

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Condução Visual de Embarcações Autónomas

Ricardo Jorge Moreira da Silva Neves

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Dr. Aníbal Castilho Coimbra de Matos

23 de Julho de 2013

Resumo

Ao longo desta dissertação, é explorada a possibilidade de implementação de um sistema de *obstacle avoidance* e *target tracking* baseado em sistemas computacionais recentes de muito baixo custo e câmaras igualmente económicas.

O principal interesse é o de completar as funcionalidades dos ASV's do OceanSys - DEEC/-FEUP, dotando-os de um sistema de visão estereoscópica que permita navegação em albufeiras, lagos ou portos que pese, ao mesmo tempo, o custo energético, computacional e preço da solução.

Inicialmente fez-se um estudo do hardware disponível para o efeito e dos métodos computacionais mais comuns em aplicações deste tipo.

Ao longo do trabalho, foi adaptado um par de câmaras por forma a obter imagens sincronizadas e foram testados alguns algoritmos que permitem identificar objetos, tendo a escolha final sido feita com base numa avaliação os tempos de execução em diversas plataformas computacionais.

Foi feito um estudo sobre visão estereoscópica e foi calibrado o par de câmaras.

Foi também experimentada a *toolbox* de *Matlab* recentemente disponibilizada, que permite a criação de programas em ambiente *Simulink*, diretamente transferíveis e executáveis em Raspberry Pi. Os resultados da sua utilização mostraram-se interessantes para visão monocular mas insuficientes no que respeita a visão estereoscópica.

O sistema final foi integrado na embarcação e testado em laboratório.

Abstract

Throughout this dissertation, we explore the possibility of implementation of an obstacle avoidance and target tracking system, supported by a very recent and low cost computational unit and equally inexpensive cameras. The main interest of this work is adding new functionalities to the OceanSys - DEEC/FEUP ASV's, providing them with a stereo vision system that allows autonomous navigation in reservoirs, lakes or ports and is able to balance, at the same time, energetic and computational cost and final price.

At first, a research was made to evaluate the available hardware in the market and the most common computational methods used in applications of this kind.

Along the work, a pair of webcams was adapted to be able to acquire synchronized image pairs and several algorithms designed with the goal of object identification were tested. The final choice was made based on the execution times experienced in several computational platforms.

A study about stereo vision was made and a camera pair was calibrated.

The new Matlab toolbox for Raspberry PI was also tested, but the end result with stereo vision applications didn't fulfill this kind of application's timing needs.

The final system was integrated in the ASV and tested in our laboratory.

Agradecimentos

Neste que é o último trabalho deste ciclo de estudos, além dos agradecimentos aos que nele comigo diretamente colaboraram, importa referir os nomes das pessoas com quem fui tendo a sorte de me ir cruzando nos corredores e nas salas de aula desta escola onde tive o privilégio de poder aprender.

À Faculdade de Engenharia, que me pôs à disposição professores como o meu Orientador, Aníbal Matos, que sempre transmitiu uma palavra de otimismo e confiança em momentos de alguma hesitação minha e que me deu, em cada escolha, liberdade de dar o rumo que achasse mais apropriado a este projeto. Que me ofereceu um laboratório com as condições necessárias para tudo quanto quis experimentar e uma equipa prestável e sempre disponível para tudo quanto precisasse.

A minha gratidão estende-se, num momento anterior ao deste trabalho, a professores como José Fernando Oliveira, cuja humanidade e conhecimento dele fazem um verdadeiro pedagogo. Ao professor Carlos Araújo Sá, pelo exemplo que representa de organização, competência e retidão. A todos os outros professores genuinamente dedicados da FEUP.

Ao sr. José Azevedo, pelas preciosas cirurgias efetuadas em circuitos microscópicos.

Ao meu amigo João, pelas trocas de ideias que fomos tendo sobre todo o tipo de temas ao longo destes anos e, fundamentalmente, por ser um ser humano bom e bem formado.

À Graça e à Margarida, por estes meses de alegria no trabalho, que nem sempre evolui como queremos mas seguramente evoluiria pior sem a presença de cada uma delas. Que o futuro reserve mais colegas desta qualidade.

Aos meus companheiros no projeto WiseNetworks.

A todos os amigos e colegas que transformam o DEEC/FEUP num local aprazível, em particular aqueles que reconheçam ter sido aqui injustamente ignorados. Terão provavelmente razão!

Às minhas referências morais, sociais, culturais e políticas: o meu tio Serafim, que entretanto perdi, o meu primo Luís, o Abel, cada um deles com uma capacidade ímpar de viver a vida com inteligência.

Aos meus pais, em especial à minha mãe, pela paciência férrea e infinita compreensão. Aos meus irmãos, por estes anos de companhia nesta carruagem de montanha russa.

À Rita, pela música e metamorfose que me soube trazer, cambiante de uma existência tímida e sem concerto num Concerto.

Ricardo Neves

“Life is what happens to you while you’re busy making other plans”

John Lennon

Conteúdo

Resumo	i
Abstract	iii
1 Introdução	1
1.1 Motivação	1
1.2 Objetivos	2
1.3 Estrutura da Dissertação	3
2 Revisão de Literatura	5
2.1 Contextualização	5
2.2 Aplicações da Visão na Robótica	13
2.3 Requisitos de um veículo não tripulado	16
2.4 <i>Obstacle Avoidance</i>	16
2.4.1 Bug	17
2.4.2 Bug2	17
2.4.3 Vector Field Histogram	17
2.5 COLREGS	20
2.6 Fatores ambientais	21
2.6.1 Luminosidade	21
2.6.2 Polarização e Reflexão	22
2.7 <i>Matching</i>	24
2.7.1 <i>Block Matching</i>	25
2.7.2 Classificadores de Haar	25
2.7.3 <i>Scale Invariant Feature Transform</i>	25
2.7.4 <i>Optical Flow</i>	26
3 Tecnologia de Visão Artificial e Visão Estereoscópica	27
3.1 Objetivo	27
3.2 Informação de profundidade na visão monocular	27
3.3 Sistemas de Cores	28
3.3.1 RGB	28
3.3.2 HSV	29
3.4 Tecnologias de sensores de imagem	29
3.5 Câmaras	33
3.5.1 Visão monocular	34
3.5.2 Webcams	34
3.5.3 Câmaras Estereoscópicas	36

3.6	Sistemas Computacionais	37
3.7	O modelo de câmara <i>pin-hole</i>	38
3.8	Par estereoscópico	39
3.9	Calibração	39
3.10	Parâmetros Intrínsecos e Extrínsecos	42
3.11	Ferramentas de Calibração	43
3.12	Testes de Calibração	46
4	Sistema de Condução Visual	49
4.1	Visão Stereo	49
4.2	Hardware	50
4.2.1	Sistema computacional Raspberry Pi	50
4.2.2	Câmaras	51
4.2.3	Sensor de Ultrassons	55
4.2.4	Montagem Final	57
4.3	Software	58
4.3.1	Aquisição e Processamento de Imagem	58
4.3.1.1	Video4Linux2	58
4.3.1.2	Extração de Caraterísticas - CVBlobs	60
4.3.1.3	Determinação de <i>thresholds</i>	63
4.3.1.4	Reconversão de mapas - remapeamento de pontos	64
4.3.1.5	Programa de seguimento e deteção de boias	66
4.3.1.6	Target Track	69
4.3.2	Outros algoritmos	71
4.3.2.1	Transformada de Hough	71
4.3.2.2	Filtros de Variância Local	73
4.3.2.3	Matlab - Simulink Support Package for Raspberry Pi Hardware	77
5	Testes Finais	79
6	Conclusões e Trabalho Futuro	85
6.1	Problemas no desenvolvimento	85
6.2	Satisfação dos Objectivos	87
6.3	Trabalho Futuro	88
	Referências	91

Lista de Figuras

1.1	<i>catamaran</i> Zarco [1]	2
2.1	Teleautomaton - corte na vista lateral [2]	6
2.2	Teleautomaton - Esquema de ligações [2]	6
2.3	COMOX [3]	6
2.4	RMS [lockmartin.com]	7
2.5	AN/AQS-20 [lockmartin.com]	7
2.6	Spartan Scout [mindef.gov.sg]	8
2.7	Lizbeth - embarcação, percurso de operação e dispositivo subaquático de sensori- zação [4]	8
2.8	RMMS - Regiões de operação [5]	9
2.9	DrosoBOTS [6]	10
2.10	Water Strider Jumping Robot [7]	10
2.11	Microrobot (a) contra e (b) a favor da corrente; (c) velocidade de transporte . . .	11
2.12	suckerbot - vencedor do AFRON	11
2.13	BOSS - vencedor do DARPA Urban Challenge 2007	12
2.14	(a) - Sistema "sonda e cesto" (b) - Detalhe do algoritmo de detecção [8]	13
2.15	Mapa de obstáculos e planeamento VFH [9]	13
2.16	(a) - Detecção de linha do horizonte (b) - Detecção de boia [10]	14
2.17	Classificação de 3 zonas distintas na imagem (a) com e (b) sem ondulação. [11] .	15
2.18	Comportamento de alg. baseados em reflexão do céu vs variação de cor [12] . .	15
2.19	<i>Obstacle Avoidance</i> - Algoritmo <i>Bug</i>	17
2.20	<i>Obstacle Avoidance</i> - Algoritmo <i>Bug2</i>	18
2.21	<i>Obstacle Avoidance</i> - Algoritmo VFH	19
2.22	<i>Obstacle Avoidance</i> - Algoritmo VFH*	19
2.23	COLREGS - estratégia de (a) ultrapassagem , (b) encontro e (c) cruzamento . . .	21
2.24	COLREGS - Ambiguidades	22
2.25	Detalhe de imagem obtida (a) sem e (b) com filtro polarizador	23
2.26	Sensor CMOS e detalhe da organização dos pixeis	23
2.27	Imagem subaquática de (a) cor e (b) polarização [13]	24
2.28	Block Matching	25
2.29	Exemplos de <i>features</i> de Haar	26
2.30	Fluxo Ótico	26
3.1	Cubo RGB	29
3.2	Sistema de Cores HSV	30
3.3	Estrutura física do píxel	30
3.4	Sensores (a) CCD (b) CMOS	31

3.5	Placa de circuito impresso de sensor (a) CCD (b) CMOS	31
3.6	Modelo de câmara <i>pin-hole</i>	38
3.7	Modelo para visão computacional	38
3.8	Montagem com par de câmaras	39
3.9	Imperfeições físicas das câmaras	40
3.10	Par alinhado	41
3.11	Disparidade vs Profundidade	42
3.12	Relação Disparidade-Profundidade	42
3.13	Matlab - Camera Calibration Toolbox	44
4.1	Diagrama do Raspberry PI utilizado [http://www.raspberrypi.org/]	51
4.2	Sistema Dessincronizado - Exemplo de Sequência de Aquisição (adaptado de [14])	51
4.3	<i>Webcam</i> - Esquema simplificado da estrutura interna	52
4.4	Aquisição de <i>frame</i> VGA - exemplo de diagrama temporal	53
4.5	Par de PS3 EyeCam após modificação	54
4.6	Câmaras Não Sincronizadas - Sinais VSYNC de ambas	54
4.7	Câmaras Sincronizadas - Sinais VSYNC de ambas	54
4.8	Aquisição com par não sincronizado (sequência)	55
4.9	Aquisição com par sincronizado (sequência)	56
4.10	Sensor HCSR04	57
4.11	Montagem e ligações entre <i>hardware</i>	57
4.12	YUV	59
4.13	Deteção de Contorno - Conetividade 8	60
4.14	CvBlobs - Sequência de Exec. no Algoritmo de Det. de Boias	61
4.15	CvBlobs - Resultados Parciais do Algoritmo de Det. de Boias	62
4.16	Objeto com Sombra	62
4.17	Imagens teste e Histogramas	64
4.18	Imagem original	65
4.19	Imagem remapeada	65
4.20	Processo de remapeamento de imagens	65
4.21	Remapeamento de pontos	66
4.22	Resultados da função <i>convertemapas</i> - a) sem e b) com interpolação	67
4.23	Algoritmo de seguimento e deteção de boias	68
4.24	Sequência do algoritmo de seguimento de alvo e deteção de boias	70
4.25	Transformada de <i>Hough</i> (pontos e linhas)	71
4.26	Transformada de <i>Hough</i> (círculos)	72
4.27	Algoritmo baseado em Transformada de <i>Hough</i> (círculos)	73
4.28	T. <i>Hough</i> : Resultados de execução do algoritmo	74
4.29	Aplicação do filtro de variância local (L=1,2,5)	76
4.30	Superfície da Água - Efeito da profundidade na razão saturação/brilho	76
4.31	<i>Simulink Support Package for Raspberry Pi Hardware</i>	77
5.1	<i>Setup</i> Final	79
5.2	Alvo a seguir	80
5.3	Cena de teste (orientação)	80
5.4	Peso computacional	81
5.5	Testes de deteção e seguimento (orientação) - Raspberry PI	82
5.6	Testes de deteção e seguimento (distância) - Raspberry PI	83

5.7	Comunicação Raspberry PI - Embarcação	84
6.1	Bola a cerca de 3m e 5m, respetivamente	85
6.2	Aquisição defeituosa a 640x120 pixeis	85
6.3	Imagem adquirida em (a) Windows (b) V4L2	86
6.4	Pico de Tensão no barramento de 5V do ASV	87
6.5	Raspberry Pi Cam	89

Lista de Tabelas

2.1	Orçamento do DOD 2011/2015 para veículos não tripulados [15]	12
2.2	Iluminâncias características de diversas situações	22
3.1	Caraterísticas de sensores CCD e CMOS	32
3.2	Modelos e caraterísticas de câmaras para visão monocular	34
3.3	Modelos e caraterísticas de <i>webcams</i>	35
3.4	Modelos e caraterísticas de câmaras estereoscópicas	36
3.5	Modelos e caraterísticas de sistemas computacionais de baixo custo	37
3.6	Pares de imagens	47
3.7	Medições com par estereoscópico	48
3.8	Influência da resolução na qualidade da medição	48
4.1	Funções da Biblioteca CvBlobs Utilizadas	63
4.2	Média e Desvio Padrão dos parâmetros Brilho e Saturação	75

Abreviaturas e Símbolos

ASV	<i>Autonomous Surface Vehicle</i>
AIS	<i>Automatic Identification System</i>
<i>fps</i>	frames por segundo
lux	Unidade de luminância (para $\lambda = 550nm$, $1lux = 1lumen/m^2 = 1/683W/m^2$)
FPGA	<i>Field – Programmable Gate Array</i>
OA	<i>Obstacle Avoidance</i>
CA	<i>Collision Avoidance</i>
VO	<i>Velocity Objects</i>
CMOS	<i>Complementary Metal–Oxide–Semiconductor</i>
CCD	<i>Charge-Coupled Device</i>
SLAM	<i>Simultaneous Localization and Mapping</i>
LADAR	<i>Laser Detection and Ranging</i>
V4L2	<i>Video For Linux Two</i>
V	<i>Volt</i>
Vpp	<i>Volt pico-a-pico</i>
GPU	<i>Graphics Processing Unit</i>
UUV	<i>Unmanned Underwater Vehicle</i>
UAV	<i>Unmanned Aerial Vehicle</i>

Capítulo 1

Introdução

Este primeiro capítulo clarifica a motivação e objetivos do trabalho desenvolvido no decurso desta dissertação, fornecendo ainda informação sumária sobre a sua estrutura.

1.1 Motivação

São diversificados os contextos de atuação da robótica móvel.

Inúmeros esforços vêm sendo feitos no sentido de tornar mais autónomas as aplicações, adequando a sua capacidade de resposta a um maior número de situações de complexidade também ela crescente. E não é por acaso que se investe com esta determinação: um robot móvel, robusto e inteligente pode, em cenários difíceis, substituir-se ao homem. Veja-se, a título de exemplo, que tarefas de inspeção em ambientes hostis como, ainda recentemente, na central nuclear de *Fukushima*, podem ser desempenhadas por robots, eliminando prejuízos sérios para a saúde humana.

No contexto português, pensando em particular nos trabalhos que incidem sobre a robótica aquática, toda a investigação e consequentes desenvolvimentos feitos nesta área revestem-se de uma enorme importância.

Portugal detém atualmente uma Zona Económica Exclusiva com mais de 1,7 milhões de km^2 tendo pedido, em Maio de 2009, uma extensão da sua jurisdição. O resultado poderá ser um aumento dessa área para quase 4 milhões de km^2 . Em resultado disto, Portugal passará a deter a 10^a maior ZEE mundial. O mar é, desde há séculos, um designio e uma fonte de riqueza do povo luso pelo que a soberania portuguesa envolve, em primeiro lugar, uma responsabilidade de vigilância. Com efeito, os veículos autónomos aquáticos podem representar um meio de recolha de informação, com efeito quer profilático, quer dissuasor, afigurando-se ao mesmo tempo como possibilitadores de enormes economias face aos meios tradicionalmente empregues nestas tarefas. A robótica autónoma deve, nos próximos anos, ver crescer a sua influência nas ações de patrulhamento de orla costeira, portos, rios e lagos, não sendo também nada despreciando o potencial de desenvolvimento nas aplicações de domínio militar.

Outro fenómeno que se vem multiplicando é a procura de soluções de baixo custo para aplicações robóticas: uma busca por *robotics e low cost* em qualquer base de dados científica gera

milhares de resultados. É isto, também, sinal de que a evolução tecnológica abriu portas e já permite a criação de dispositivos com um elevado grau de confiabilidade, funcionando em teatros de operação muito concretos e tendencialmente resistindo a condições mais agrestes.

Da perspetiva da procura de novas soluções, a produção de informação com um trabalho desta natureza tem, inerente, a vantagem da aplicabilidade múltipla: muita da base tecnológica da robótica aquática, terrestre e aérea é partilhada, sendo possível a reutilização ou fácil adaptação do conhecimento produzido.

1.2 Objetivos

O objetivo principal deste projeto é a implementação de um sistema de sensorização com cujos dados seja possível a um ASV cumprir trajetórias, seguindo referências visuais e desviando-se de obstáculos. Há a previsão de que as referências visuais tenham uma qualquer característica distintiva, que permita aos algoritmos de controlo identificá-las. Essa característica pode ser, por exemplo, uma cor ou forma específica.

