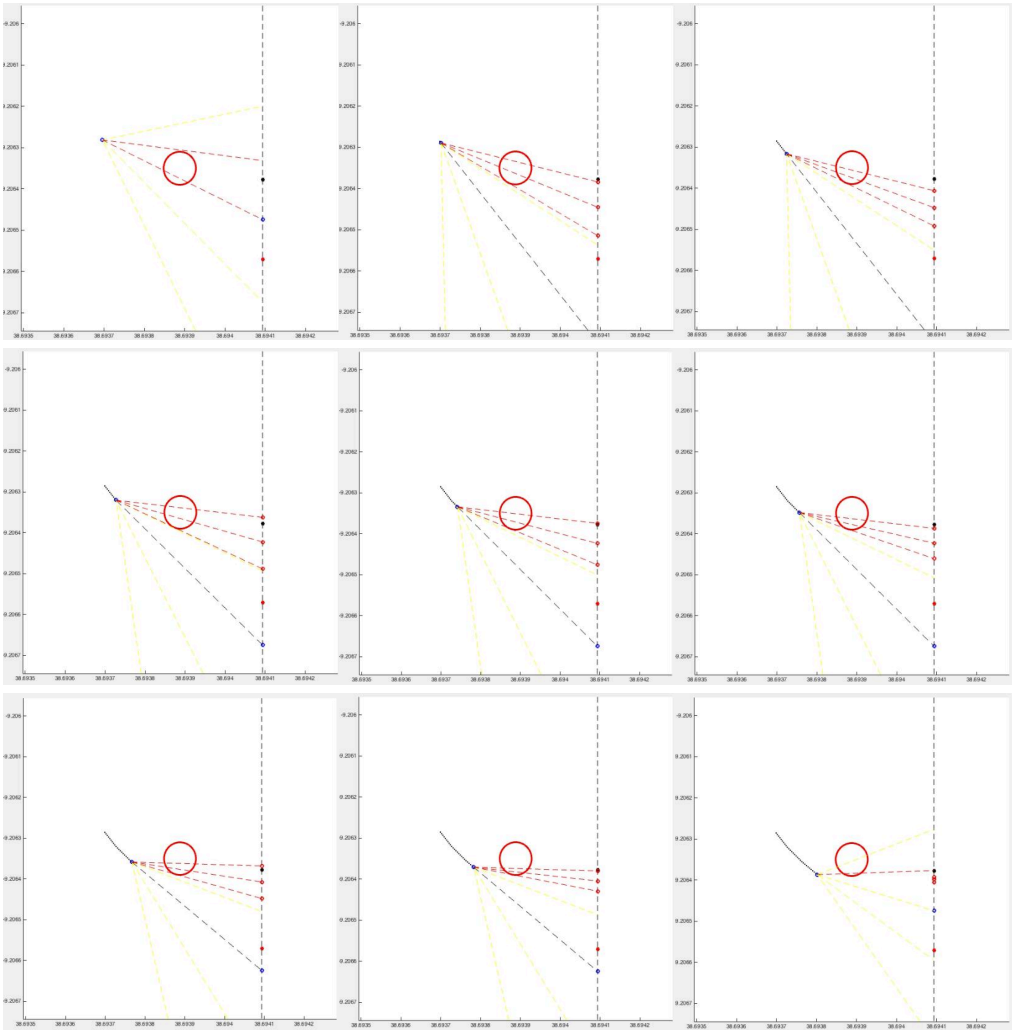

Figura 4.4 Simulação gráfica com a função de perigo mais complexa (a)