Presume-se ainda que o veículo funcionará com luminosidade e em águas pouco agitadas, como as de um tanque ou uma albufeira.

Importa referir que todo o controlo de motores e eletrónica de potência do veículo está já implementado, estando o controlo a criar responsável pela geração de trajetórias.

As características finais do veículo poderão servir de base a que, de futuro, execute missões de patrulhamento de uma dada área, dando nessa situação resposta a situações como a da oclusão do veículo infrator perseguido.

O sistema destina-se aos ASV's Zarco e Gama, *catamarans* do grupo de investigação OceanSys do DEEC/INESC, da FEUP.

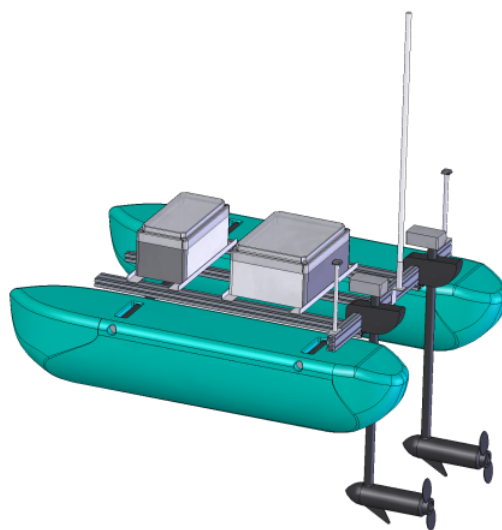


Figura 1.1: *catamaran* Zarco [1]

1.3 Estrutura da Dissertação

Esta dissertação encontra-se organizada em seis capítulos.

O **segundo capítulo** é feita uma revisão da literatura, sendo explorada a evolução da robótica autónoma e algumas suas aplicações no presente. Faz-se uma busca extensa das câmaras disponíveis no mercado e exploram-se algoritmos de processamento de imagem utilizáveis no âmbito do trabalho desenvolvido.

No **terceiro capítulo** é estudado o problema da visão estereoscópica e da calibração de câmaras, descrevendo o primeiro da perspetiva geométrica e, no caso do segundo, dando exemplos de ferramentas e apresentando resultados de testes de calibração do par de câmaras.

No **quarto capítulo** detalha-se o modo de aquisição de imagem, explicando o processo de aquisição de imagem pelo sensor e a adaptação efetuada às *webcams* utilizadas. São também explorados algoritmos de identificação de objetos e feita uma comparação da sua performance em diferentes sistemas computacionais. Aqui são ainda apresentados exemplos da utilização da *toolbox* de *Matlab/Simulink* para interação com Raspberry Pi.

No **quinto capítulo** são apresentados os resultados de testes do sistema implementado.

O **sexto capítulo** está reservado para conclusões, problemas surgidos e sugestões de trabalho futuro.

Capítulo 2

Revisão de Literatura

2.1 Contextualização

Vem de longe a aspiração de criar dispositivos autónomos para o desempenho de tarefas rotineiras. Podemos recuar pelo menos até à Grécia Antiga, ao mito de Talos: robot forjado por Hefesto com a ajuda de Ciclopes, foi confiada a este gigante de bronze a proteção da ilha de Creta. Tinha por função impedir a entrada de estrangeiros e a saída de habitantes sem autorização de Minos, rei de Creta, patrulhando diariamente três vezes o seu perímetro [16].

Já fora do domínio mitológico, um marco importante foi protagonizado em 1898 por Nikola Tesla. Numa piscina interior de Madison Square Garden apresentou, perante uma multidão de nova-iorquinos incrédulos, uma embarcação que podia ser pilotada através de um controlo remoto sem fios. A patente, registada nesse mesmo ano, reforça a ideia de um Tesla visionário:

“... The invention which I have described will prove useful in many ways. Vessels or vehicles of any suitable kind may be used, as life, dispatch, or pilot boats or the like, or for carrying letters, packages, provisions, instruments, objects, or materials of any description, for establishing communication with inaccessible regions and exploring the conditions existing in the same, for killing or capturing whales or other animals of the sea, and for many other scientific, engineering, or commercial purposes; but the greatest value of my invention will result from its effect upon warfare and armaments, for by reason of its certain and unlimited destructiveness it will tend to bring about and maintain permanent peace among nations.” [2].

A descoberta de Tesla podia ser reproduzida para outros fins. Uma das consequências que dela resultou foi a possibilidade de, daí em diante, poder pilotar-se uma embarcação sem um operador a bordo. Nas décadas seguintes, o mundo viu-se envolvido em conflitos bélicos com um grau de sofisticação e uma dimensão nunca antes vistos. Muitas das inovações nos veículos em geral e em sistemas autónomos em particular resultaram e continuam a resultar de necessidades impostas por

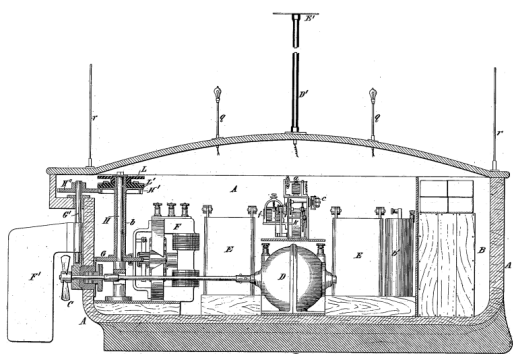


Figura 2.1: Teleautomaton - corte na vista lateral [2]

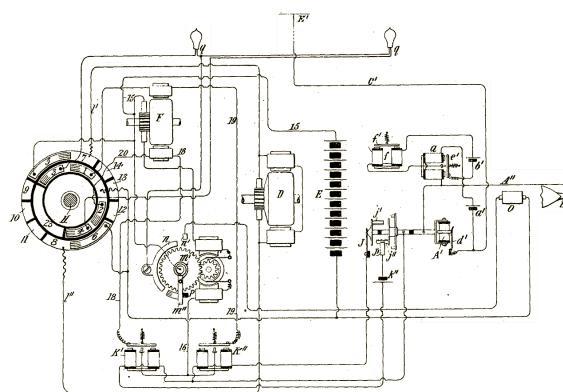


Figura 2.2: Teleautomaton - Esquema de ligações [2]

esses momentos da história. Se é verdade que a primeira guerra mundial representou o advento da aviação de guerra, explorada numa vertente de reconhecimento, defesa e ataque, é-o também que foi principalmente um conflito decorrido nas trincheiras, dominado por combates de artilharia e infantaria. Um quarto de século medeia entre o início de primeira e segunda grandes guerras. Na nova guerra, o potencial de devastação foi amplificado por uma panóplia de evoluções tecnológicas que foram ficando à disposição dos militares. A título de exemplo, a batalha de Okinawa, que envolveu os três ramos das forças armadas no opúsculo do conflito, opondo os EUA ao Japão, dá uma imagem do nível de destruição a que o mundo estava a assistir: 150000 vítimas, 7830 aviões japoneses destruídos, 50 navios afundados e 300 danificados. São números como estes que justificam o aparecimento daquele que é considerado o primeiro veículo não tripulado com algum tipo de autonomia – o “torpedo Comox”. De origem canadiana, o Comox foi a resposta à encomenda de um dispositivo automático que criasse uma cortina de fumo para proteção de tropas e veículos em operações anfíbias, tarefa habitualmente desempenhada por barcos ou aviões [3]. O seu sistema de condução automática tirava partido de um magnetómetro e a sua rota pré-programada era inalterável. Ao contrário dos grandes LCI¹ americanos, esta embarcação foi construída com a



Figura 2.3: COMOX [3]

¹ Os LCI (Landing Craft Infantry) são embarcações de fundo plano que permitem o desembarque rápido de tropas

exigência de ser pequena, rápida e barata.

Embora não tenha chegado a ser utilizado em cenário de batalha, fez-se prova do seu funcionamento em diversos ensaios que, por serem realizados já em 1944 não permitiram os aperfeiçoamentos necessários para utilização naquele conflito.

O fim da guerra não fez estagnar a evolução. Nos anos 50, a marinha americana usou embarcações não tripuladas em zonas de ensaios nucleares, na recolha de amostras de água para análise dos níveis de radioatividade. O constante envolvimento dos EUA em conflitos armados faz com que, na década de 60 sejam usadas no Vietname embarcações daquele género na entrega de munições e na deteção de minas. Outras nações mundiais acabaram por seguir o exemplo e, elas próprias, desenvolver veículos não tripulados para deteção de minas [17].

O RMS, um semi-submersível com propulsão diesel de alta eficiencia, é um exemplo de um ASV da atualidade, usado na deteção de minas. Desenvolvido pela Lockheed Martin, este veículo de 7 metros, capaz de velocidades superiores a 16 nós, cumpre planos de missão pré-programados. Com base em sensores eletro-óticos e acústicos (Figura 2.5), identifica objetos de interesse e reporta essa informação a embarcação base. Segundo o construtor, este veículo é capaz de cobrir uma área 5 vezes mais rápido e com um custo 10 vezes inferior ao dos meios tradicionais.

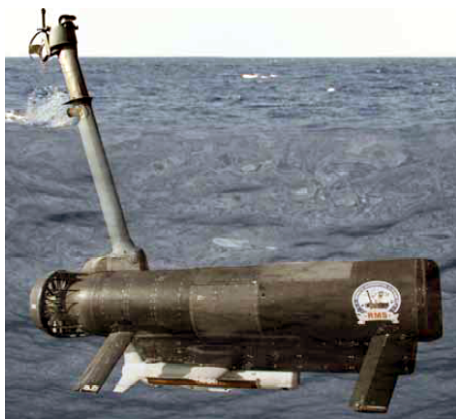


Figura 2.4: RMS [lockmartin.com]

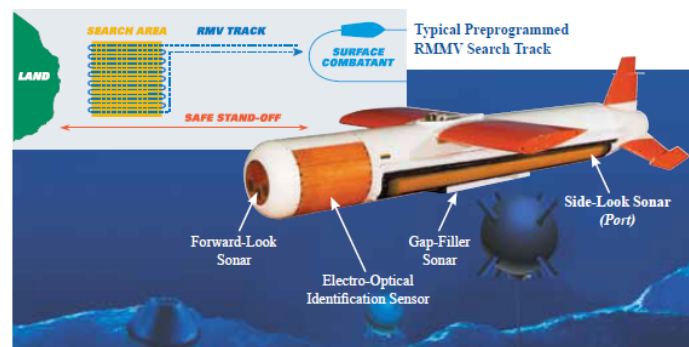


Figura 2.5: AN/AQS-20 [lockmartin.com]

O *Spartan Scout*, um semi-rígido desenvolvido conjuntamente pelos EUA, França e Singapura, é capaz de funcionar em modo autónomo ou parcialmente autónomo, podendo ser operado por controlo remoto. Com 7 metros, atinge velocidades de até 28 nós e suporta até 1500kg. Existe uma versão de maior porte, com 11m e *payload* de 2500kg. Entre outras, funcionou em missões de proteção de forças militares e cenários de ataque de precisão, fornecendo também proteção contra terrorismo e pirataria.

Os últimos 15 anos foram um período fértil no desenvolvimento e utilização, por parte de universidades, empresas e governos, de ASV's dedicados a missões científicas. A justificação é serem pequenos e facilmente manobráveis por apenas uma pessoa, além do crescente leque de sensores disponível e reduzidos custos de operação.

em praias. Podem ser de múltiplos tipos, existindo uma variante que permite lançar uma cortina de fumo que protege tropas e embarcações de ataques aéreos.



Figura 2.6: Spartan Scout [mindef.gov.sg]

A limnologia - estudo das águas interiores - é uma área científica no âmbito da qual surgiram variadas aplicações. Entre as melhorias proporcionadas pela robótica aquática nesta disciplina, está a substituição do método tradicional de recolha pontual de amostras a uma dada profundidade feitas por meio de recipientes, por uma vigilância mais abrangente, espacial e temporalmente, das características bacteriológicas e composicionais das águas lacustres. Exemplo disso, o *Lizbeth* [4] é um catamaran criado para vigiar estas condições no lago Zurique que tira partido de um equipamento de recolha de parâmetros limnológicos. Movendo-se a uma velocidade inferior a 1 m/s, este ASV atravessa o lago em linha reta recolhendo amostras num padrão de zigue-zague em profundidade, entre os 3 e os 20 m. Obtém medidas de temperatura, condutividade, oxigénio dissolvido, PAR (radiação fotossinteticamente ativa) e fluorescência de pigmentos algais. Parâmetros que no passado eram adquiridos num número muito limitado de pontos do lago e extrapolados para a restante área podem agora ser adquiridos com frequência semanal obtendo-se uma descrição mais rica, fruto da recolha de novas amostras ao longo daquele percurso a cada 2 segundos.

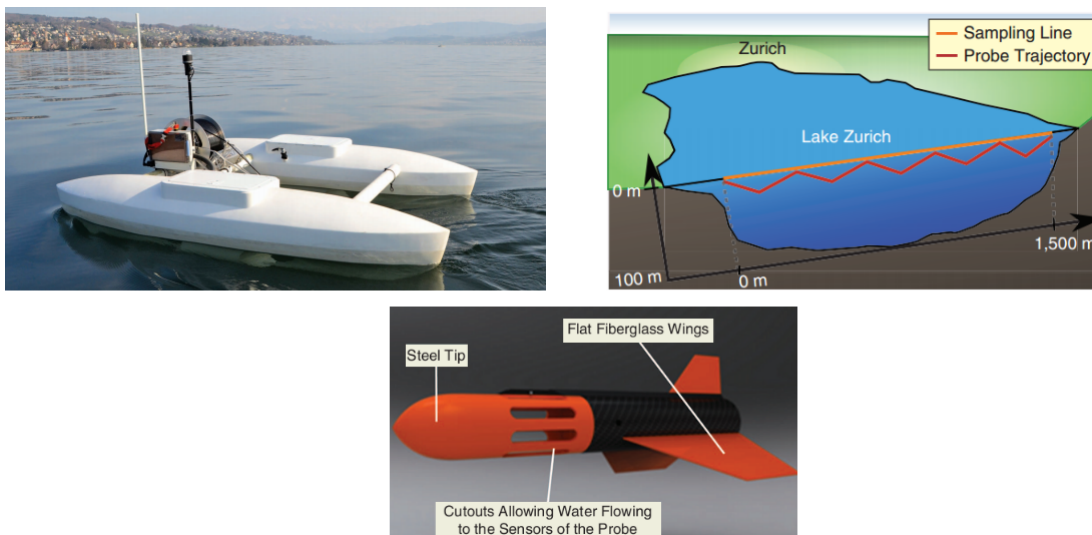


Figura 2.7: Lizbeth - embarcação, percurso de operação e dispositivo subaquático de sensorização [4]

Uma das atuais tendências de investigação nesta área é a colaboração entre veículos autónomos. Sistemas que tiram vantagem das sinergias da operação em conjunto, agregando diferentes tipos de veículos. Em [5], temos disso exemplo: o sistema RDMR (*Risk-aware Mixed-initiative Dynamic Replanning*) tira partido de uma rede de UAV's e UUV's para a execução de missões que envolvem grande variedade de restrições, entre elas, zonas em que os UUV's não podem vir à superfície. Neste sistema, um operador define os limites de operação de cada veículo (velocidades, profundidade/altitude, momentos de operação). Entre eles, os veículos organizam-se, através de uma hierarquia de níveis de planeamento, por forma a executarem as missões eficientemente. Um dado relevante relativamente a este sistema é que são os UAV's a fazer a comunicação entre o operador e os UUV's; para tal, os UUV's têm de emergir. Um ASV num sistema como este pode, em certos casos, revelar-se de enorme utilidade. Funcionando na linha de água, i.e., na interface água/ar, pode servir de elemento retransmissor contínuo entre os dois meios, dispensando a subida à superfície do veículo submarino. Este facto é relevante por vários motivos: por uma questão de manter a camuflagem, porque pode haver vantagem em manter comunicação em zonas de elevado tráfego e também porque, desta forma, o submarino pode continuar a evitar as regras de navegação a que estão sujeitas as embarcações à superfície.

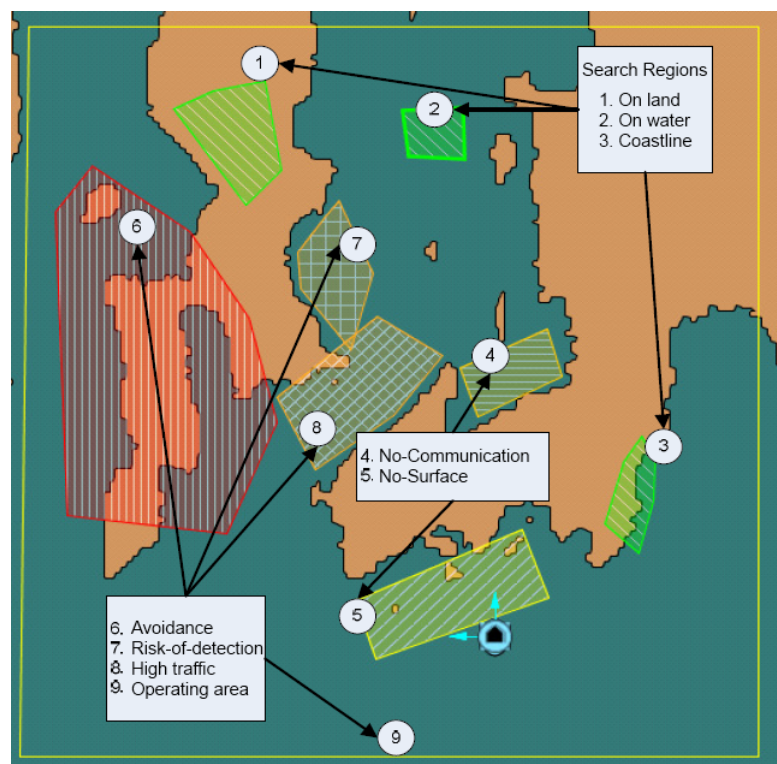


Figura 2.8: RMMS - Regiões de operação [5]

Inspirados em padrões encontrados no mundo animal, surgiram também aplicações que envolvem múltiplos ASV's. Embora o cérebro dos animais seja muito menor do que o dos humanos, em grupo, estes conseguem cumprir tarefas surpreendentes superando por vezes a capacidade de

um humano na realização das mesmas. Fazem-no obedecendo a um conjunto limitado de regras. Os sistemas incorporados no ASV tiram partido de meta-heurísticas que replicam, por exemplo, o comportamento das moscas da fruta (*Drosophila*) na sua busca por comida e parceiro, para organização do grupo e definição eficiente de rotas. É nesta filosofia que assenta o projeto *DrosoBOTS* [6], em que são usados veículos aquáticos circulares com 38cm de diâmetro equipados com capacidade de processamento, GPS, transceiver RF, USB e RS232, Bússola, sensores de temperatura e profundidade. São capazes de se auto-organizar e comunicar entre si e com a estação base a distâncias de 36Km e tiram partido de uma arquitetura de controlo distribuído.



Figura 2.9: DrosoBOTS [6]

Entre outros exemplos de algoritmos daquele género utilizados em aplicações similares estão o ABC (*Artificial Bee Colony Algorithm*) [18], [19], FA (*Firefly Algorithm*) [20] ou o ACO (*Ant Colony Optimization Algorithm*) [21].

Os exemplos anteriores permitem perceber que as realizações da robótica autónoma tomam forma com feitios e escalas muito diferentes. Um grupo de investigadores de uma universidade chinesa desenvolveu, no ultimo ano, um robot aquático [7] com o aspeto de um inseto (alfaiate) capaz de se manter à superfície da água. O pequeno robot pesa 11 g (250 vezes mais que um exemplar animal) mas é capaz de mimetizar os movimentos do animal e manter-se também à superfície graças ao uso de espuma de níquel, um material superhidrofóbico, nas suas pernas e patas. O uso de um micromotor DC e de uma engrenagem redutora permitem-lhe saltos de 14cm de altura e 35 cm de comprimento.



Figura 2.10: Water Strider Jumping Robot [7]

Os tempos mais recentes trouxeram mesmo aplicações de escala sub-milimétrica: é possível controlar, individualmente, via sistemas *wireless*, microrobots capazes de deslocar ou encaminhar micropartículas ou mesmo de funcionalidades de biosensoreamento. Na área da saúde, a aplicação destes desenvolvimentos a terapias convencionais pode vir a substituir uma parte das intervenções invasivas. Em [22], demonstra-se o exemplo de um microrobot a transportar partículas de poliestireno de $5\ \mu\text{m}$ num canal microfluídico de $150\ \mu\text{m}$ de secção. O robot é accionado por um campo magnético controlado através de um programa de Labview com um joystick, e é capaz de se deslocar num fluido em movimento.

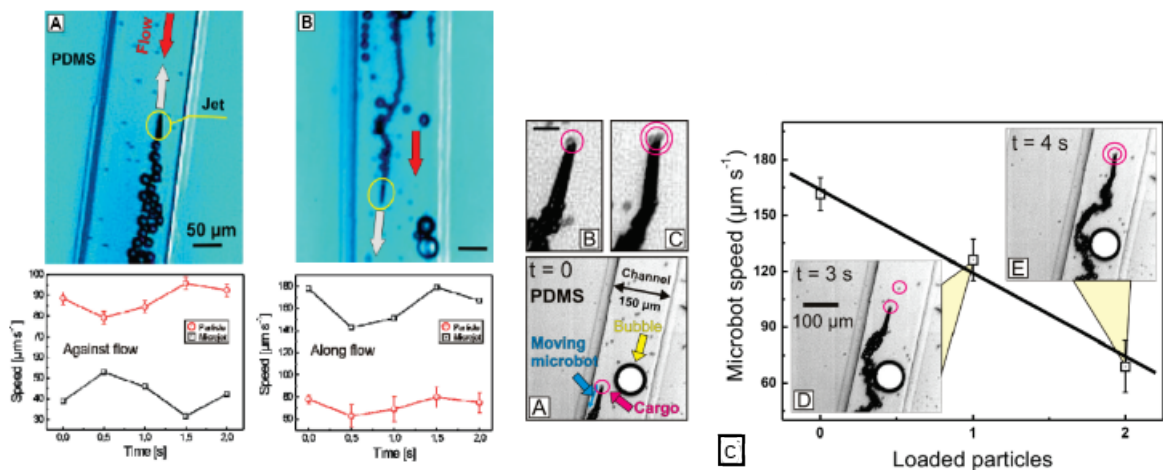


Figura 2.11: Microrobot (a) contra e (b) a favor da corrente; (c) velocidade de transporte

A robótica é também promovida por meio de concursos e desafios internacionais.

Em 2012 realizou-se o *AFRON 10 Dollar Robot Challenge* [23]. Com um orçamento de 10 dólares, estudantes e entusiastas desenvolveram robots capazes de se locomoverem. O objetivo é espalhar o ensino da robótica no continente africano fornecendo ferramentas de estudo e teste muito económicas. O vencedor do concurso construiu um robot funcional por menos de 9 dólares (Figura 2.12)



Figura 2.12: suckerbot - vencedor do AFRON

O *DARPA challenge*, um concurso promovido aperiodicamente pela *Defence Advanced Research Projects Agency* do *DoD* desde 2004, premeia com valores elevados o desenvolvimento de

robots fortemente autônomos. Desde a sua origem que inúmeros veículos de elevada sofisticação, capazes de funcionar autonomamente em cenários de grande complexidade como o deserto ou o ambiente urbano, foram construídos e testados para concorrer nesta competição. A edição de 2012 prolongar-se-á por 27 meses e é dedicada a robots humanoides capazes de executar tarefas em ambientes perigosos.



Figura 2.13: BOSS - vencedor do DARPA Urban Challenge 2007

O DoD, departamento de defesa dos EUA, tem alocados, no plano de investimento para os anos de 2011 a 2015, cerca de 33 mil milhões de dólares para veículos não tripulados. Um quinto deste valor é destinado a investigação e desenvolvimento e metade destinada a contratos e aquisições.

Comparativamente com o plano do mesmo departamento traçado em 2009, a perspectiva de investimento global para o período de cinco anos aumentou 73 por cento, sendo que grande dessa fatia corresponde a contratos e aquisições [15].

Unmanned Funding (\$ Mil)							
Fiscal Year	Defense Prog	FY11	FY12	FY13	FY14	FY15	Total
Air	RDTE	1,106.72	1,255.29	1,539.58	1,440.57	1,296.25	6,638.40
	PROC	3,351.90	2,936.93	3,040.41	3,362.95	3,389.03	16,081.21
	OM	1,596.74	1,631.38	1,469.49	1,577.65	1,825.45	8,100.71
Domain Total		6,055.36	5,823.59	6,049.48	6,381.17	6,510.72	30,820.32
Fiscal Year	Defense Prog	FY11	FY12	FY13	FY14	FY15	Total
Ground	RDTE	0.00	0.00	0.00	0.00	0.00	0.00
	PROC	20.03	26.25	24.07	7.66	0.00	78.01
	OM	207.06	233.58	237.50	241.50	245.96	1,165.60
Domain Total		227.09	259.83	261.57	249.16	245.96	1,243.61
Fiscal Year	Defense Prog	FY11	FY12	FY13	FY14	FY15	Total
Maritime	RDTE	29.69	62.92	65.72	48.60	47.26	254.19
	PROC	11.93	45.45	84.85	108.35	114.33	364.90
	OM	5.79	4.71	3.76	4.00	4.03	22.28
Domain Total		47.41	113.08	154.32	160.94	165.62	641.37
Fiscal Year	Defense Prog	FY11	FY12	FY13	FY14	FY15	Total
All Unmanned	RDTE	1,136.41	1,318.21	1,605.29	1,489.16	1,343.52	6,892.59
	PROC	3,383.86	3,008.63	3,149.32	3,478.96	3,503.36	16,524.12
	OM	1,809.59	1,869.67	1,710.75	1,823.15	2,075.44	9,288.59
Domain Total		6,329.86	6,196.50	6,465.36	6,791.27	6,922.31	32,705.30

Tabela 2.1: Orçamento do DOD 2011/2015 para veículos não tripulados [15]

2.2 Aplicações da Visão na Robótica

Nas diferentes vertentes da robótica, a visão encontra espaço para múltiplas aplicações. No caso dos veículos, a visão pode servi-los diretamente, ajudando à localização e identificação de referências úteis à sua movimentação, podendo também, noutros casos, estar a servir apenas o utilizador, a quem pode interessar, por exemplo, a monitorização de um processo ou uma área.

No campo dos veículos aéreos, encontramos exemplos de ambos. Em [8] foi desenvolvido um sistema de reabastecimento de combustível em veículos aéreos, para ser usado durante o voo em veículos tripulados e não tripulados. Usa visão monocular e, através de uma sequência de aplicações de máscaras, filtros e análise das características geométricas dos objetos detetados, promove o acoplamento correto do sistema de "sonda e cesto"(Figura 2.14).

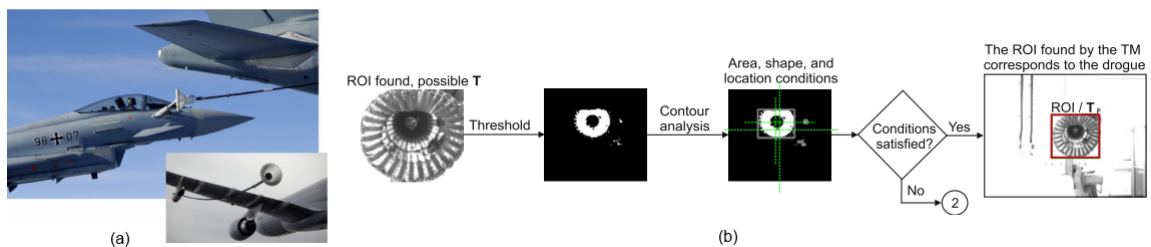


Figura 2.14: (a) - Sistema "sonda e cesto" (b) - Detalhe do algoritmo de detecção [8]

Os autores de [9] desenvolveram um Quadrirotor com capacidade de se movimentar autonomamente em território interior e exterior desconhecido usando três câmaras. O par estereoscópico frontal fornece informação os algoritmos de navegação VFH+ (Figura 2.15) e Bug que correm *on board* e as imagens são enviadas via WIFI para uma estação de terra que corre um processo de SLAM. A câmara orientada para baixo corre um algoritmo de fluxo ótico que ajuda na estimação de pose. Este trabalho demonstra desde logo um ponto que é comum à maioria das aplicações que utilizam imagem:

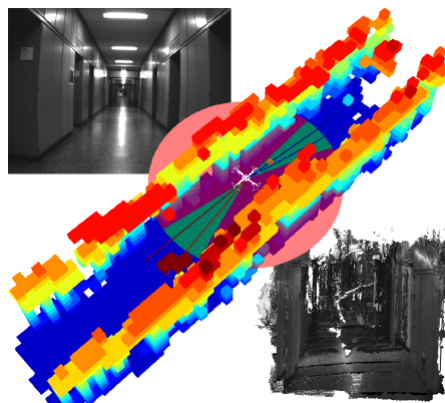


Figura 2.15: Mapa de obstáculos e planeamento VFH [9]

a exigência dos algoritmos obriga a um sistema computacional *on board* poderoso que, neste caso, assenta numa plataforma customizada Intel Core2Duo tendo, a terceira câmara, apontada para baixo, um processador dedicado ARM Cortex-M4F com DSP.

O sistema corre ainda algoritmos de *feature matching* - SURF [24] - e RANSAC [25] para estimação de pose.

Em [10] os autores exploram a possibilidade de incorporar navegação baseada em visão num pequeno veleiro.

Tipicamente as embarcações dispõem de sistemas AIS, usados para conhecer os veículos que se encontram no seu entorno. A lacuna deste tipo de sistema é que apenas deteta embarcações que também o possuam, visto funcionar com base numa troca de mensagens via radio VHF.

O sistema desenvolvido pretende, com base em visão monocular, detetar a linha do horizonte, detetar boias que delimitem trajetos e fazer estimação de fluxo ótico.

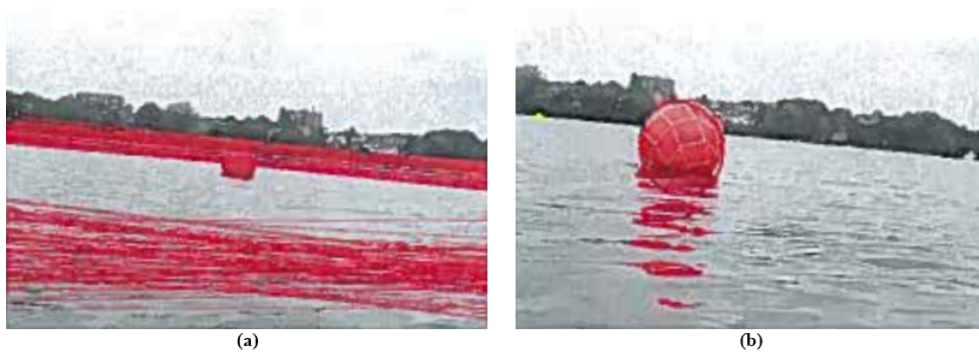


Figura 2.16: (a) - Detecção de linha do horizonte (b) - Detecção de boia [10]

A deteção de linha do horizonte é conseguida após a aplicação de um filtro gaussiano e um detetor de contornos, seguido de um transformada de *Hough* linear. A decisão de classificação tem em conta o comprimento de linha, a sua estabilidade ao longo do tempo e a proximidade ao centro da imagem, que é por norma, a zona em que surge o horizonte. A deteção de boias faz-se após uma conversão do espaço de cores, de RGB para HSV, seguida da utilização de uma sequência de operações morfológicas para limpeza da imagem. A classificação é feita com base em funções de OpenCV e uma elipse envolvente permite a estimação do raio da boia e respetivo centro. O fluxo ótico é usado para determinar a rotação (*Roll*), sendo, para o efeito, considerados todas as *features* detetadas acima da linha do horizonte em cada duas *frames*. Note-se, por exemplo, na (Figura 2.16) (b), que as reflexões da água são identificadas como áreas candidatas a boias, mas o processo de triagem faz com que sejam descartadas.

A necessidade de identificar tipos de terreno é também crucial para certas aplicações autónomas. Em [11] é explorada a possibilidade de distinção de três regiões distintas numa imagem usando diversas tecnologias. O funcionamento durante a noite obriga à utilização de sensores térmicos ou LADAR. No caso de operação durante o dia, é experimentada a utilização de câmaras. A divisão entre céu, água e restante terreno permite desde logo perceber que a média de brilho das regiões classificadas como céu normalmente é saturada, mesmo em situações nubladas. Este valor

médio é sempre superior ao valor de zonas classificadas como restante terreno, estando a média das zonas de água entre ambos. O parâmetro saturação, considerado isoladamente, não parece ser muito relevante. A experiência dos autores mostra também que o classificador é sensível à ondulação: se as águas estiverem paradas, refletindo dessa forma parte do restante terreno, ele classificará parte da região como terreno; em contrapartida, classifica corretamente como água zonas que tenham uma pequena ondulação (Figura 2.17). Sugere-se que a utilização de visão estereoscópica, com a noção de distância e altura relativamente ao par de câmaras é um bom caminho para uma correta reclassificação dessas áreas.

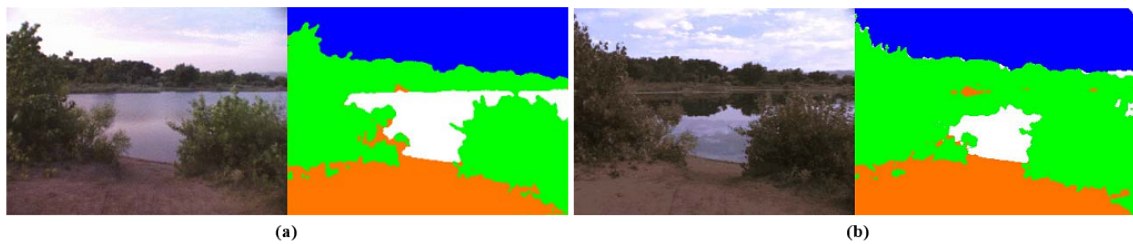


Figura 2.17: Classificação de 3 zonas distintas na imagem (a) com e (b) sem ondulação. [11]

Num documento mais recente [12] dos mesmos autores, o problema da deteção da água baseada na variação de cor é explorado com maior profundidade. Ao contrário do que acontece noutros tipos de terreno, o rácio $\frac{Saturao}{Brilho}$ varia uniformemente ao longo da zona de água, do ponto mais próximo ao mais longínquo. As zonas mais distantes refletem mais intensamente o céu enquanto que nas zonas mais próximas refletem principalmente a cor que a água tem. A partir do comportamento de cada um dos parâmetros dos sistemas de cor HSV e RGB com a variação do ângulo de incidência da luz é possível classificar com mais certeza maiores áreas de água. A Figura 2.18 permite comparar o resultado final deste algoritmo com um algoritmo que apenas classifica água com base na reflexão do céu, numa imagem em que o veículo se encontra próximo da zona de água.

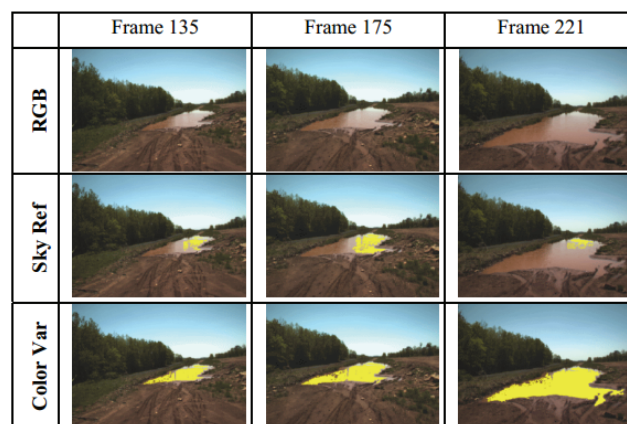


Figura 2.18: Comportamento de alg. baseados em reflexão do céu vs variação de cor [12]

Também aqui, dada a exigência da aplicação, foi utilizado uma máquina Intel Core2Duo 2.6GHz.