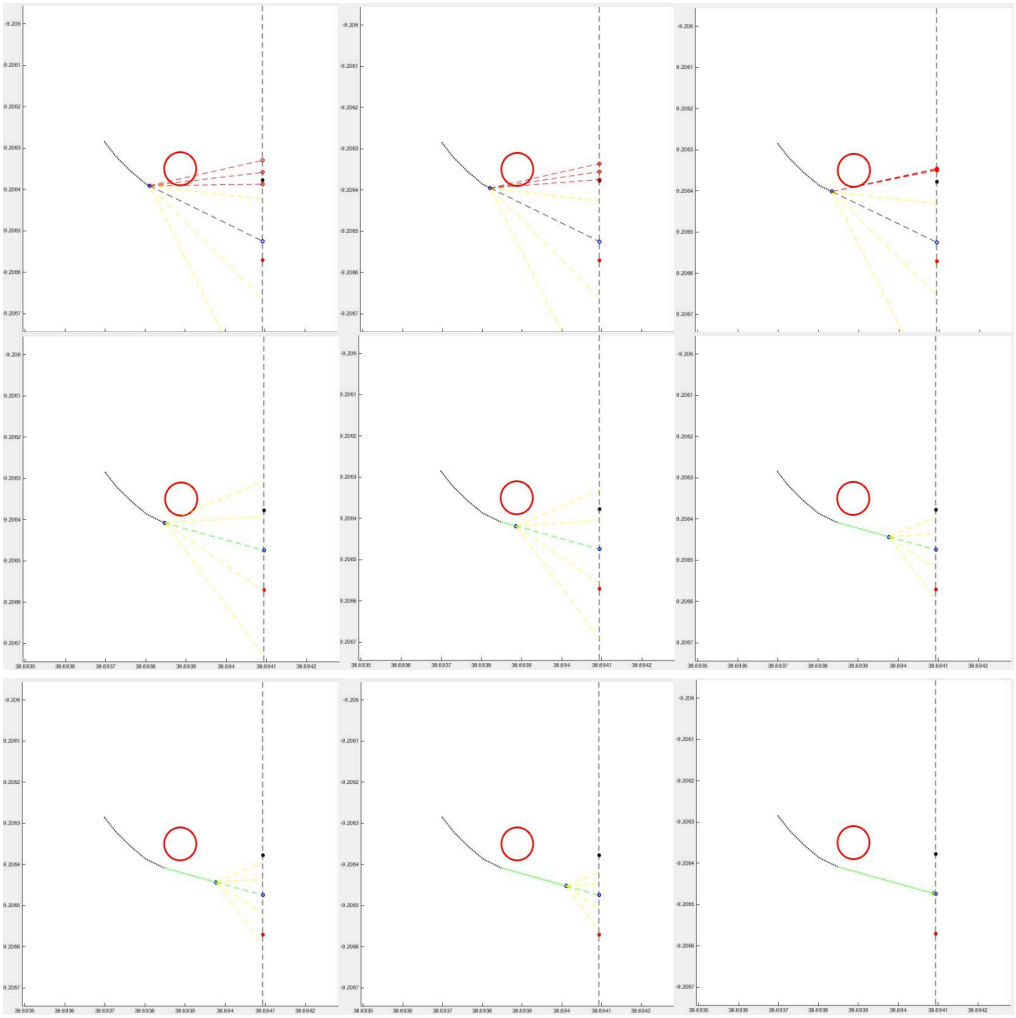

Figura 4.5 Simulação gráfica com a função de perigo mais complexa (b)

Tal como as simulações do ponto anterior (4.2.1) os resultados são os esperados, e por isso conclui-se que esta função também funciona corretamente.

4.2 Validações laboratoriais

Para testar as soluções propostas nas subsecções do ponto 3.2, e recorrendo à lógica descrita nas mesmas secções, foram desenvolvidos e integrados no Arduino os vários algoritmos. Na Figura 3.2 e na Figura 3.5 está descrito parte desse algoritmo. Nestes testes, foi atribuído à variável AREA_MAXIMA o valor 1000.

Após a conclusão dos algoritmos, ligou-se a Pixy ao Arduino e fizeram-se alguns testes no ambiente controlado ilustrado na Figura 4.6. O Arduino foi ligado a um computador para visualizar as respostas dos algoritmos.

O campo de visão da Pixy é muito parecido com o campo desenhado no fundo da Figura 4.6, mas em vez de ser composto por linhas retas, é composto por linhas curvas. No entanto, esta aproximação é considerada aceitável, uma vez que o erro gerado será insignificante.

4.2.1 Situação simplificada

Teste com a lógica do ponto 3.2.1.

Na Figura 4.7 está um exemplo de uma imagem recebida pela Pixy. A linha vertical que se encontra no centro da imagem corresponde ao centro do referencial da Pixy, ou seja, ao ponto C da Figura 3.1, e as duas linhas ao lado correspondem nessa imagem aos pontos B e D da mesma figura, querendo isto dizer que a zona de perigo está entre estas duas últimas linhas.

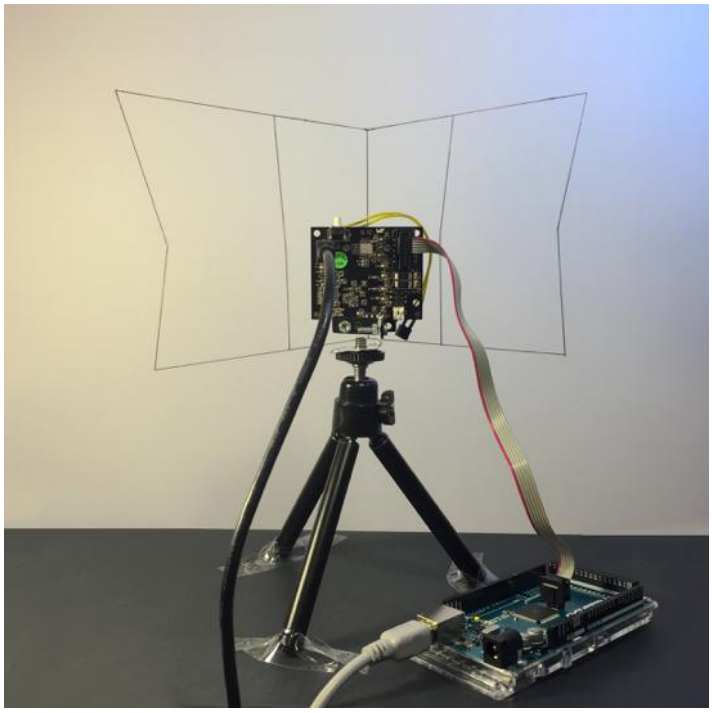

Figura 4.6 - Ambiente de testes do código em Arduino com θ=90º

Nesta figura, foi colocado um objeto no campo de visão anteriormente descrito onde parte desse objeto está dentro da zona de perigo. O resultado devolvido pelo algoritmo era o esperado e está ilustrado Figura 4.8.