2.3 Requisitos de um veículo não tripulado

Os veículos não tripulados podem desempenhar uma grande variedade de missões. Embora com algumas características comuns, as exigências e a importância de cada um desses fatores diferem de caso para caso. A listagem abaixo reúne algumas destas características [26].

- Persistência;
- Baixo custo;
- Discrição e, em alguns casos, invisibilidade;
- Capacidade de detetar, localizar, seguir, identificar e participar objetos de interesse autonomamente;
- Capacidade de recolher, disseminar e agir baseado em vários tipos de informação;
- Capacidade de auto-organização, com plataforma individual e sensores a funcionarem como um todo;
- Manutenção de uma ligação a sistemas hierarquicamente superiores, sob controlo humano;
- Não imposição de riscos ou exigências em demasia aos operadores humanos.

2.4 *Obstacle Avoidance*

Uma das características essenciais a um veículo autónomo é a sua capacidade de evitar obstáculos. Por norma, duas abordagens são seguidas para dar resposta a este problema:

- **deliberativa** - planeamento da trajetória evitando obstáculos fixos ou móveis mais distantes ou remotos como, por exemplo, a linha de costa;
- **reativa** - usada no curto alcance, com obstáculos que surgem na proximidade do veículo. O sistema não conta com eles e é obrigado a utilizar rapidamente estratégias evasivas.

As duas abordagens são frequentemente usadas em conjunto e o objetivo é lidar com obstáculos conhecidos *à priori* (através de processos deliberativos) e outros que surjam e representem um perigo mais imediato (processo reativo) de forma a que esses eventos produzam a menor alteração possível na rota pré-planeada rumo ao destino.

Do ponto de vista da dinâmica do veículo, há um compromisso evidente entre a capacidade de antecipação conseguida pelos processos acima referidos e a velocidade a que pode deslocar-se em segurança.

Os algoritmos mais elementares de deteção de obstáculos têm apenas uma noção do ambiente na vizinhança do robot. O mais elementar destes algoritmos é o *Bug*.

2.4.1 Bug

A execução deste algoritmo processa-se da seguinte forma: partindo dum ponto inicial, o robot segue uma linha que o liga ao destino. Quando aparecem obstáculos, aproxima-se deles e, partindo de um ponto de entrada na vizinhança do obstáculo (**A**) contorna cada um completamente uma vez, para decidir qual é o ponto de saída (**B**) mais próximo do destino. Escolhido o ponto de saída, o robot prossegue o contorno até voltar a encontrar o ponto partindo, no momento em que o descobre, numa linha em direção ao destino. O procedimento é válido para todos os obstáculos com que se cruza. Isto torna-o extremamente ineficiente mas robusto por ser capaz de chegar a qualquer ponto a que o robot tenha acesso.

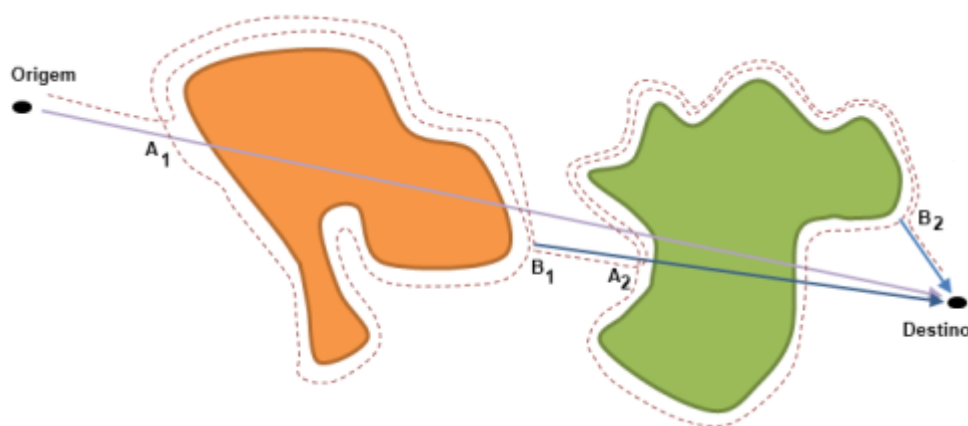


Figura 2.19: Obstacle Avoidance - Algoritmo Bug

2.4.2 Bug2

O Bug2 resulta de uma alteração introduzida neste algoritmo. O robot tenta seguir a linha que une a origem ao ponto de destino. Encontrando um obstáculo, contorna-o até se voltar a cruzar com essa linha seguindo-a novamente, a partir desse momento. Repete este procedimento para todos os obstáculos com que se cruza rumo ao destino.

2.4.3 Vector Field Histogram

Dos mesmos autores do algoritmo *VFF - Virtual Force Field*, resolve alguns dos principais problemas com que se deparam métodos baseados em campos de potencial, por exemplo, oscilação em passagens estreitas. O *VFH* [27] (Figura 2.21) baseia-se num modelo do mundo representado sob forma de uma grelha bidimensional, *2-D histogram grid*. Funciona em três passos:

- **Passo 1** - Cada posição da grelha contém um valor de certeza, representando a possibilidade de existência de um obstáculo. Para cada leitura de um feixe do sensor utilizado, o conteúdo da célula correspondente à distância d é incrementado.

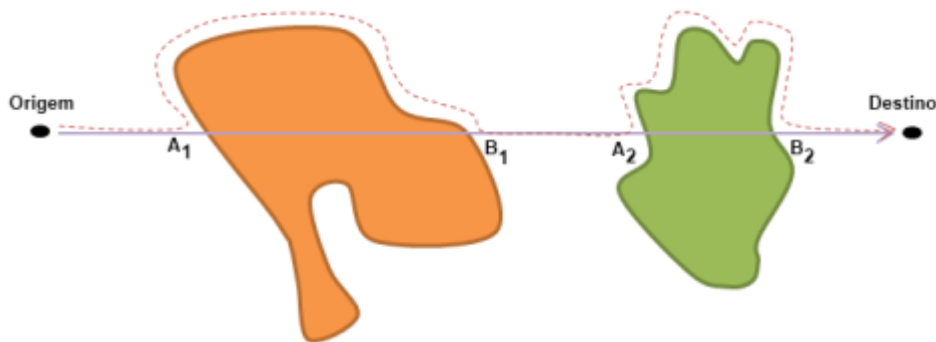


Figura 2.20: *Obstacle Avoidance* - Algoritmo *Bug2*

- **Passo 2** - Considerando apenas uma porção da grelha de histograma na vizinhança do robot, a janela ativa, transforma essa grelha bidimensional num histograma polar unidimensional.
- **Passo 3** - É analisado o histograma e são identificados picos e vales. Impõe-se, no diagrama polar, um *threshold*, definido experimentalmente, para separar orientações com e sem obstáculos. Do conjunto de vales com largura superior à do robot, é escolhido aquele que minimiza uma função de custo. Esta função é influenciada por três fatores: o alinhamento do robot face ao destino, a diferença entre a sua direção atual e a direção do destino e a diferença entre a direção anteriormente tomada pelo robot e a nova direção.

O compromisso é feito no *threshold* que definimos: quanto mais elevado for esse valor, mais provável se torna o robot equacionar uma trajetória arriscada garantindo, em contrapartida, que explorará um maior número de caminhos possíveis.

É um algoritmo que tem maior robustez ao ruído dos sensores porque o histograma criado incorpora informação de várias leituras passadas dos sensores. Em contrapartida, da mesma forma que acontece com algoritmos baseados em campos de potencial, o VFH pode ficar bloqueado em mínimos locais; a estratégia nestes casos passa por utilizar um algoritmo de planeamento global que gere *waypoints* intermédios para o dali retirar o robot.

O mesmo grupo de investigação desenvolveu entretanto versões melhoradas deste algoritmo. O VFH^+ [28] introduz aproximações à cinemática dos robôs, assumindo que descrevem trajetórias curvilíneas. Introduce histerese no *threshold* do histograma e utiliza um novo tipo de histograma, *masked polar histogram*, que gera uma melhor representação dos caminhos verdadeiramente possíveis ao robot, por levar em conta a relação entre a velocidade e as trajetórias possíveis. Não sendo possível seguir nenhuma trajetória com a velocidade atual, o robot redu-la até conseguir prosseguir.

Uma terceira evolução deste algoritmo, o VFH^* [29] explora o algoritmo A^* , um algoritmo de pesquisa de caminhos. Resolve problemas enfrentados pelas versões anteriores (VFH , VFH^+), problemas esses que resultam de naqueles algoritmos ser feito um planeamento de *obstacle avoidance* puramente local. A (Figura 2.22) serve para exemplificar: para as versões anteriores, seguir

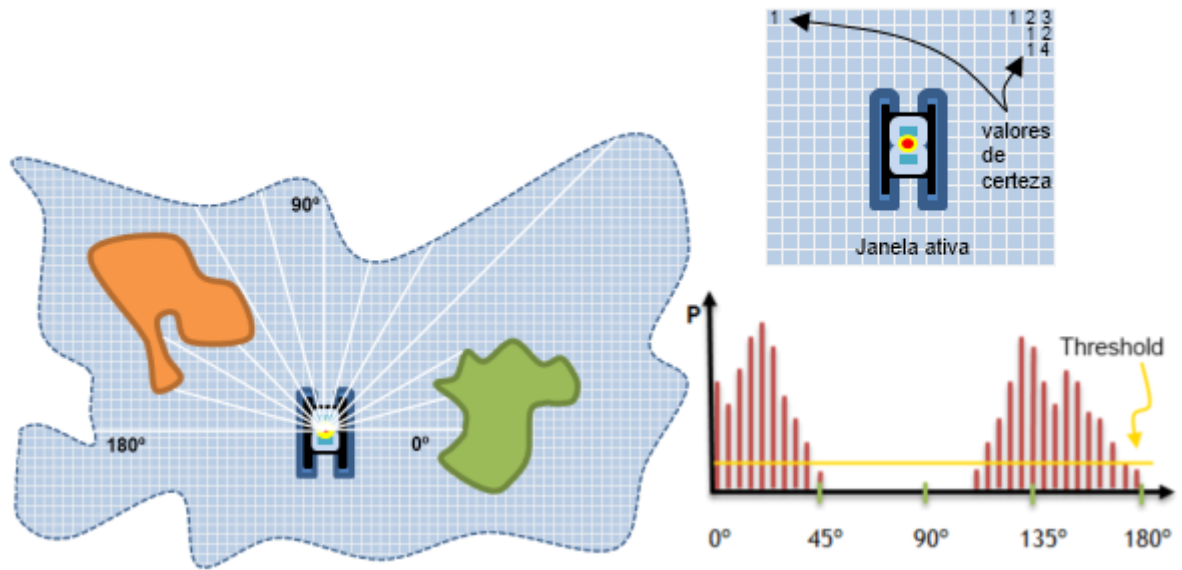


Figura 2.21: *Obstacle Avoidance* - Algoritmo VFH

o caminho **a** ou **b** é indiferente porque a zona coberta pelos sensores - a laranja - identifica apenas o um obstáculo e o cais direito e esquerdo. Um algoritmo como o VFH^+ acertaria apenas metade das vezes.

Para todas as direções possíveis, o VFH^* analisa *à priori* a nova posição e orientação depois de se mover uma unidade de distância. Este processo é repetido para todas as posições projetadas, criando-se uma árvore com a profundidade que for desejada. A escolha do rumo a seguir faz-se, como nos anteriores, com base numa função de custo.

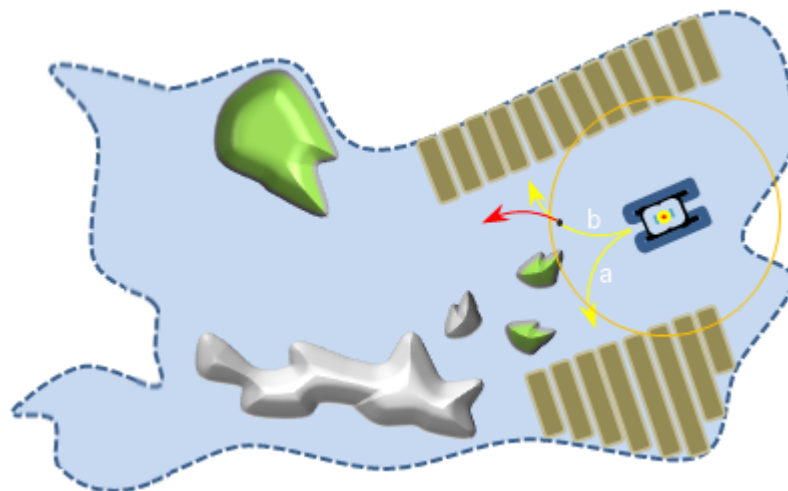


Figura 2.22: *Obstacle Avoidance* - Algoritmo VFH^*

A profundidade da árvore tem relação direta com o alcance do algoritmo, isto é, com a capacidade que o robot terá escolher eficientemente os caminhos em ambientes com mais obstáculos. Existe, contudo, um *tradeoff* entre a profundidade da árvore e o tempo de execução do algoritmo.

2.5 COLREGS

Publicado em 1972 pela *International Maritime Organization* e com a última revisão em 2001, o COLREGS - Regulamento Internacional para Evitar Abalroamentos no Mar - é um conjunto de normas que regula a condução de embarcações. Estão incluídas normas relativas a cruzamento, ultrapassagem e aproximação de veículos bem como informação sobre sinais sonoros e luminosos. O mais comum é encontrar-se, no domínio dos veículos autónomos, sistemas que utilizam apenas algumas das regras de cruzamento, ultrapassagem e aproximação de embarcações. Em ambientes com muito tráfego é essencial que as embarcações obedeçam ao COLREGS; mesmo assim, se uma delas não obedecer é necessária a intervenção de um sistema de *obstacle/collision avoidance*. Do ponto de vista da robótica autónoma, os comportamentos do COLREGS devem ser distinguidos dos gerados por métodos de *collision avoidance* pela sua orientação para "como" as colisões são evitadas, numa perspectiva mais preventiva. As normas COLREGS devem ser parte integrante do algoritmo de navegação. Em [30] são enunciados quatro aspetos que devem motivar o emprego do COLREGS juntamente com mecanismos de CA no planeamento de trajetórias:

- **Mecanismos de CA são insuficientes** - à semelhança de um mau condutor que não acerta noutros veículos por pouco, uma embarcação a comportar-se dessa forma inspira pouca confiança;
- **O sistema de CA deve seguir uma convenção** - quando as trajetórias de duas embarcações se encontram próximas, a estratégia aplicada para evitar o embate deve ser conhecida e partilhada por ambas. Este ponto é ainda de maior importância quando pensamos em cenários de interação de veículos autónomos com outras embarcações;
- **Missões coordenadas com o CA** - é necessário um sistema de controlo que permita um equilíbrio entre os *inputs* do CA e a missão a cumprir pela embarcação;
- **Mapas estáticos são insuficientes** - num cenário mais geral, os obstáculos não são apenas fixos. Nesta âmbito, o trabalho feito no JPL (CALTECH) [31] funde o COLREGS com a noção de *Velocity Objects*. Cada embarcação ou obstáculo é transformado, do ponto de vista do sistema de navegação do ASV, num cone no espaço das velocidades; colisões futuras são evitadas desde que se garanta que o vetor de velocidade do ASV se mantém no exterior do cone de velocidade dos obstáculos. A noção de velocidade associada a cada deteção torna indistinta a ideia de obstáculo e embarcação, acrescentando ainda informação importante para decisão em situações ambíguas, do ponto de vista do COLREGS.

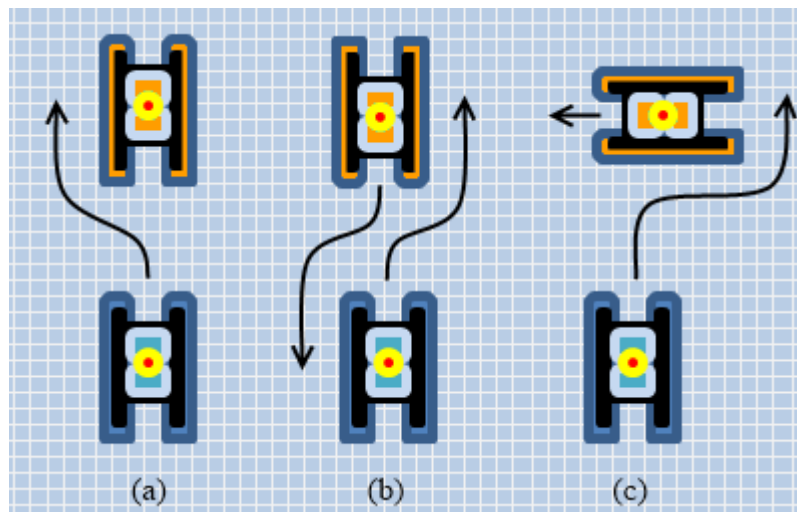


Figura 2.23: COLREGS - estratégia de (a) ultrapassagem , (b) encontro e (c) cruzamento

Por ter sido desenvolvido para ser usado por humanos, o COLREGS é por vezes vago no que respeita a distâncias de segurança ou ângulos a partir dos quais deve aplicar-se. Veículos de grande porte exigem tomadas de decisão com maior folga espacial e temporal do que os pequenos ASV's. Faz parte do desenvolvimento dos sistemas de navegação dos veículos autónomos transformar alguma desta informação qualitativa em métodos que lhes permitam percorrer uma dada sequência de waypoints (alterando-a se necessário) em segurança .

Circunstâncias há ainda, como a da Figura 2.24, em que o sistema tem de decidir se tem ou não necessidade de aplicar as normas do COLREGS. Enquanto que na situação (a) o ASV azul não deve cedência de passagem ao laranja, no caso (b), aplica-se a regra 15 do COLREGS, sendo o azul obrigado a dar passagem ao laranja.

Há ainda situações em que mais do que uma regra se aplica ao mesmo tempo, situações potenciadas pela maior aglomeração de embarcações num mesmo perímetro. Em [30], uma arquitetura multi-objetivo *behaviour based* lida com várias regras em simultâneo; a influência que cada regra vai ter no comportamento da embarcação depende da forma como o ASV está a deslocar-se nesse instante.

2.6 Fatores ambientais

2.6.1 Luminosidade

A luminosidade não é a mesma ao longo do tempo nem espaço. Fatores como clima, proximidade da costa e qualidade do ar afetam-na diretamente. Latitudes próximas do polo norte mantêm, ao longo do dia, iluminâncias médias muito inferiores às observadas nos países do sul da Europa. Além da luminosidade e à semelhança do que acontece com o olho humano, para uma câmara transições bruscas de luminosidade representam um problema que tem de ser tido em conta. Um

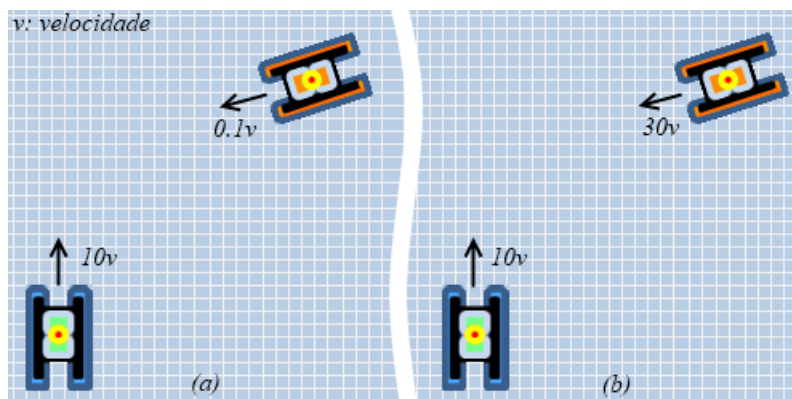


Figura 2.24: COLREGS - Ambiguidades

veículo operando no exterior pode estar sujeito a enormes variações, quer por questões meteorológicas, quer por aspetos orográficos, físicos ou construtivos do ambiente em que se move. A Tabela 2.2 reúne valores aproximados de níveis de iluminância para um conjunto de situações .

Situação	Iluminância (lux)
Luz do Sol (direta)	50000-130000
Luz do Sol (indireta)	10000-20000
Céu encoberto	1000
Local de trabalho (biblioteca, restaurante, escritório)	200-500
Dia muito escuro	100
Iluminação de autoestrada	15
Pôr-do-sol	10
Crepúsculo	1
Noite (Lua cheia)	0.3
Noite (Lua/céu nublado)	0.1
Céu estrelado (normal)	0.001
Céu estrelado (poucas estrelas)	0.00001

Tabela 2.2: Iluminâncias características de diversas situações

2.6.2 Polarização e Reflexão

A movimentação em meio aquático acarreta o problema da reflexão da luz ambiente.

Em cenários de luminosidade intensa, o espelho formado pela linha de água pode criar um encandeamento das câmaras cegando-as momentaneamente. O uso de filtros polarizadores pode minorar este problema embora, por não serem completamente transparentes, acarrete uma redução permanente da luminosidade da cena captada. Tipicamente utilizam-se polarizadores lineares

ou circulares. O efeito de ambos é o mesmo mas, no caso das câmaras com auto-focagem, deve utilizar-se o polarizador circular. O sistema de auto-focagem (AF) é regido por uma parte da luz que entra e é direcionada, através de um espelho semi-transparente *beam-splitter*, para os sensores AF. Se a luz que atinge o espelho ainda estiver polarizada (acontece com um polarizador linear), a quantidade entregue aos sensores AF vai depender da orientação do filtro polarizador. No caso dos polarizadores circulares, a luz é inicialmente polarizada mas atinge o sistema *beam-splitter* já despolarizada pelo que este problema não se manifesta. Os efeitos do uso de um polarizador são visíveis na figura 2.25.



Figura 2.25: Detalhe de imagem obtida (a) sem e (b) com filtro polarizador

Inspirados na navegação egocêntrica dos insetos, os investigadores de [32] desenvolveram um sensor CMOS para navegação que faz a análise da luz polarizada segundo as orientações 0° , 45° e 90° (figura 2.26). Em aplicações exteriores, este tipo de sensores dá pistas sobre a orientação baseado na ideia de que o grau de polarização da luz do céu para uma dada elevação do sol é constante. Esta tecnologia pode, de futuro, com as cada vez menores dimensões dos processos de fabrico e a possibilidade de coexistência de várias tecnologias num único *chip* formando um sistema, permitir a produção de sensores de imagem seletivos, capazes de minorar o prejuízos impostos pelo efeito especular da água.

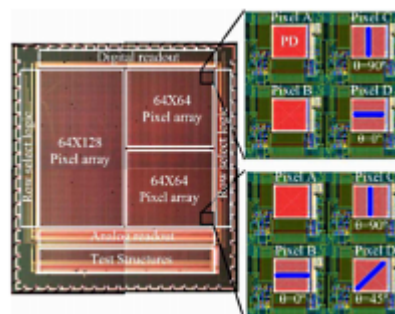


Figura 2.26: Sensor CMOS e detalhe da organização dos pixeis

Cada material tem uma dada assinatura espectral. Empresas como a Fluxdata oferecem polarímetros que permitem ir além do reconhecimento da assinatura espectral de um dado material,

facilitando a recolha de imagens polarimétricas complementares às imagens com informação de cor. Trata-se de equipamentos com 3 sensores CCD com resoluções típicas de uma câmara, cada um com um filtro polarizador. Os filtros têm orientações de 0° , 45° e 90° (à semelhança do caso referido acima). A escolha destas orientações, e podiam ser outras, está explicada em [13] e prende-se com a necessidade de medir o grau de polarização parcial da luz; a imagem de polarização resulta de se ignorar, em cada pixel, a componente não polarizada da luz, preenchendo-o com a cor correspondente à componente polarizada. A Figura 2.27 ilustra a vantagem da utilização de mais esta informação na deteção de objetos em condições de fraca visibilidade.

Aplicações tirando partido de imagens de polarização incluem, entre outros, identificação, reconhecimento e discriminação de objetos, segmentação de imagem [33] ou reconstrução de superfícies com transparências [34]. Recentemente surgiram ainda aplicações no âmbito da estereoscopia. Em [35] testam-se com sucesso vários métodos de *local matching* em imagens de polarização, isto é, de estabelecimento de correspondência entre objetos em zonas das duas imagens. A validade dos resultados é aferida através da sua comparação com informação *ground truth*, proveniente de uma identificação manual de cada um dos pontos de interesse. Neste sistema, além da calibração do par estereoscópico é necessária uma calibração fotométrica.

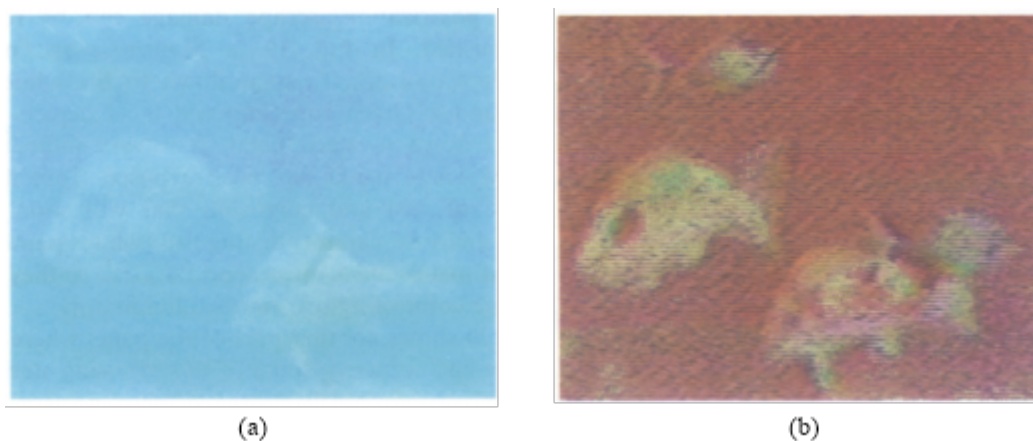


Figura 2.27: Imagem subaquática de (a) cor e (b) polarização [13]

2.7 Matching

O processo de correspondência ou *matching* é aquele através do qual se identifica a projeção de um mesmo ponto do espaço na imagem de cada câmara. No campo da visão estereoscópica, os métodos de *matching* tentam encontrar correspondências entre pixels de ambas as imagens. Estes métodos são tipicamente divididos em duas grandes categorias:

- **Métodos de correspondência local**, que analisam pontos na vizinhança do píxel de interesse;

- **Métodos de correspondência global**, que analisam a imagem completa ou grandes áreas da mesma;

Segundo [36], os métodos de correspondência local podem ser divididos em três grandes categorias: *Block Matching*, *Gradient-Based Optimization / Optical Flow* e *Feature Matching*.

2.7.1 Block Matching

Na descrição sumária de [36] dos métodos de *Block Matching*, refere-se que "procuram a máxima correspondência ou um mínimo de erro sobre uma pequena região". A ideia destas estratégias passa por comparar, por exemplo, entre frames, um dado bloco de uma imagem com vários blocos da imagem seguinte, dentro de uma janela de pesquisa 2.28.

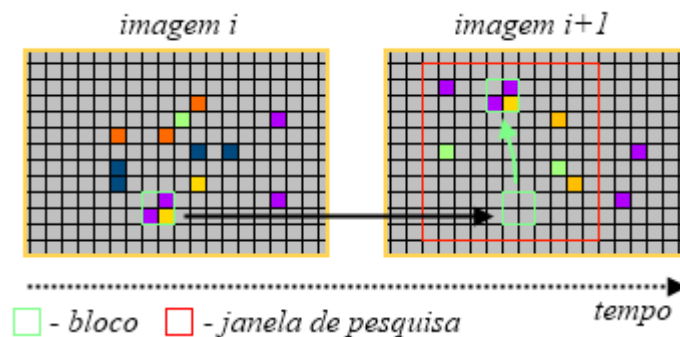


Figura 2.28: Block Matching

2.7.2 Classificadores de Haar

O classificador de Haar é utilizado para detecção de objetos e, em vez de utilizar a intensidade dos pixels como característica diferenciadora, utiliza a variância do contraste entre secções de imagem retangulares e adjacentes. A informação de contraste é utilizada para definir, dentro dessa área retangular, zonas mais claras e mais escuras. Os descritores de Haar são normalmente formados por várias (2 ou 3) destas secções (figura 2.29). Este método exige treino usando uma grande quantidade de imagens, parte delas em que o objeto apareça (positivos) e outra em que não esteja (negativos). É comum utilizar-se o método para identificar pessoas; nestes casos, um treino eficaz exige tipicamente vários milhares de imagens de com pessoas de diferentes características. Os classificadores são também facilmente escaláveis, para lidar com objetos de diferentes tamanhos.

2.7.3 Scale Invariant Feature Transform

O SIFT é um método que faz extração de características distintivas e invariantes em imagens em que estejam presentes objetos de interesse. Estes descritores não variam com o tamanho nem com a rotação dos objetos e são resilientes a mudanças de perspectiva. Por norma, os descritores são muito distintivos, identificando com grande probabilidade mesmo objetos de pequena dimensão

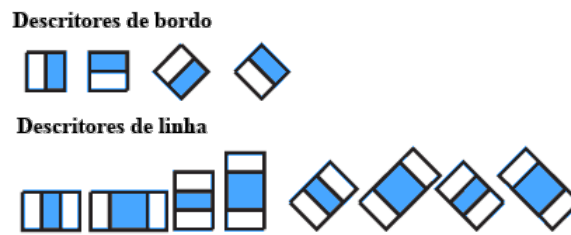


Figura 2.29: Exemplos de *features* de Haar

em cenários densamente povoados. Em [37] encontra-se uma descrição completa do funcionamento desta técnica, com explicação de todos os passos a seguir na implementação.

2.7.4 Optical Flow

O fluxo ótico é o movimento aparente dos padrões de brilho numa imagem. O primeiro princípio em que assenta a maioria dos algoritmos de fluxo ótico é a ideia de que quando um pixel flui de uma zona da imagem para uma nova localização, a sua intensidade de cor não varia. Assume-se, entre outras coisas, que a iluminação da cena é uniforme [38]. Do ponto de vista da visão monocular, se atentarmos à zona verde da imagem 2.30, este pressuposto pode ser descrito por:

$$I_E(x, y, t) = I_E(x + u, y + v, t + 1) ,$$

em que $I_E(x, y, t)$ é a intensidade do pixel de coordenadas (x, y) no instante t .

Uma comparação detalhada da *performance* de diversos métodos de fluxo ótico pode ser encontrada em [39] e, mais recentemente, em [40]

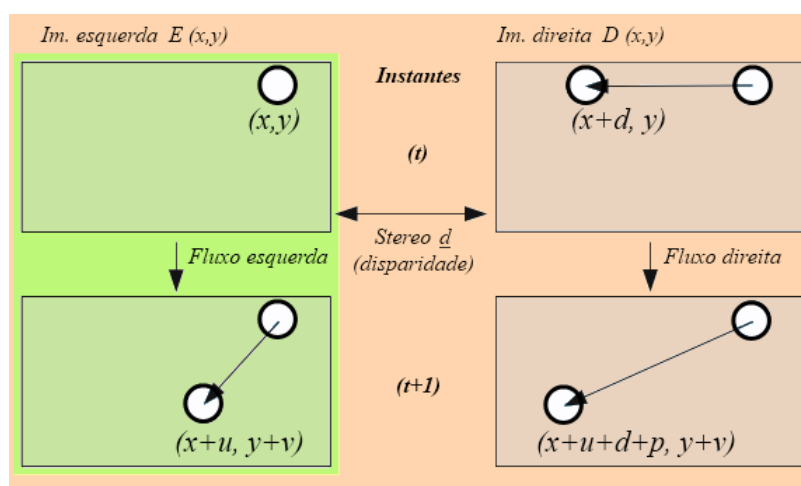


Figura 2.30: Fluxo Ótico

Capítulo 3

Tecnologia de Visão Artificial e Visão Estereoscópica

Este capítulo introduz e descreve o conceito de visão estereoscópica. Partindo da visão monocular, com a apresentação de um modelo de câmara, avança introduzindo visão baseada em pares de câmaras, esclarecendo relações geométricas entre câmaras. No final é discutido o problema da calibração.

3.1 Objetivo

A visão estereoscópica tenta obter informação tridimensional do mundo tendo tirado partido duas ou mais imagens bidimensionais.

3.2 Informação de profundidade na visão monocular

A imagem bidimensional fornece, por si só, informação variada sobre a profundidade dos objetos numa cena. São pistas relevantes, que devem utilizar-se em combinação com a visão estereoscópica, não só por serem um acréscimo de informação, mas também como forma de validação. Em [41] estão listadas várias destas pistas, das quais se destacam:

- **Luz e Brilho** — dependem da orientação do objeto face às fontes luminosas, contribuindo para a noção de profundidade;
- **Tamanho Relativo** — Um mesmo objeto, encontrando-se mais longe parece mais pequeno do que quando está mais próximo. Numa imagem bidimensional, terá obrigatoriamente uma área menor quando se encontra mais distante. Por recurso à memória, o ser humano

consegue decidir sobre a distância a que se encontram objetos que não conhece. Ao contrário do que acontece com os humanos, para uma máquina, o processo de determinação da profundidade de um objeto é lento e exigente;

- **Interposição** — Se um objeto obstrui outro que conseguimos ver parcialmente, podemos concluir que o primeiro se encontra mais próximo de nós;
- **Textura** — Em objetos texturados, temos noção da sua proximidade através do detalhe com que conseguimos perceber a sua textura. Com o aumento da distância, a textura torna-se menos perceptível;
- **Paralaxe de Movimento** - Numa viagem de carro, os postes de iluminação parecem mover-se muito mais depressa do que as montanhas que estão ao longe.

3.3 Sistemas de Cores

A escolha do melhor sistema ou espectro de cores para o processo de identificação de objetos em imagens é da maior relevância. Em [42] encontra-se um estudo extensivo de sistemas de cores e métodos de segmentação que elenca vantagens e desvantagens de cada um deles. Alguns pormenores relevantes de dois dos mais importantes sistemas de cores são sumariamente descritos de seguida.

3.3.1 RGB

O RGB é o sistema de cores mais comumente utilizado em monitores ou câmaras e replica os recetores do olho humano.

É um sistema de cores aditivo, isto é, cada cor resulta do somatório das componentes vermelha, verde e azul que a compõem. No âmbito computacional, o mais comum é que cada canal tenha 8 bits sendo representado por um valor entre 0 e 255. Neste sistema, o branco resulta da soma dos valores máximos de cada componente, o preto da soma dos mínimos e um nível de cinza dos outros casos em que as três componentes tomam o mesmo valor. No RGB, as informações de cor (*chroma*) e intensidade (*luma*) encontram-se misturadas. Desta forma, se for desejado mais brilho numa cor, todas as componentes devem ser escaladas do mesmo valor; se desejarmos uma cor com saturação máxima, sabemos que ela se vai encontrar numa das faces visíveis do cubo RGB (ver Figura 3.1).

Em aplicações de robótica móvel, em que se pretende fazer identificação de objetos e zonas de interesse em imagens, o sistema RGB revela-se ineficaz na presença de regiões de sombra provocadas por variações na iluminação. A alta correlação entre as três componentes faz com que uma variação no brilho (também chamado de luminância ou intensidade) provoque variação idêntica nos três canais. Em [42] é também referida a problemática da avaliação da semelhança entre duas cores visto que o espaço RGB não representa as cores numa escala uniforme: duas cores

iguais com intensidades diferentes, por exemplo azul, $(0,0,10)$ e $(0,0,240)$ estão mais distantes do que duas diferentes, por exemplo, verde $(0,50,0)$ e vermelho $(60,0,0)$.