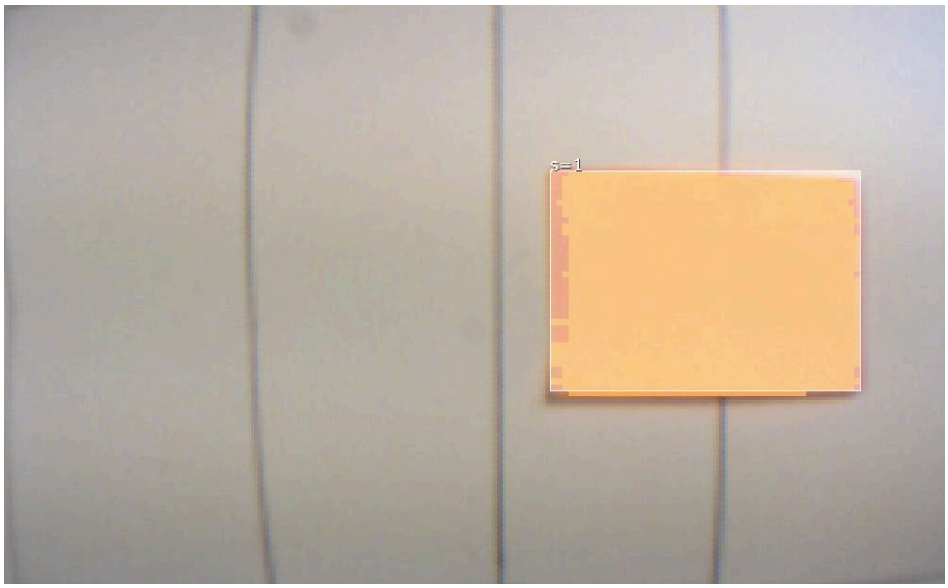

Figura 4.7 - Imagem recebida pela Pixy

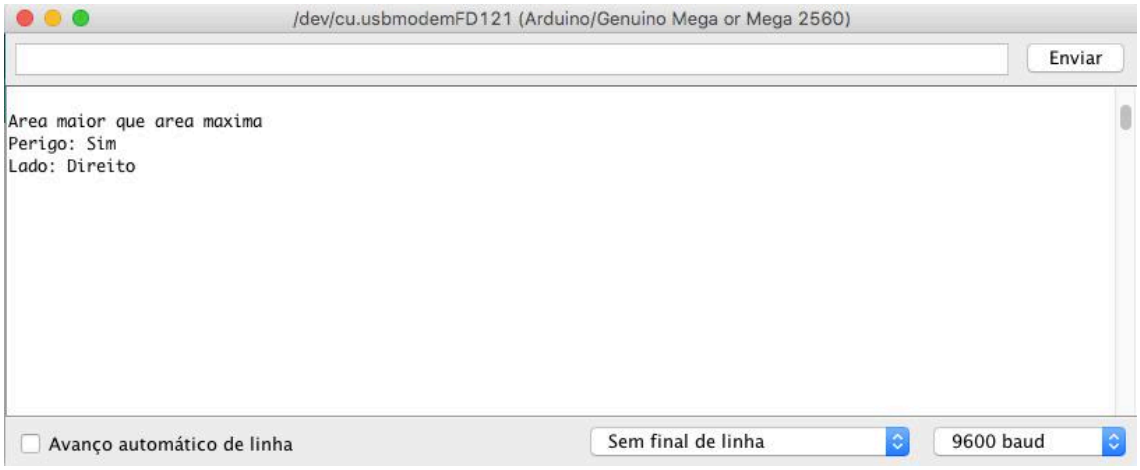

Figura 4.8 - Resultado do algoritmo integrado no Arduino

Após o primeiro resultado ser o esperado foram realizados muito mais testes para comprovar que o algoritmo realmente funcionava em todas as situações. Na Figura 4.9 pode-se ver alguns desses testes e resultados.