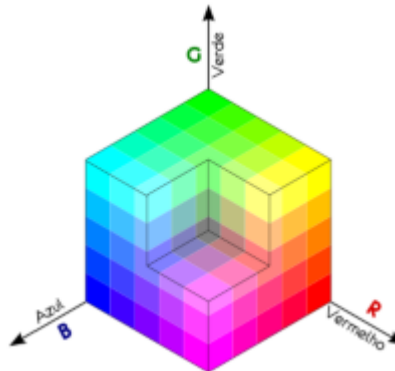


Figura 3.1: Cubo RGB

3.3.2 HSV

O sistema HSV separa cada cor em matiz (H - *Hue*), saturação (S - *Saturation*) e brilho (V - *Value*) conforme apresentado na Figura 3.2. Aqui, a informação de cor e intensidade estão separadas: a de cor é representada pelos valores de H e S e a de intensidade pelo canal V . A matiz representa as cores elementares e é determinada pelo comprimento de onda dominante. A saturação é uma medida da pureza da cor e contém informação da quantidade de luz branca misturada com a matiz. O brilho refere-se à quantidade de luz. A matiz varia entre 0 e 360° enquanto a saturação e o brilho variam entre 0 e 100% . No caso particular do OpenCV, a matiz varia entre 0 e 180 para que seja possível utilizar uma variável de 8 bits, e saturação e brilho estão escaladas para $0-255$. O grande interesse deste sistema é o facto de a matiz ser praticamente invariante face ao nível de iluminação, o que o torna extremamente útil na separação de objetos de cores diferentes. Uma desvantagem decorrente da transformação não linear feita a partir do modelo RGB é a existência de uma singularidade junto ao eixo do cilindro que faz com que valores de H muito do centro sejam numericamente instáveis. Quando $S=0$, H não está definido e se $V=0$, S é indefinido.

3.4 Tecnologias de sensores de imagem

Se em tempos houve uma resposta clara à pergunta "*Qual a melhor tecnologia em sensores de imagem?*", na atualidade as diferenças entre as duas tecnologias dominantes já não permite resposta imediata.

Vindas dos anos 70, durante muito tempo uma delas - o CCD (*Charged-coupled device*) - foi o referencial de qualidade de imagem. Tradicionalmente, o sensor CCD estava disponível com maiores resoluções, pixels de menor dimensão e um menor nível de ruído do que a tecnologia alternativa, o CMOS (*Complementary Metal-Oxide-Semiconductor*). Sendo embora ambos

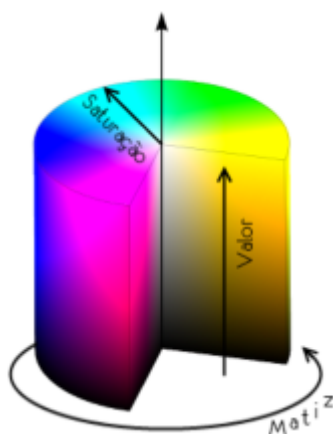


Figura 3.2: Sistema de Cores HSV

contemporâneos, o sensor CMOS não suscitou grande interesse até aos anos 90 porque, quando surgiu, os processos de fabrico limitavam a sua performance. No presente, o sensor CMOS é objeto de intensa investigação e desenvolvimento impelidos pelas necessidades de muita eletrónica de consumo. O baixo consumo, a capacidade de *windowing*, isto é, de ler apenas partes da area do sensor permitindo assim maior *framerate* e possibilidade de zoom digital, *pan* e *tilt*, o menor custo de produção (porque pode ser produzido numa linha convencional e testado ainda na *wafer*).

Em qualquer dos dois casos, a capacidade de reproduzir cor provém da utilização de um filtro que restringe a gama de comprimentos de onda a entrar num dado pixel (Figura 3.3). Sem ele, o resultado da aquisição seria uma imagem monocromática de maior resolução efetiva, sem os processos de interpolação normalmente envolvidos na aquisição a cores.

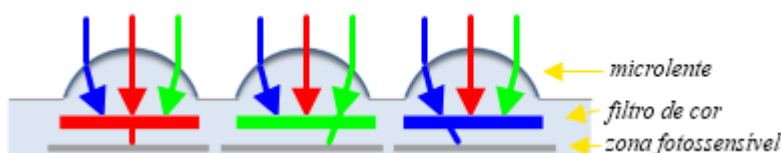


Figura 3.3: Estrutura física do píxel

A topologia de ambos os tipos de sensores tem relação direta com algumas das vantagens e defeitos de cada um deles. Num sensor CCD, cada pixel pode ser visto como um condensador de tecnologia MOS que converte fotões em energia eléctrica, armazenando-a até ser transferida sob forma de sinal analógico, normalmente através de um canal para um único amplificador de saída. Pelo facto de partilharem um mesmo amplificador e subsequente ligação a um conversor ADC (Figura 3.4 a)), no sensor CCD a imagem tem tradicionalmente uma maior uniformidade e um menor ruído *fix pattern* (FPN); em contraponto, o circuito externo exigido para uma câmara será necessariamente mais complexo do que no caso de uma câmara de tecnologia CMOS de prestação semelhante (ver Figura 3.5).

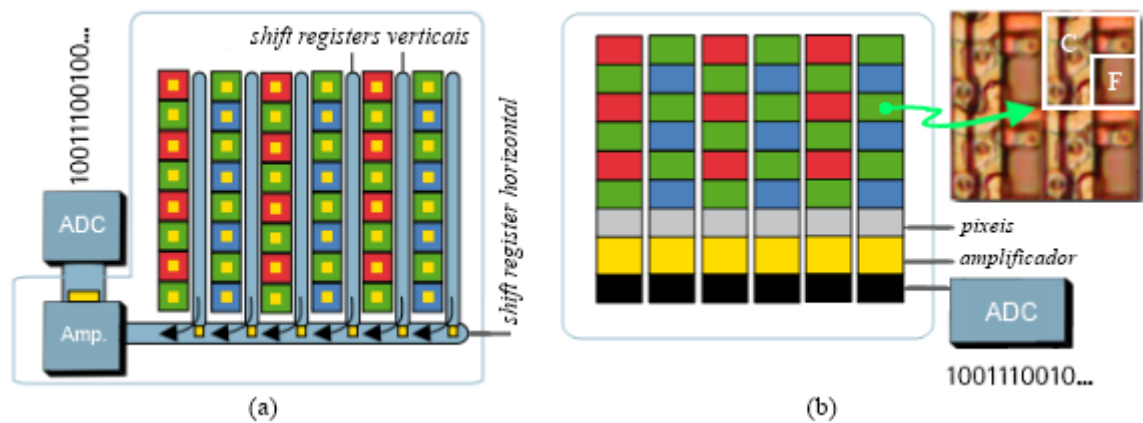


Figura 3.4: Sensores (a) CCD (b) CMOS

No caso do sensor CMOS, cada pixel é constituído por um fotodíodo, um condensador e vários transístores. Esta estrutura produz uma conversão píxel a píxel que é afetada de FPN, porque os componentes de dois píxeis não são exatamente iguais. No detalhe de um píxel da Figura 3.4 b), a letra **F** identifica o fototransístor e a letra **C** o circuito auxiliar embutido no próprio *chip*. Um dos problemas originalmente encontrado nesta tecnologia, a menor sensibilidade e fraca prestação em condições de baixa luminosidade, era devido à área ocupada por este circuito auxiliar: o *fillfactor* ou área sensível de um sensor CMOS varia entre 30 e 70% enquanto o dos circuitos CCD é de 100%; isto, combinado com o facto de o píxel ser maior, garante ao CCD a entrada de maior quantidade de luz e, em consequência, maior sensibilidade.

Muito embora o baixo consumo seja um dos argumentos para a escolha de sensores CMOS, isto é verdadeiro para equipamentos de baixo custo, tipicamente alimentados pelo cabo de dados. O mesmo pode não acontecer em sistemas de alta qualidade e velocidade CMOS, que habitualmente consomem mais que as congéneres CCD. A questão da velocidade é também uma vantagem do CMOS: por utilizarem píxeis ativos e terem todo o circuito auxiliar dentro do chip, os atrasos de propagação são menores.

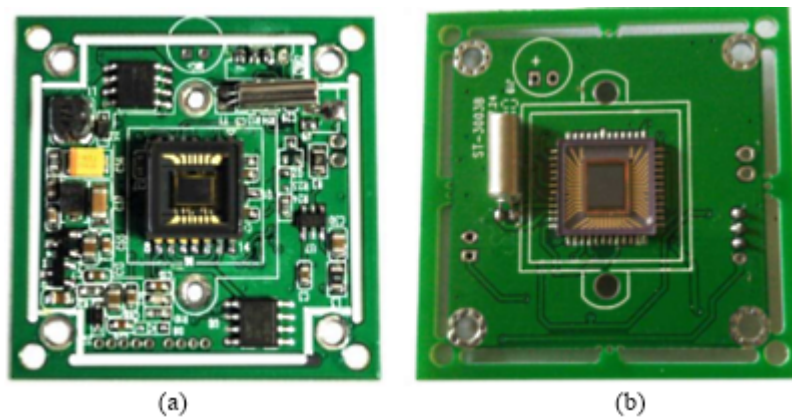


Figura 3.5: Placa de circuito impresso de sensor (a) CCD (b) CMOS

Atualmente, fabricantes da tecnologia CCD batem-se pela redução de consumos, que podem superar em 100x os de um sensor CMOS equivalente. Em contrapartida, para os CMOS a busca faz-se no sentido de uma uniformização e melhoria global da qualidade da imagem. Na busca efetuada, percebe-se que os sensores do tipo CMOS estão a ganhar quota de mercado e a aceder a nichos de mercado anteriormente reservados apenas a câmaras CCD. Na área científica, por exemplo na astronomia ou na medicina, continua a notar-se uma preferência por sensores CCD, por alguns dos motivos referidos acima. Aplicações com requisitos mais apertados de velocidade de aquisição recorrem a tecnologia CMOS.

A atualidade trouxe-nos sensores com resoluções impensáveis ainda há poucos anos: em 2010, a Canon anunciou a criação de um sensor CMOS de 120 MP e em 2011, a PhaseOne lançou a IQ180, com um sensor CCD de 80MP. Já em 2012, a Nokia apresentou o *smartphone* Pureview 808, equipado com um sensor CMOS de 41 MP.

A Tabela 3.1, com informação recolhida em [43], reúne, comparando, algumas propriedades importantes dos dois tipos de sensores. Mais pormenores sobre cada uma das tecnologias podem ser encontrados num estudo disponível em [44].

Caraterística	CCD	CMOS
Sinal à saída do pixel	<i>Pack</i> de Eletrões (Carga)	Tensão
Sinal à saída do chip	Tensão (analogico)	Bits (digital)
Sinal à saída da câmara	Bits (digital)	Bits (digital)
<i>Fill factor</i>	Elevado	Moderado
Inconsistências (amplificador)	Não aplicável	Moderado
Ruído do sistema	Baixo	Moderado
Complexidade do sistema	Elevada	Baixa
Complexidade do sensor	Baixa	Elevada
Componentes básicos de câmara	Sensor + múltiplos componentes / eletrónica de suporte+ lente	Sensor + lente (hardware extra necessário para imagem de maior qualidade)
Custo comparativo de I&D	Mais baixo	Mais elevado
Custo relativo do sistema	Dependente da aplicação	
Performance	CCD	CMOS
Responsividade	Moderada	Ligeiramente melhor
Gama dinâmica	Elevada	Moderada
Uniformidade (resposta entre pixels)	Elevada	Baixa a Moderada
Uniformidade (tempo de exposição)	Elevado	Baixo
Velocidade	Moderada a Elevada	Mais elevada
<i>Windowing</i>	Limitado	Extensivo

Tabela 3.1: Caraterísticas de sensores CCD e CMOS

3.5 Câmaras

A escolha de uma câmara, os parágrafos anteriores permitem percebê-lo, não está limitada à resolução do sensor. Disponibilidade, preço, parâmetros tecnológicos como o tipo de sensor, campo de visão e qualidade da lente, dimensões e geometria, compatibilidade com ambientes de desenvolvimento são, entre outros, fatores a ter em conta no momento da escolha. Algumas câmaras, com interfaces menos comuns, obrigam à aquisição de *hardware* acessório para o processo de aquisição. *Framegrabbers* analógicos e digitais são também comuns, economizando recursos computacionais ao sistema principal por via de um pré-processamento capaz de extrair características de interesse da imagem ou simplesmente tratar do processo de sincronização de câmaras. Empresas como a *Advanced Micro Peripherals* disponibilizam uma grande variedade de equipamentos deste tipo, importando salientar a gama multicanal desenvolvida para o fator de forma PC/104. Sistemas baseados em tecnologia FPGA são também escolha possível para lidar com problemas de sincronização entre câmaras: a XESS fornece o XuLA-200 por um valor muito baixo (\$55) e é possível encontrar *online* exemplos funcionais de adaptação de pares de câmaras PCB.

Uma nova geração de *smartcams*, de fornecedores como a *TeleDyne* ou a *National Instruments*, representa um tipo de sistema embebido com grande capacidade de processamento e pacotes de software associados que lhe permitem, *off-the-shelf*, o processamento, tratamento, armazenamento e extração de características de imagens.

Empresas como a *TeleDyne*, a *The Imaging Source* ou a *Point Grey*, oferecem uma gama muito vasta de câmaras convencionais ou estereoscópicas destinadas a aplicações científicas ou industriais. As Tabelas 3.2, 3.3 e 3.4 resultam de uma pesquisa que, não cobrindo completamente a oferta disponível - por ser impossível - dá uma imagem de uma parte relevante do mercado das câmaras da atualidade.

3.5.1 Visão monocular


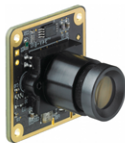

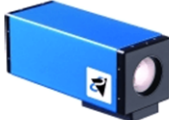



	Modelo	Caraterísticas	
The Imaging Source	Séries DMK e DFK (219-820 eur)	78 modelos distintos Sensores: CMOS, CCD, mono ou cores Res: 320x240 – 2592x1944 pixels @ 6-150 fps Interface: USB, IEEE-1394, GigE Focagem: Manual Trigger: com ou sem FOV: 19 modelos de lente disponíveis (não incluídos na câmara) Consumo: 1.25 a 5 W	
	Séries DMM,DFM (board) (179-480 eur)	23 modelos distintos Sensores: CMOS, CCD, mono ou cores Res: 320x240 – 2592x1944 pixels @ 6-150 fps Interface: USB, IEEE-1394 Focagem: Manual Trigger: com ou sem FOV: 38 modelos de lente disponíveis (não incluídos na câmara) Consumo: 1.25 a 2.5 W	
	Séries DMK e DFK (Autofocus) (319-399 eur)	4 modelos distintos Sensores: CMOS, mono ou cores Res: 320x240 – 2592x1944 pixels @ 6-150 fps Interface: USB Focagem: Auto / Manual Trigger: sem FOV: 38 modelos de lente disponíveis (não incluídos na câmara) Consumo: 1.25 W	
	Séries DMK e DFK (Zoom) (570-830 eur)	12 modelos distintos Sensores: CCD, mono ou cores Res: 640 x 480, 1024 x 768 pixels @ 30 – 60 fps Interface: IEEE-1394 Focagem: Auto / Manual Iluminância min.(mono/cores): 0.03/0.1 lux Trigger: com ou sem Consumo: 2.5 a 5 W	
PointGrey	Grasshopper 3	Sensores: CCD, mono ou cores Res: 1928 x 1448 – 3376 x 2704 pixels @ 9-26 fps Interface: USB 3.0 Trigger: sim (externo e software)	
	Flea 3	Sensores: CMOS, mono ou cores Res: 1328x1048 - 4096x2160 pixels @ 21-150 fps Interface: USB 3.0 Trigger: sim (externo e software) Consumo: < 3W	
CCTV (board)	Vários (~10-30 eur)	Sensores: CCD,CMOS Sistema TV.: NTSC/PAL Res.: 420 – 700 linhas de TV Iluminância mín.: < 0.2 lux FOV: 26° ~ 140° Interface: Analógico (conector BNC) Consumos: ~ 1.5 W	

Tabela 3.2: Modelos e caraterísticas de câmaras para visão monocular

3.5.2 Webcams

As *webcams* atraem atenção pelo seu baixo custo. Na atualidade, e depois de uma busca cuidadosa, não se encontram câmaras deste tipo com sensores CCD. Um dado curioso é haver um modelo já descontinuado da Philips, o SPC900NC, que é alvo de inúmeras adaptações e verdadeira especulação por parte da comunidade astronoma. A Tabela 3.3 agrega alguns dos modelos mais proeminentes atualmente no mercado. Justamente por se tratar de equipamento destinado a um mercado de massas, a informação técnica sobre ele disponibilizada é reduzida, não havendo *datasheets* que permitam, por exemplo, conhecer gamas dinâmicas ou iluminâncias mínimas para a quase totalidade dos modelos.

	Modelo	Framerate	Resoluções disponíveis	FOV (lente)	Requisitos mínimos	Focagem	Preço (2012)	
LOGITECH	C270	30fps @ 640x480	4:3 - 320x240, 640x480 16:9 - 360p, 480p, 720p	60° (plástica)	1 GHz CPU 512 MB RAM (256mb em algumas refs.)	Manual	25 eur	
	C525	30fps @ 640x480	4:3 - 320x240, 640x480 16:9 - 360p, 480p, 720p	69° (plástico)	1 GHz CPU 1GB RAM	Auto	42 eur	
	C910	30fps @ 640x480	4:3 - 320x240, 640x480, 1600x1200 16:9 - 360p, 480p, 720p, 1080p	83° (vidro)	1 GHz CPU 512 MB RAM	Auto	75 eur	
	C920	30fps@ 1080p	16:9 - 360p, 480p, 720p, 1080p	78° (vidro)	1.0 Ghz CPU 256MB RAM	Auto	98 eur	
MICROSOFT	HD3000	30 fps	Video: 16:9 - 720p Fotos: 1280X800	68,5°	Intel Dual Core 1.6 GHz 1 GB RAM	Manual	22 eur	
	HD5000	30 fps	Video: 16:9 - 720p Fotos: 1280X800	66°	Intel Dual Core 1.6 GHz 1 GB RAM	Auto	35eur	
GENIUS	Widecam 320	30 fps	640 x 480 pixels	100° (vidro)	Intel 1.6Ghz 1GB RAM	Manual	15 eur	
	Widecam F100	30 fps	12MP (Interpolados), 1920 x 1080, 1280 x 720, 640 x 480	120° (vidro)	Intel Core2Duo 1.8GHz 512MB RAM	Manual	50 eur	
	WideCam 1050	30fps	5MP (Interpolados), 1280 x 720, 640 x 480	120° (vidro)	Intel 2.4GHz 512MB RAM	Manual	37 eur	
PHILIPS	SPZ3000	30 fps	Video: 160 x 120, 176 x 144, 320 x 240, 352 x 288, 640 x 480 Foto: 1280 x 1024 (1.3MP), 2560 x 2048 (Interpolados)	50°	1,6 GHz ou equivalente 512 MB de RAM Iluminância > 5 lux	Manual	20 eur	
	SPZ5000	60fps	Video: 160 x 120, 176 x 144, 320 x 240, 352 x 288, 640 x 480 Foto: 1280 x 1024 (1.3MP), 2560 x 2048 (Interpolados)	80°	1,6 GHz ou equivalente 512 MB de RAM Iluminância > 5 lux	Manual	17 eur	
SONY	Playstation Eye Cam	15 - 120 fps	320 x 240 , 640 x 480	56-75°	ND	Manual	15 eur	

Tabela 3.3: Modelos e características de webcams

3.5.3 Câmaras Estereoscópicas











	Modelo (preço)	Caraterísticas		Hardware Auxiliar
MobileRobotics	MobileRanger (3000- 7500 eur)	<p>Baseline: 6cm (≠ por encomenda)</p> <p>Sensores: 2 x CMOS (monocromáticos) (cores por encomenda)</p> <p>Res.: 752x480 pixels @ 60 fps</p> <p>Gama dinâmica: > 60dB</p> <p>FOV: 75°(H), 52°(V)</p> <p>Interface: LVDS → PC104 (FPGA)</p> <p>Consumo: 500mW</p> <p>Calibração: de fábrica</p> <p>Opção: sistema Pan/Tilt</p>		 <p>nDepth™ Vision Processor Fornecer informação monocromática, cor ou disparidade (92 níveis) a 30 <i>fps</i>.</p>
TYXZ	DeepSea Stereo Camera V2	<p>Baseline: 3, 6, 8, 14, 22cm</p> <p>Sensores: 2 x Micron MT9V022 CMOS (func. monocromático ou cores)</p> <p>Res.: 512x480 pixels @ 60 fps</p> <p>Gama dinâmica: até 100dB</p> <p>FOV: 40°(H), 62° (H) ou 83° (H)</p> <p>Interface: carta PCI</p> <p>Iluminância mín. (mono/cores):0.1/5 lux</p> <p>Calibração: de fábrica</p>		 <p>DeepSeaV2.3 Stereo Processor Fornecer informação monocromática, cor ou profundidade (máx. 52 disparidades) a 60 <i>fps</i></p>
	DeepSea Stereo Camera G3 EVS	<p>Baseline: 3, 6, 8, 14 cm</p> <p>Sensores: 2 x Aptina MT9V024 CMOS [func. mono ou mono + cores (esq)]</p> <p>Res.: 512x480 pixels (cor) 512x2048 pixels (profundidade)</p> <p>Gama dinâmica: até 100dB</p> <p>FOV: 40°(H), 62° (H) ou 80° (H)</p> <p>Interface: carta PCI</p> <p>Iluminância mín. (mono/cores):0.1/5 lux</p> <p>Calibração: de fábrica</p>		
Minoru	Minoru3D (50 eur)	<p>Baseline: 6 cm</p> <p>Res: 1280x480, 800x600, 704x288, 640x480, 640x240, 352x288, 320x240 pixels @ 15-30 <i>fps</i></p> <p>Interface: USB</p> <p>Consumo: 1.5 W</p> <p>Calibração: descalibrada</p>		
nVela	Hydra Stereo (400 eur)	<p>Sensores: 2 x CMOS (cores)</p> <p>Res: 752x480 pixels @ 21 <i>fps</i></p> <p>Interface: USB</p>		
Surveyor	Surveyor SVS (400 eur)	<p>Baseline: 10.75 cm</p> <p>Sensores: 2 x OV9655 (1.3 MP) CMOS (cores)</p> <p>Res: 1280x1024, 640x480, 320x240, 160x120 pixels</p> <p>Gama dinâmica: 50 dB</p> <p>FOV: 90° (120° opcional)</p> <p>Consumo: 2W</p>		
PointGrey	Bumblebee 2 Stereo (1400-1800 eur)	<p>Baseline: 12 cm</p> <p>Sensores: 2 x Sony CCD</p> <p>Res: 640x480 ou 1024x768 (modelos ≠s) (mono ou cores) @ 16– 48 <i>fps</i></p> <p>FOV: 43° (H), 65° (H), 100° (H)</p> <p>Interface: IEEE-1394a</p> <p>Calibração: de fábrica</p> <p>Consumo: 2.5 W</p>		
	Bumblebee XB3 Stereo (2600 eur)	<p>Baseline: 12 – 24 cm</p> <p>Sensores: 3 x Sony CCD</p> <p>Res: 1280 x960 @16 <i>fps</i> (mono ou cores)</p> <p>FOV: 43° (H), 66° (H)</p> <p>Interface: IEEE-1394a</p> <p>Calibração: de fábrica</p> <p>Consumo: 4 W</p>		

Tabela 3.4: Modelos e caraterísticas de câmaras estereoscópicas

3.6 Sistemas Computacionais

Atualmente encontram-se disponíveis uma série de plataformas computacionais potentes, utilizáveis numa série de aplicações distintas. Nos últimos anos, em particular graças à grande difusão da tecnologia ARM através do mercado de telemóveis, *tablets* e computadores de baixo custo, impulsionada pela necessidade de um compromisso entre performance e autonomia, sistemas deste tipo dão resposta a tarefas computacionalmente exigentes com um nível de consumo muito baixo.

Percebendo a existência de uma comunidade de consumidores domésticos interessada em criar e experimentar aplicações ou pequenos projetos envolvendo uma componente de informática e/ou eletrónica, alguns fabricantes desenvolveram placas com as mais variadas funcionalidades, protocolos de comunicação, entradas e saídas, compatíveis com sistemas operativos de utilização generalizada. Estas placas podem, por isso, ser tidas em conta para aplicações na área da robótica, onde muito frequentemente se utiliza *hardware* de fabricantes especializados e, conseqüentemente, caro.

A tabela 3.5 reúne algumas das propostas mais interessantes da atualidade.







Modelo	CPU	GPU	Memória	Sist. Operativos Compatibilidade	E/S mais relevantes	Consumo	Dimensões	Preço
Allwinner A10 Dev Board	1.2GHz Allwinner A10 ARM Cortex A8	Mali-400	DDR3 512Mb / 1 Gb + 4 Gb memória interna	Android, Linux	2x USB 2.0 Ethernet Wifi 802.11n RS-232 HDMI Leitor SDcard Saída Audio	< 700mA (5V)	8.6x 5.3cm	\$65 
CubieBoard2	ARM® Cortex™-A7 Dual-Core	ARM Mali 400 MP2	1GB DDR3 + 4Gb memória interna	Android, Linux	*2x USB 2.0 *Ethernet *HDMI *Audio *IR *SATA *96 pinos disponíveis e\ I2C, SPI, RGB/LVDS, CSI/TS, FM-IN, ADC, CVBS, VGA, SPDIF-OUT, R-TP, ...	~500 mA (5V)	10x6cm	\$59 
Hardkernel Odroid-U2	1.7GHz Exynos4412 ARM Cortex-A9 Quad Core 1.7GHz	Mali-400 Quad Core	2 GB DDR2	Android, Linux	*2xUSB 2.0 *Ethernet *Audio *Leitor MicroSD *Micro HDMI	1-1.5A (5V)	4.8x5.2cm	\$89 
Hardkernel Odroid-X2	1.7GHz Exynos4412 ARM Cortex-A9 Quad Core 1.7GHz	Mali-400 Quad Core	2 GB DDR2	Android, Linux	*6xUSB 2.0 *Micro USB *Ethernet *Audio *Entrada para câmara *Leitor SD *50 Pin (LCD-RGB, PWM, ADC, SPI, I2C, UART and GPIO)	~ 1A (5V)	9.0x9.4cm	\$135 
BeagleBoard BeagleBone Black	AM335x Cortex A8 ARM	-	512 Mb DDR3	Linux	*HDMI *RS-232 *Leitor MicroSD *VDD_ADC(1.8V), 3.3V I/O, McASP0,SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7 AIN (1.8V Max), XDMA, ...	210-460 mA (5V)	8.6x5.3cm	\$45 
A13-OLinuXino-WIFI	A13 Cortex A8 processador a @ 1GHz		512 MB + 4 GB de memória interna	Android, Linux	*3xUSB *WIFI 802.11n *Leitor cartão SD *Saída VGA *Audio *UEXT *GPIO's (I2C, UART, I/O's, ...)	1A (5V)	12x12cm	€55 

Tabela 3.5: Modelos e características de sistemas computacionais de baixo custo

3.7 O modelo de câmara *pin-hole*

O modelo *pin-hole*, representado na Figura 3.6, parte do princípio de que todos os raios que incidem no plano da imagem entram na câmara escura através de um pequeno orifício. Este "buraco de alfinete" é o centro ótico da câmara.

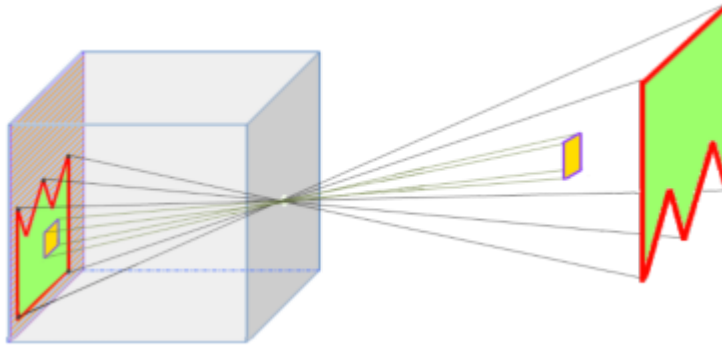


Figura 3.6: Modelo de câmara *pin-hole*

A Figura 3.7 mostra a perspectiva da visão computacional. O plano da imagem encontra-se à distância focal f do centro ótico, isto é, quando $Z=f$. Um ponto no espaço, A , surge representado em a , no local em que o plano da imagem se vê atravessado pela linha que une A ao centro ótico da câmara. O ponto p , ponto principal, resulta da interseção do eixo principal com o plano da imagem.

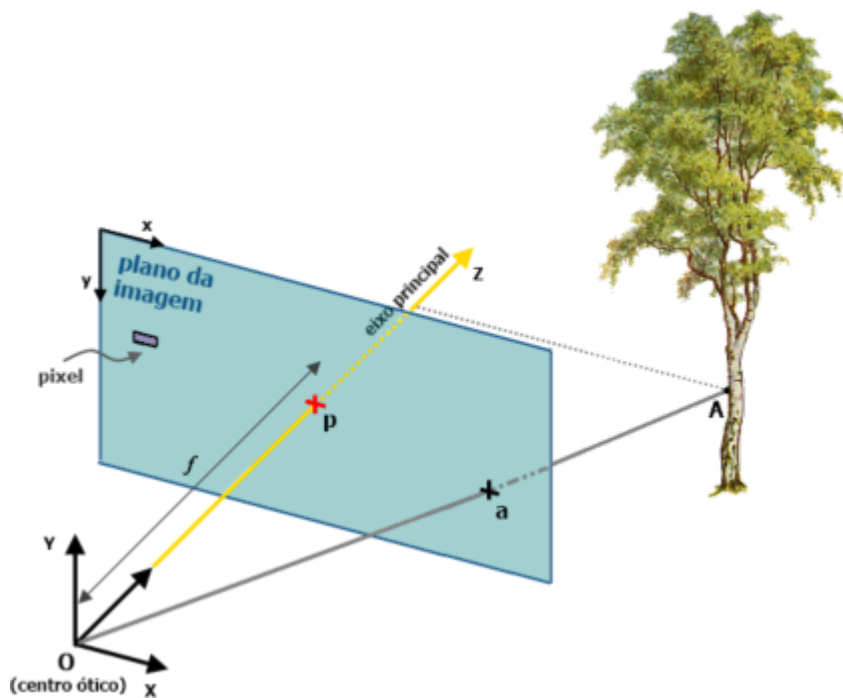


Figura 3.7: Modelo para visão computacional

3.8 Par estereoscópico

O problema da interposição de objetos revela uma insuficiência da utilização de uma câmara apenas. De todos os pontos sobre a linha que une **A** a **O** apenas um, o mais próximo, é mapeado no plano da imagem. A Figura 3.8 ilustra este caso: o ponto **B** está refletido em **c** mas **A** não. Na mesma figura está também ilustrada a restrição epipolar. Um ponto, visível para ambas as câmaras (neste caso **B**), tem representação **C** no plano de imagem esquerdo. A restrição epipolar diz-nos que a representação correspondente desse ponto no plano de imagem da direita estará obrigatoriamente sobre a linha amarela. A linha amarela é uma projeção do segmento \overline{OA} no plano de imagem da câmara direita. Esta restrição reduz enormemente o esforço computacional para encontrar correspondências entre objetos nas duas câmaras, essenciais ao cálculo da distância a que se encontra o objeto. Ao invés de um varrimento de toda a imagem para encontrar a correspondência, a busca faz-se agora apenas segundo a linha epipolar.

A existência da segunda câmara permite a determinação da distância a que se encontra qualquer ponto avistado por ambas, desde que seja conhecida a disposição geométrica das câmaras. Essa determinação faz-se por meio de triangulação, após um processo de calibração do par estereoscópico.

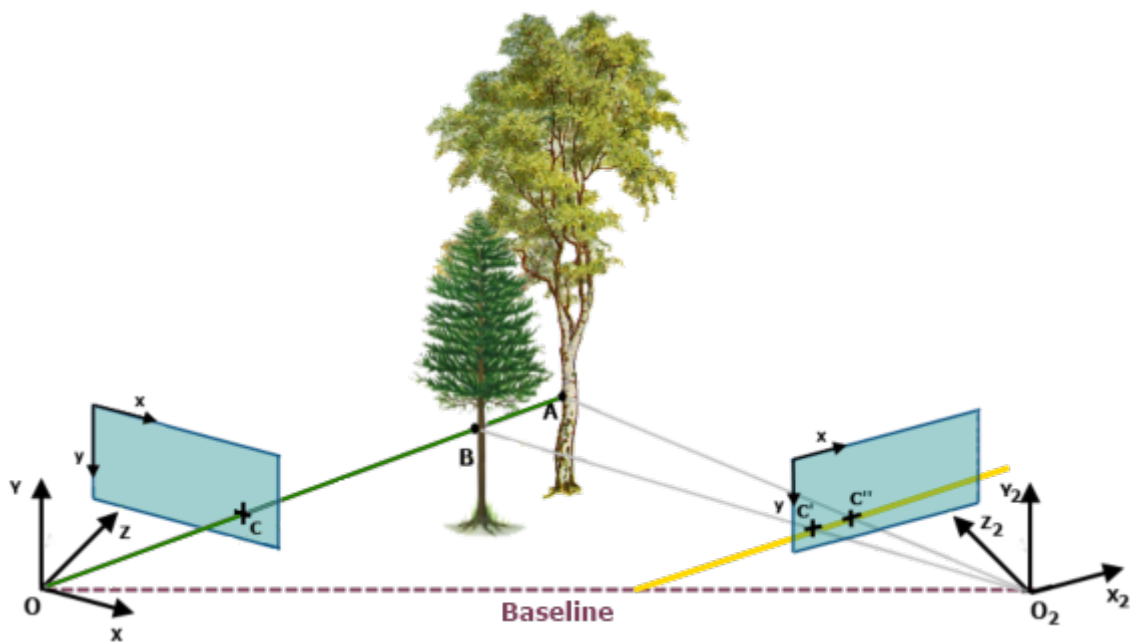


Figura 3.8: Montagem com par de câmaras

3.9 Calibração

O processo de calibração é indispensável para se poder levar a cabo a medição de profundidade dos elementos visíveis em ambas as imagens.

Embora algumas estruturas stereo sejam fornecidas já calibradas de fábrica, por exemplo, as *Bumblebee* da *Pointgrey*, estes modelos são normalmente bastante dispendiosos. Os processos de fabrico recorrem a sensores e lentes de alta qualidade e a um grande cuidado no processo de montagem. Pelo contrário, quando recorremos a estruturas stereo ou mesmo câmaras de mais baixo custo, a calibração tem necessariamente de se fazer por via do software. Optando por esta última via, lentes com imperfeições e sensores geometricamente mal posicionados (Figura 3.9) são problemas que não podemos resolver intervindo ao nível *hardware*; se assim fosse, além de não dispormos de equipamento para esse efeito, perder-se-ia a vantagem da utilização de componentes de baixo custo. O objetivo aqui é ter um par não geométrica mas matematicamente alinhado.

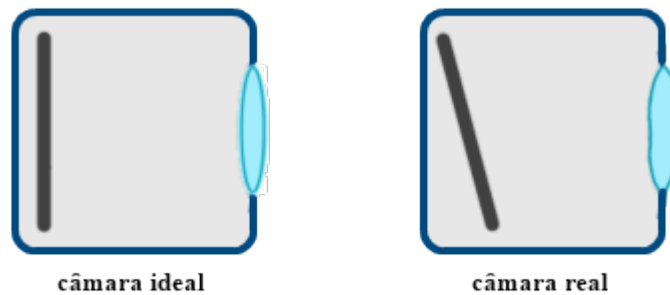


Figura 3.9: Imperfeições físicas das câmaras

Com a exceção de aplicações de curto alcance, em que se pretende melhorar a resolução na proximidade arranjando as câmaras de forma a que os eixos principais convirjam num ponto, a configuração de interesse é a de paralelo frontal [45]. Nesta configuração, as câmaras devem estar posicionadas de forma a que os eixos principais sejam paralelos e os planos de imagem horizontalmente alinhados. Na Figura 3.10 surgem representados os epípolos e_1 e e_2 , da configuração desalinhada. Com o alinhamento dos planos de imagem, os epípolos aproximam-se do infinito, não tendo por isso representação. Assumindo que foram eliminadas as distorções intrínsecas à câmara e estando os planos de imagem alinhados, os pontos D' e D'' deverão estar sobre a mesma linha, permitindo uma avaliação direta e unidimensional da disparidade. Atente-se à diferença face ao cenário anterior, reparando na discrepância vertical entre os pontos d' e d'' : além dos problemas já referidos acima, procurar correspondências num sistema desalinhado aumenta em muito o risco de falso *match*.