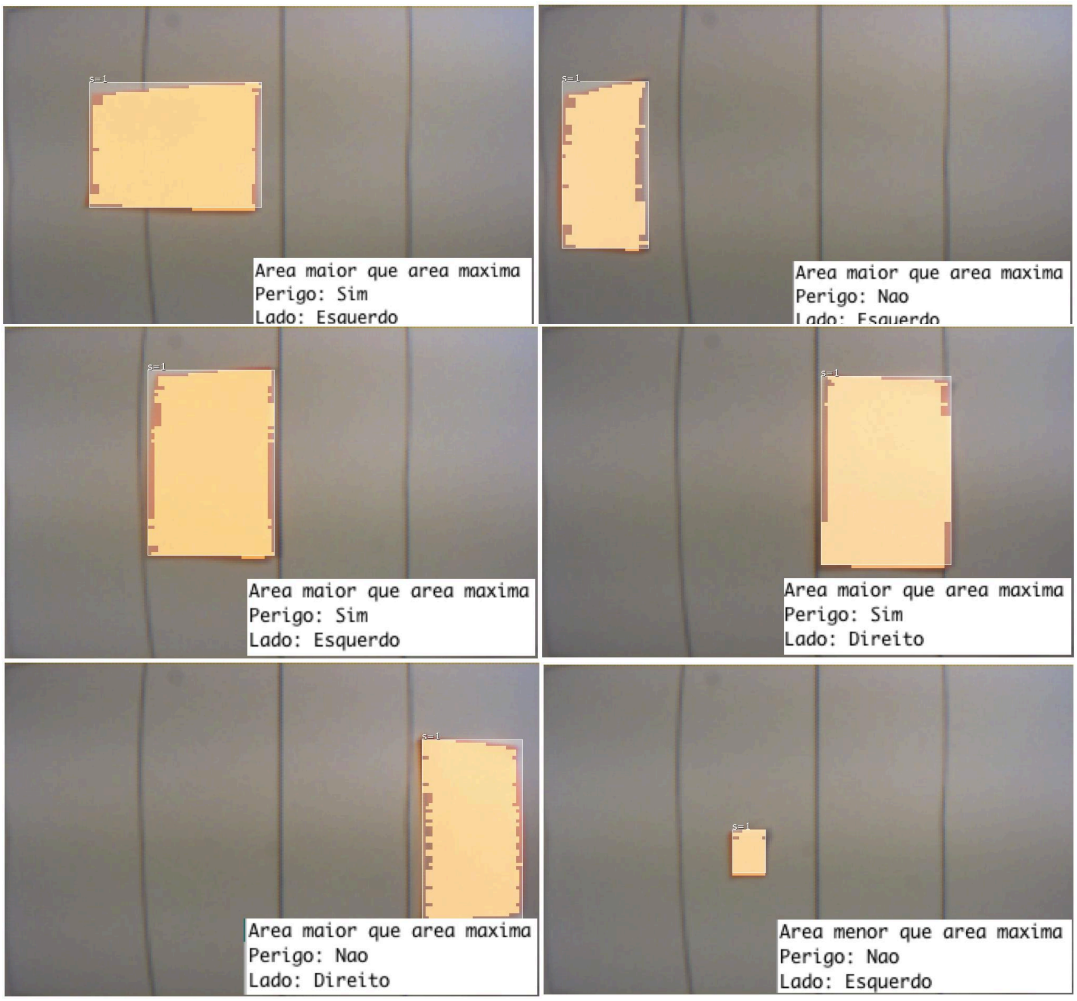

Figura 4.9 - Conjunto de testes e resultados do algoritmo integrado no Arduino

Como se pode observar todos os resultados são os esperados, ficando assim em condições de se dizer que o algoritmo de perigo funciona corretamente.

4.2.2 Situação com vento moderado

Para testar a solução proposta para o ponto 3.2.2, e recorrendo à lógica descrita nesse ponto, desenvolveu-se um algoritmo no Arduino que aplica a rotação anteriormente descrita aos pontos utilizados. Após essa rotação, utilizou-se novamente o algoritmo utilizado nos testes do ponto 4.2.1 para a verificação da existência de perigo. À semelhança desse ponto, após a conclusão do algoritmo, voltou-se a ligar a Pixy ao Arduino e fizeram-se testes no ambiente controlado ilustrado Figura 4.10. Como se pode observar a Pixy foi rodada para a esquerda de modo a que o valor do ângulo θ fosse igual a 55º. O Arduino foi novamente ligado a um computador para visualizar as respostas do algoritmo.

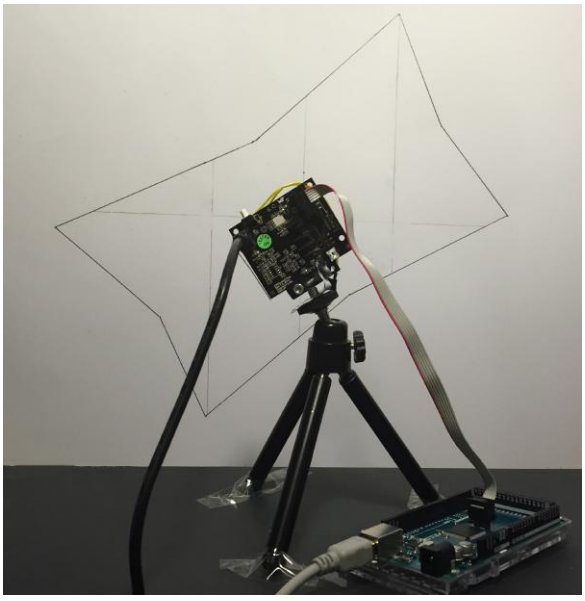

Figura 4.10 - Ambiente de testes do código em Arduino com θ=55º

Na Figura 4.11 está representada a imagem recebida pela Pixy quando θ=55º, ou seja, quando a Pixy está na posição da Figura 4.10. A linha azul foi acrescentada à imagem original e pretende representar a linha do horizonte real. Na imagem também aparecem 3 linhas perpendiculares à linha azul, onde a linha do meio representa o centro do referencial da Pixy depois da rotação e as outras duas linhas limitam a nova zona de perigo.

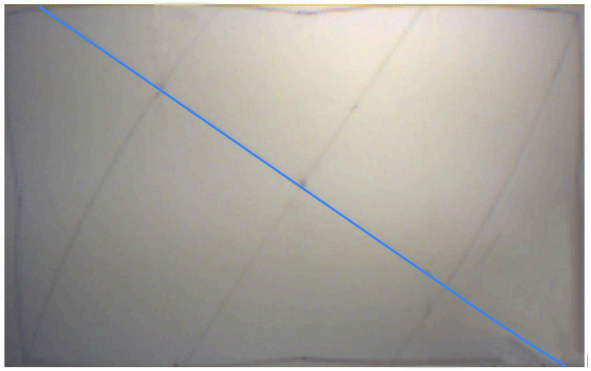

Figura 4.11 - Imagem recebida pela Pixy com θ=55º

Foram então colocados objetos no campo de visão da Pixy e observados os resultados que estão apresentados na Figura 4.12.

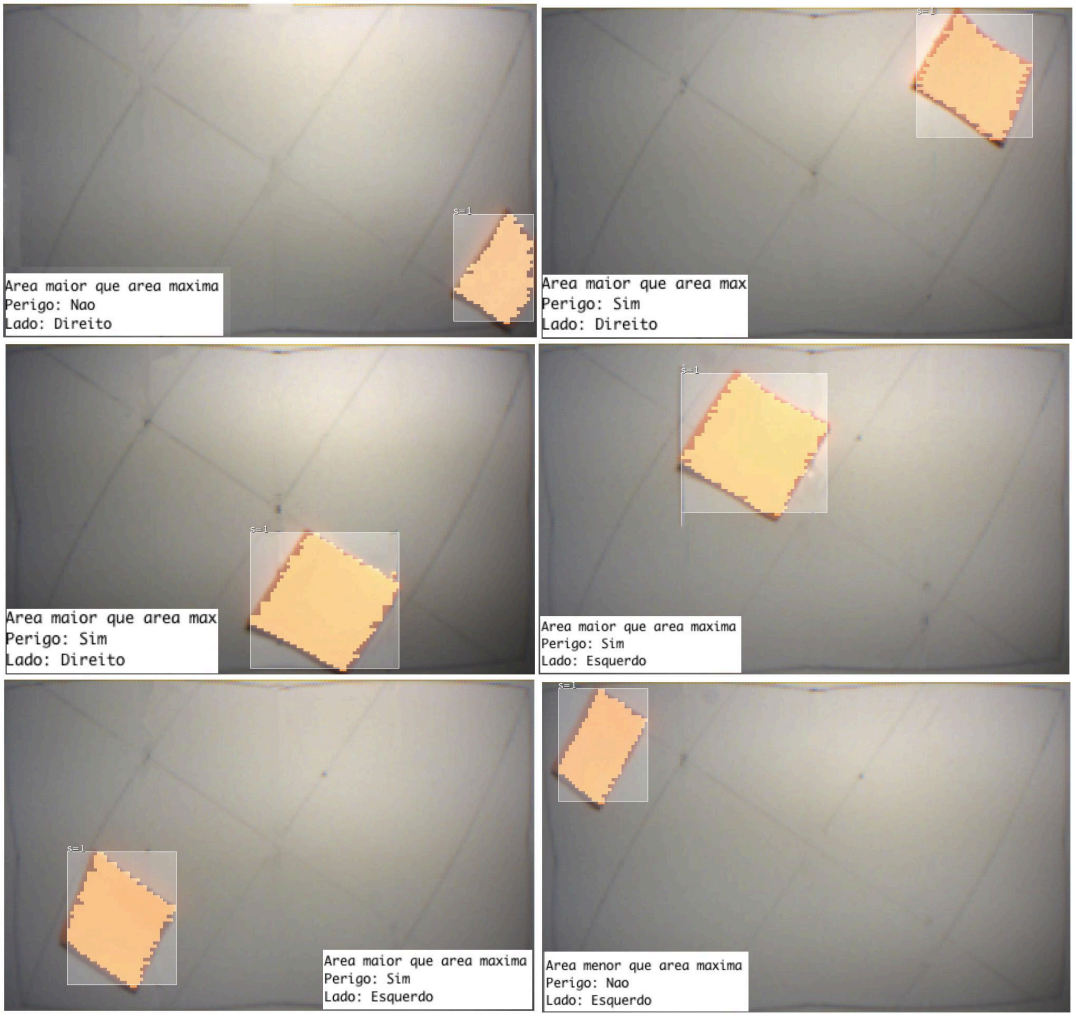

Figura 4.12 - Conjunto de testes e resultados do algoritmo rotação com θ=55º

Como se pode observar todos os resultados são os esperados.

Depois destes testes, foram feitos mais alguns com a Pixy inclinada para a direita em vez de para a esquerda. Novamente todos os testes estavam corretos, ficando assim em condições de se dizer que o algoritmo de rotação funciona corretamente.

4.2.3 Situação com corrente

De modo a testar a lógica desenvolvida no ponto 3.2.3, a zona de perigo foi deslocada para a direita e para a esquerda e em ambas as situações foram feitos testes práticos. Tais testes e resultados podem ser visualizados na Figura 4.13 e na Figura 4.14.

A zona de perigo foi deslocada em (75/4)º, ou seja, em 18,75º o que equivale a deslocar o centro da zona de perigo (PIXY_MAX_X-PIXY_MIN_X)/4 para a direita ou para a esquerda.

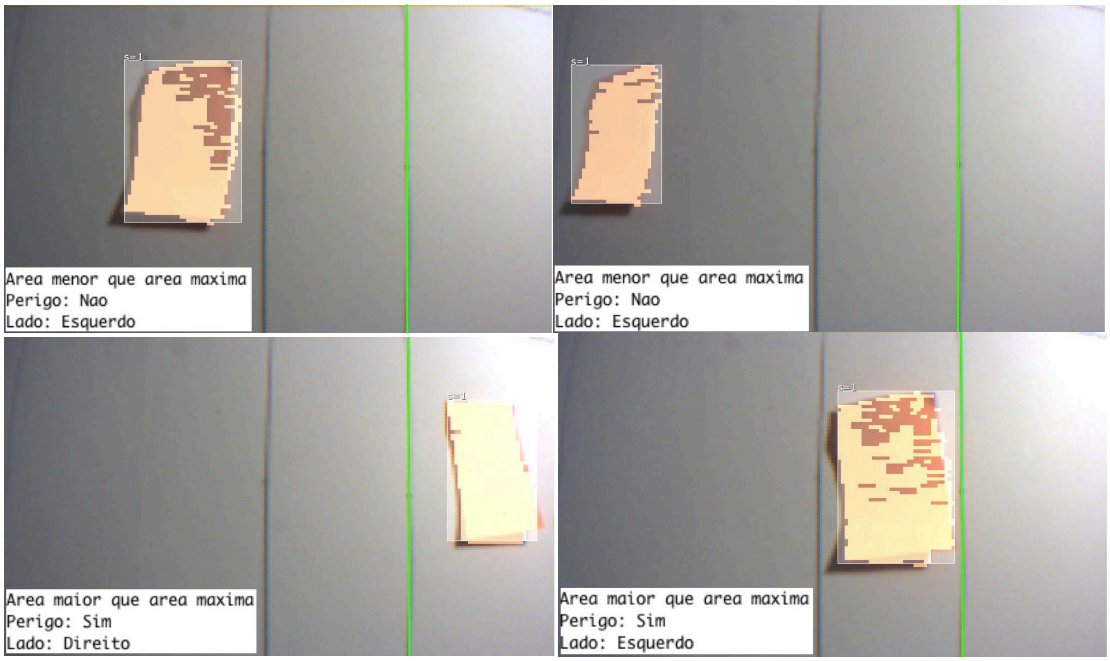

Figura 4.13 – Testes e resultados. Zona de perigo deslocação para a direita

Pode visualizar-se na Figura 4.13 varias linhas verdes. Essas linhas foram desenhadas após os testes e servem para representar o novo centro da zona de

perigo. A totalidade dessa zona vai desde a linha central a preto até à extremidade direita de cada imagem, isto para o caso em que a deslocação da zona de perigo é efetuada para a direita. Quando a deslocação é efetuada para a esquerda será a mesma lógica, mas para o lado oposto.

Como foi explicado, o pretendido seria deslocar o centro da zona em graus, mas tal não é possível. Então, tem de se fazer uma equivalência entre os graus e a largura da imagem de forma a se obter os valores pretendidos.

Para tal, teve-se em conta o ângulo de visão da Pixy ser 75º e a largura da imagem equivalente (PIXY_MAX_X-PIXY_MIN_X). Se o objetivo for deslocar para a direita o centro da zona de perigo em (75/4 º) o equivalente será somar ao seu valor atual (PIXY_MAX_X-PIXY_MIN_X)/4. A partir desta lógica é fácil fazer a equivalência de qualquer valor em graus para os valores de largura de imagem utilizadas na Pixy.

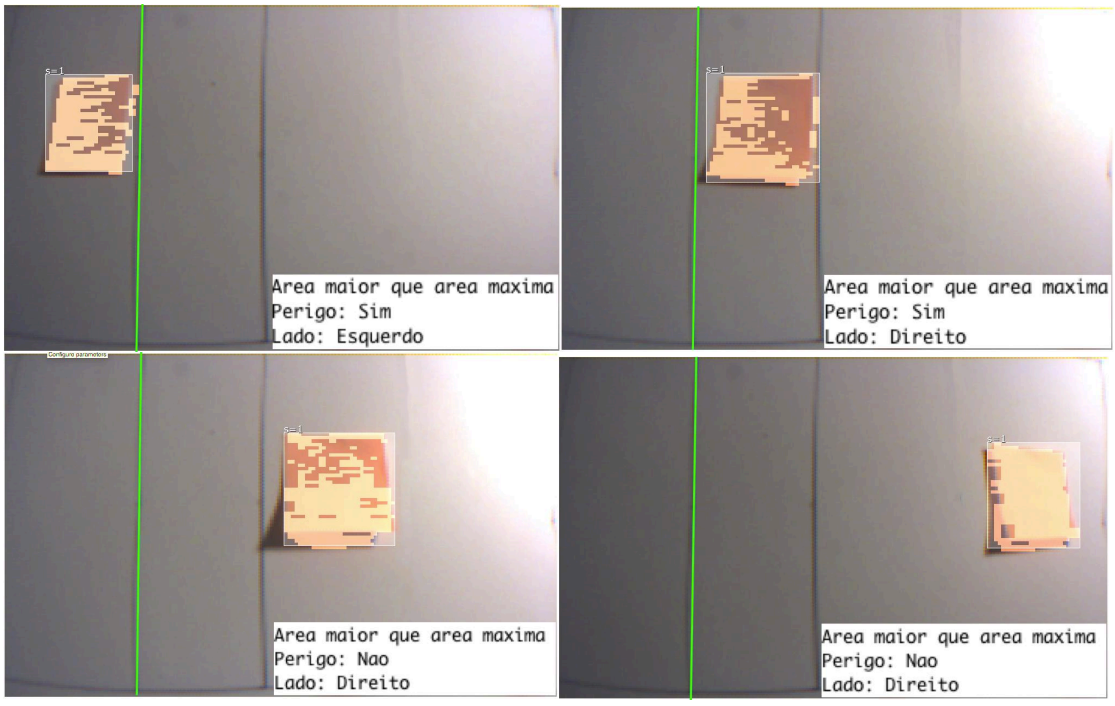

Figura 4.14 - Testes e resultados. Zona de perigo deslocação para a esquerda

Como se pode comprovar todos os testes efetuados tiverem os resultados pretendidos.

4.2.4 Situação com múltiplos obstáculos

Depois de desenvolver a lógica descrita no ponto 3.2.5, foi criado o algoritmo a inserir no Arduino e foram feitos alguns testes.

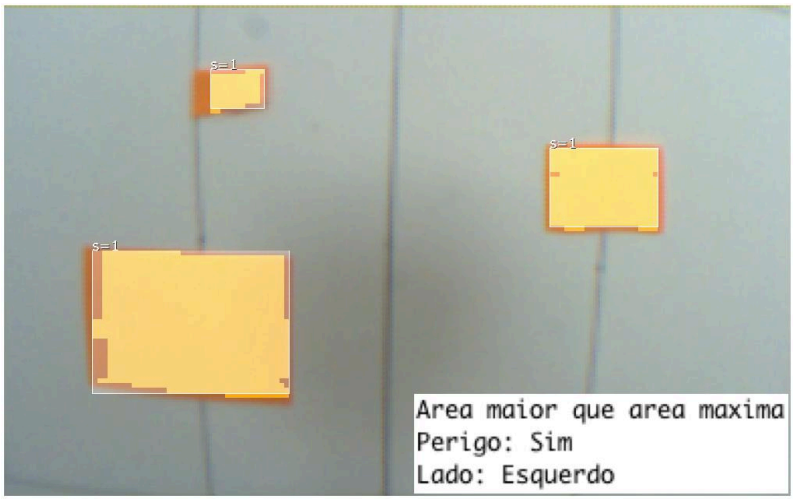

Figura 4.15 - Teste do algoritmo que escolhe só o obstáculo maior

A Figura 4.15 é o exemplo de um desses teste e pode observar-se que o algoritmo responde como pretendido, uma vez que só considera um dos obstáculos (o de maior área).

CAPÍTULO 5

CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação apresenta uma solução de baixo custo para implementar uma estratégia de prevenção de colisões para um modelo de pequena escala de um veleiro com navegação autónoma.

O projeto teve como base o sensor Pixy que é um módulo autónomo de aquisição e processamento de imagens que foi integrado na plataforma Arduino atualmente utilizada para controlar o veleiro usado no laboratório.

Inicialmente foi desenvolvido o algoritmo para deteção de perigo. Este algoritmo tem em consideração as variáveis de entrada que provêm do sensor ótico Pixy e como variáveis de saída a existência ou não de perigo e de que lado o perigo se encontra. Primeiramente foi desenvolvido o algoritmo de perigo para uma situação simplificada, ou seja, para uma situação em que a Pixy se encontra completamente estável e nivelada e posteriormente foram feitas algumas adaptações que já consideram a inclinação e desnivelamento da Pixy, situações que ocorrem naturalmente quando existe vento, corrente ou ondulação. Os algoritmos anteriormente referidos foram inseridos no Arduino, testados e validados em laboratório onde se observou a sua eficácia.

Após testada e validada a parte da receção de dados provenientes da Pixy e também a parte do sistema de deteção de perigo, foi desenvolvido o sistema anticolisão para o veleiro se desviar das boias que se encontrem na sua rota. Para simular este sistema de anticolisão em ambiente Matlab foram desenvolvidas algumas funções auxiliares. Funções essas que servem para simular o andamento do veleiro, simular a Pixy e o algoritmo de perigo anteriormente desenvolvido e validado.

Finalmente o algoritmo de anticolisão foi simulado em Matlab em várias coordenadas distintas e validado em todas elas.

Conclui-se através dos testes realizados em laboratório a validade das propostas apresentadas antecipando como trabalho futuro a sua integração no controlador do veleiro desenvolvido na faculdade.

Como trabalho futuro também se sugere a inclusão no controlador atualmente disponível de estratégias de prevenção de colisões entre diferentes veleiros, assegurando uma navegação de acordo com as regras comuns de navegação à vela. Tal poderá ser realizado através da comunicação direta com outros veleiros (através da troca de informações sobre a posição, direção e velocidade), onde cada veleiro poderá ser considerado como um componente totalmente integrado num ambiente de sistema partilhado, complementando assim as suas capacidades de navegação autónoma.

Referências

[1] "WRSC/IRSC." [Online]. Available: http://www.roboticsailing.org/en/2008. [Accessed: 27-Feb-2017].

[2] The International Robot Sailing Conference, "World Robotic Sailing Championship," 2016. [Online]. Available: http://www.roboticsailing.org/. [Accessed: 27-Feb-2017].

[3] "SailBot, International Robotic Sailing Competition." [Online]. Available: http://sailbot.org/?page_id=9. [Accessed: 27-Feb-2017].

[4] "Microtransat, The Microtransat Challenge." [Online]. Available: http://www.microtransat.org/. [Accessed: 27-Feb-2017].

[5] J. Esteves, L. Gomes, and A. Costa, "Collision Avoidance System for an Autonomous Sailboat," *IECON'2017 – 43rd Annu. Conf. IEEE Ind. Electron. Soc.*

[6] H. M. C. Marques, "Visualizer of sailboat navigation integrating instrumentation emulators," Dissertação de Mestrado, FCT-UNL, 2017.

[7] J. C. Alves and N. A. Cruz, "FASt - An autonomous sailing platform for oceanographic missions, DOI: 10.1109/OCEANS.2008.5152114," *Ocean. 2008*, pp. 0–6, 2008.

[8] J. C. Alves and N. A. Cruz, "Um sistema computacional reconfigurável embarcado num veleiro autónomo, https://repositorio-aberto.up.pt/handle/10216/70017," *Actas das V Jornadas sobre Sist. Reconfiguráveis - REC2009*, p. 8, 2009.

[9] H. Erckens, G. A. Büsser, C. Pradalier, and R. Y. Siegwart, "Avalon: Navigation strategy and trajectory following controller for an autonomous sailing vessel," *IEEE Robot. Autom. Mag.*, vol. 17, no. 1, pp. 45–54, 2010.

[10] L. Gomes, M. Santos, T. Pereira, and A. Costa, "Model-Based Development of an Autonomous Sailing Yacht Controller, DOI: 10.1109/ICARSC.2015.20," *ICARSC'2015 – 2015 IEEE Int. Conf. Auton. Robot Syst. Compet.*, pp. 103–108.

[11] M. F. P. Santos, "eVentos 4 - Controlador para navegação autónoma de veleiro em modo de regata," Dissertação de Mestrado, FCT-UNL, 2015.

[12] L. Gomes, A. Costa, F. Moutinho, and R. Mota, "Attracting students to engineering through autonomous sailing yacht development," *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2015–June, no. June, pp. 3252–3257, 2015.

[13] L. Gomes, A. Costa, D. Fernandes, H. Marques, and F. Anjos, "Improving instrumentation support and control strategies for autonomous sailboats in a regatta contest," *; ISRC'16 - 9th Int. Robot. Sail. Confer; Sept. 5-10, 2016; Viana do Castelo, pt*, pp. 1–12.

[14] R. Stelzer, T. Pröll, R. Stelzer, T. Pröll, and R. I. John, "Fuzzy Logic Control System for Autonomous Sailboats, DOI: 10.1109/FUZZY.2007.4295347," 2007.

[15] B. Gutenberg, "Propagation of Sound Waves in the Atmosphere," vol. 226, no. 1942, C. I. of Technology, Ed. California, 2014, pp. 151–155.

[16] “ITEAD itead.cc, Waterproof Ultrasonic Sensor.” [Online]. Available: https://www.itead.cc/waterproof-ultrasonic-sensor.html. [Accessed: 27-Feb-2017].

[17] J. . HALLIDAY, R.;RESNICK,R.; WALKER, *Fundamentos de Física, Volume 2: Gravitação, Ondas e Termodinâmica.* Rio de Janeiro, 2009.

[18] “LIDAR-Lite, Dragon Innovation.” [Online]. Available: http://www.dragoninnovation.com/projects/32-lidar-lite-by-pulsedlight. [Accessed: 01-Mar-2017].

[19] “LIDAR-Lite v3, GARMIN.” [Online]. Available: https://buy.garmin.com/en-US/US/p/557294. [Accessed: 01-Mar-2017].

[20] “LIDAR Lite v3 Arduino Library.” [Online]. Available: https://github.com/garmin/LIDARLite_v3_Arduino_Library. [Accessed: 01-Mar-2017].

[21] “Sensor UT390B da Uni-T.” [Online]. Available: http://www.uni-trend.com/productsdetail.aspx?ProductsID=792&ProductsCateId=806&CateId=806. [Accessed: 01-Mar-2017].

[22] “Solder and Flux, Arduino laser distance meter.” [Online]. Available: http://blog.qartis.com/arduino-laser-distance-meter/. [Accessed: 01-Mar-2017].

[23] “AIS International Maritime Organization.” [Online]. Available: http://www.imo.org/en/OurWork/Safety/Navigation/Pages/AIS.aspx. [Accessed: 01-Mar-2017].

[24] “AIS Australian Government.” [Online]. Available: https://www.amsa.gov.au/navigation/services/ais/. [Accessed: 01-Mar-2017].

[25] BAILEY and JANSKY, *The Loran-C System of Navigation*, no. February. Washington, D. C., 1962.

[26] “Electronic Design, Pixy.” [Online]. Available: http://electronicdesign.com/embedded/smart-programming-and-peripherals-reduce-power-requirements. [Accessed: 27-Feb-2017].

[27] T. Yamawaki and S. Yamano, “60-GHz millimeter-wave automotive radar,” *Fujitsu Ten Tech. J.*, no. 11. 1998.

[28] “Panasonic, Millimeter-wave Radar Technology.” [Online]. Available: http://www.panasonic.com/global/corporate/technology-design/technology/radar.html. [Accessed: 01-Mar-2017].

[29] “IEE Milimeter wave radars.” [Online]. Available: http://www.grss-ieee.org/millimeter-wave-radars/. [Accessed: 01-Mar-2017].

[30] “Arduino Mega 2560.” [Online]. Available: http://www.electroschematics.com/7963/arduino-mega-2560-pinout/. [Accessed: 27-Feb-2017].

[31] “Arduino Mega.” [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMega2560. [Accessed: 27-Feb-2017].

[32] “Raspberrypi.” [Online]. Available: https://www.raspberrypi.org/. [Accessed: 01-Mar-2017].

[33] “opensource.com, What is Raspberry Pi.” [Online]. Available: https://opensource.com/resources/what-raspberry-pi. [Accessed: 01-Mar-2017].

[34] “Raspberry Pi 3.” [Online]. Available: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/. [Accessed: 01-Mar-2017].

[35] D. Miranda, R. Grisi, and S. Lodovici, *Geometria Analítica e Vetorial*, Versão 9. Santo André, Universidade Federal do ABC, 2015.