A Figura 3.11 apresenta uma vista superior da situação apresentada na imagem anterior. Depois de encontradas as projeções de D nos dois planos de imagem, as distâncias x_{De} e x_{Dd} são medidas. Por norma, na visão computacional a origem do referencial está no canto superior esquerdo da imagem.

Medidas as distâncias, podemos encontrar o valor da disparidade:

$$\text{Disparidade} = x_{De} - x_{Dd}$$

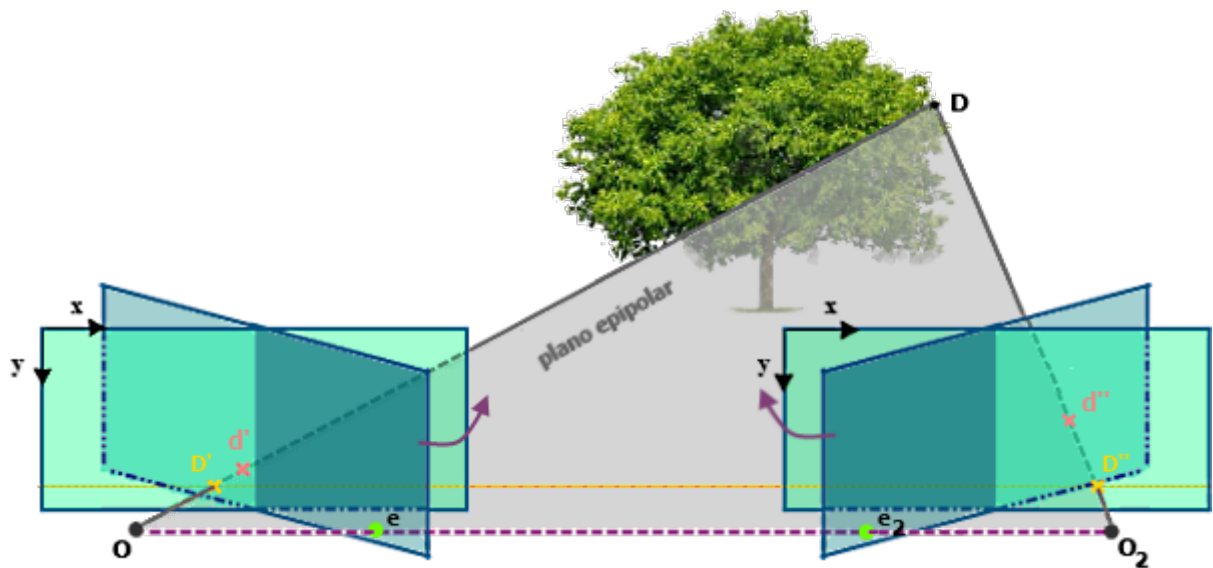


Figura 3.10: Par alinhado

Esta informação, juntamente com dados conhecidos da configuração geométrica que lhe está associada, permite-nos inferir o valor da profundidade Z :

$$\begin{aligned} \frac{Baseline}{Z} &= \frac{Baseline + X_{Dd} - X_{De}}{Z - f} \Leftrightarrow \\ \Leftrightarrow \frac{(Z - f) \cdot Baseline}{Z \cdot (Z - f)} &= \frac{Z \cdot (Baseline + X_{Dd} - X_{De})}{Z \cdot (Z - f)} \Leftrightarrow \\ \Leftrightarrow Z &= \frac{Baseline \cdot f}{X_{De} - X_{Dd}} \Rightarrow \\ Profundidade &= \frac{Baseline \cdot f}{Disparidade} \end{aligned}$$

Na Figura 3.12 esboça-se a relação entre a disparidade e a profundidade. Conforme espectável, quando temos um objeto mais próximo, temos uma disparidade maior. Isto resulta também num menor erro no cálculo da profundidade.

A disparidade é um valor inteiro, porque a imagem está discretizada e a sua menor unidade é 1 pixel. Assim sendo, quanto mais longe estiver o objeto maior será o erro na distância obtida, como é visível pela derivada acentuada quando a disparidade se aproxima de zero.

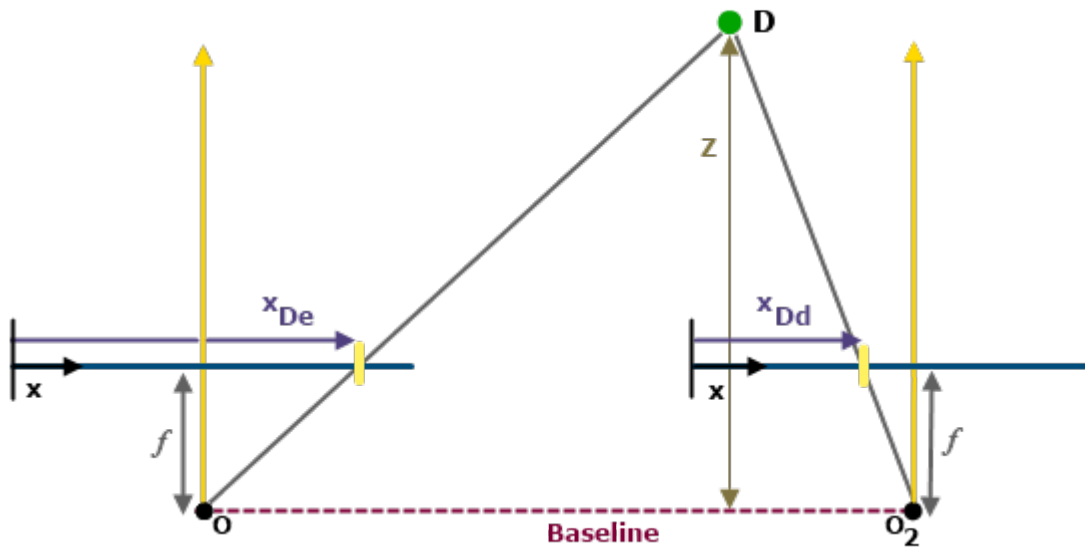


Figura 3.11: Disparidade vs Profundidade

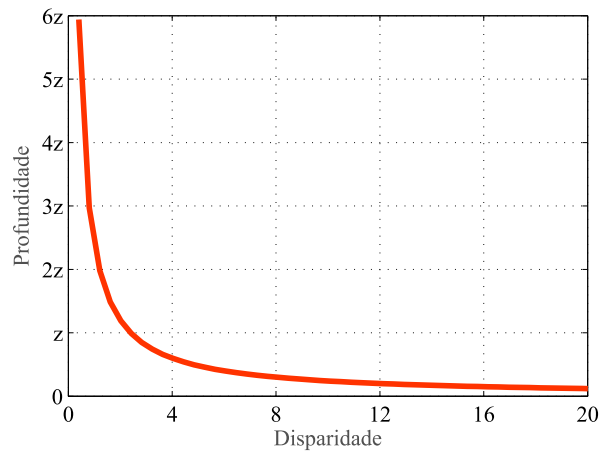


Figura 3.12: Relação Disparidade-Profundidade

3.10 Parâmetros Intrínsecos e Extrínsecos

A calibração das câmaras procura eliminar os problemas resultantes, quer de defeitos estruturais da própria câmara, quer dos desalinhamentos de uma câmara face a outra, no caso do par estereoscópico.

Os parâmetros extrínsecos definem a localização e orientação da câmara face a um referencial conhecido do mundo; os intrínsecos servem para estabelecer a relação entre as coordenadas de um ponto da imagem e as coordenadas do referencial da câmara.

Os parâmetros intrínsecos são os seguintes:

- **Distância Focal** - distância, em pixels, entre o centro de projeção e o plano imagem (segundo o eixo horizontal e vertical)

- **Skew ou Coeficiente de desfasamento** - coeficiente que relaciona o eixo horizontal e vertical do sensor (e portanto, da imagem produzida) dado que os pixels podem não ser perfeitamente quadrados. É a relação entre o número de pixels por unidade de comprimento em cada uma daquelas direções. Tem valor zero para a generalidade dos sensores atuais.
- **Pixel Aspect Ratio** - Relação entre a altura e largura do pixel do sensor. Tipicamente com valor um nos sensores atuais.
- **Centro ótico** - projeção da origem do referencial da câmara no plano da imagem (segundo eixo horizontal e vertical)
- **Distorções** - coeficientes de distorção radiais e tangenciais. A *Toolbox* de Matlab apresentada na secção seguinte considera cinco (3 radiais e 2 tangenciais), o OpenCV considera até oito (6 radiais e 2 tangenciais).

Os extrínsecos são a rotação tridimensional (R_x, R_y, R_z) e translação tridimensional (T_x, T_y e T_z) do referencial da câmara face a qualquer outro referencial (no caso do par estereoscópico, do referencial da câmara direita face ao da câmara esquerda).

3.11 Ferramentas de Calibração

Estão disponíveis inúmeros métodos e ferramentas de calibração online.

Tipicamente é utilizado um tabuleiro de xadrez para o efeito, embora haja calibrações feitas com padrões de pontos e objetos tridimensionais. Os padrões têm uma qualquer propriedade conhecida do software de calibração (ex. cantos dos quadrados do tabuleiro de xadrez que devem coincidir em linhas que se entrecruzam), devem ser observados pelas câmaras de diversas perspectivas de maneira a, iterativamente, melhorarem a compensação das distorções do hardware, até que um valor de erro mínimo de reprojeção seja atingido. Calibrações que envolvam poucas imagens, imagens de perspectivas pouco variadas ou em que o padrão de calibração se encontra demasiado longe da câmara são de pouca utilidade. O padrão utilizado deve sempre cobrir uma área tão grande quanto possível da imagem e não sofrer, ele próprio, de qualquer tipo de deformação. A calibração deve fazer-se sempre utilizando a maior resolução possível do equipamento, ainda que essa não seja a resolução com que se pretende funcionar. Sendo esse o caso, os parâmetros intrínsecos da câmara (distância focal e ponto de interseção do centro ótico no plano de imagem) exceto distorções da lente devem ser sujeitos ao fator de escala do tipo $e_x = \frac{\text{resolucao_horizontal_destino}}{\text{resolucao_horizontal_calibracao}}$ e $e_y = \frac{\text{resolucao_vertical_destino}}{\text{resolucao_vertical_calibracao}}$.

Existe uma *Toolbox* de Matlab, que permite a calibração monocular e de pares estereoscópicos. O seu funcionamento prevê a calibração individual de cada câmara, devolvendo informação dos parâmetros intrínsecos de cada uma - Distância focal, Centro ótico, Distorções e Coeficiente de Desfasamento. Assumindo que cada par de imagens direita-esquerda foi adquirido com o tabuleiro visível por ambas as câmaras, bem como que não houve, durante a calibração, alterações na disposição geométrica das câmaras no suporte, pode fazer-se a calibração estereoscópica. O final

desse processo devolve o vetor de rotação e translação da câmara direita em relação à esquerda e otimiza os parâmetros intrínsecos de forma a minimizar o erro de reprojeção. O processo de calibração em Matlab não é, ao contrário de outros, automático. O utilizador tem de definir os limites do tabuleiro para cada imagem usada na calibração e também, para cada uma, fazer ajustes aos desvios dos cantos dos quadrados assinalados pelo Matlab. Todo o processo se encontra descrito em [46] e outros detalhes também bem descritos neste excelente documento do DEM-FEUP [47]. A primeira configuração de câmaras utilizada foi calibrada utilizando esta *Toolbox*. A disposição das câmaras encontra-se visível na Figura 3.13.

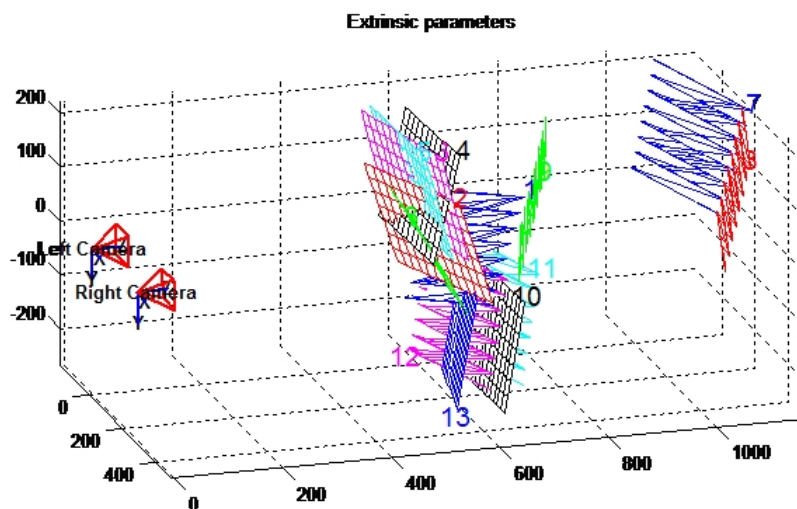


Figura 3.13: Matlab - Camera Calibration Toolbox

Existem ainda as funções de OpenCV, que foram utilizadas ao longo de todo o trabalho. Se considerarmos, como foi o caso, um tabuleiro de xadrez de 13 x 8 quadrados para a calibração, a sequência de passos seguinte é executada para a calibração estereoscópica:

- **Abertura ou recolha imagem da câmara/câmaras**
- **Contagem e registo de cantos do tabuleiro detetados** - Utilizando a função *cvFindChessboardCorners*, armazena as coordenadas de todos os cantos do tabuleiro
- **Melhora a precisão dos cantos detetados** - Utilizando a função *cvFindCornerSubPix*, recebe como entrada os cantos detetados e devolve, limitado por um critério de paragem (n° de iterações ou precisão pretendida), os mesmos cantos com uma localização mais precisa relacionando para tal, propriedades de ortogonalidade entre vetores que ligam as coordenadas do canto detetado a outros pontos e o gradiente da zona envolvente;
- **Executa calibração estereoscópica** - Com a função *cvStereoCalibrate*, que parte de valores iniciais para os parâmetros extrínsecos da câmara (matrizes de rotação e translação) provindos da função *cvCalibrateCamera2*, utiliza iterativamente um algoritmo de otimização Levenberg-Marquardt para, localmente, reduzir o erro de reprojeção dos cantos, sendo

interrompida assim que atingido um critério de paragem. Esta função transforma as imagens esquerda e direita em imagens complanares.

- **Calcula termos a utilizar na retificação** - Nesta fase, usando a função *cvStereoRectify*, obtemos os parâmetros para o alinhamento vertical das imagens e uma matriz que se revelará de extrema importância para inferir posição dos objetos detetados pelo algoritmo, apresentada em baixo.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & \frac{(c_x - cx')}{T_x} \end{bmatrix} \quad \text{Com:} \quad Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

com que c_x e c_y são as coordenadas da projeção do centro ótico da câmara esquerda no plano da imagem, c'_x idêntica coordenada na câmara direita, f a distância focal da câmara esquerda e T_x é coordenada do vetor de translação dos parâmetros extrínsecos do par estereoscópico.

É aquela matriz que permite relacionar as coordenadas 2D da reflexão do ponto na imagem, (x,y) , com as coordenadas 3D (X,Y,Z) do ponto no mundo. Tal acontece, para um dado ponto (x,y) , através das seguintes relações:

$$d = x_{direita} - x_{esquerda}$$

$$X = x_{esquerda} * Q(0,0) + Q(0,3);$$

$$Y = y_{esquerda} * Q(1,1) + Q(1,3);$$

$$Z = Q(2,3);$$

$$W = d * Q(3,2) + Q(3,3);$$

**

$$X(3D) = \frac{X}{W}$$

$$Y(3D) = \frac{Y}{W}$$

$$Z(3D) = \frac{Z}{W}$$

**

Pode fornecer-se ao programa informação sobre a dimensão dos quadrados do tabuleiro, sendo que dessa forma todos os valores gerados pelas relações acima estão nas unidades com que esses valores foram fornecidos. Caso contrário, os valores $X(3D)$, $Y(3D)$ e $Z(3D)$ são múltiplos do lado do quadrado do tabuleiro.

- **Gera mapas para a função de remapeamento** - a função *cvInitUndistortRectifyMap* recebe, para cada câmara, matrizes com os parâmetros da câmara e com os parâmetros retificados da câmara, coeficientes de distorção e a matriz de rotação da câmara proveniente da função *cvStereoRectify* para gerar os mapas de remapeamento para a função *cvRemap*, isto é, mapas com informação para correção da posição de cada pixel da imagem. Para cada imagem, são gerados dois mapas, um por coordenada.

Adiante perceber-se-á que houve necessidade de criar uma função de reconversão destes mapas, para obtenção de informação de correção de cada ponto, individualmente.

- **Algoritmo de Matching (Block Matching ou Graph Cut)** - Opcionalmente, utilizando as funções *cvFindStereoCorrespondenceBM* ou *cvFindStereoCorrespondenceGC* pode estabelecer-se correspondência entre as imagens e gerar mapas como forma de disparidade para aferir a qualidade da calibração.

3.12 Testes de Calibração

No sentido de aferir a qualidade da calibração, adquiram-se pares de imagens obtidos de diferentes perspectivas e extraíram-se, a partir deles, distâncias relativas a um mesmo elemento.

Com uma fita métrica foram tiradas medidas aproximadas das distâncias X, Y e Z da bola face ao par de câmaras. Os valores obtidos podem comparar-se com os inferidos pelo par estereoscópico tendo por base duas resoluções de imagem diferentes e encontram-se reunidos na Tabela 3.7. A tabela regista as coordenadas (em pixels) do centro da bola azul na imagem esquerda e direita do par e respetiva disparidade. Para garantir maior precisão na deteção do objeto e respetivo centro em cada imagem, não houve recurso a nenhum algoritmo; a identificação fez-se individualmente num editor de imagem. Com base nessa informação e na matriz **Q** obtida pelo processo de calibração, obtêm-se os valores das distâncias X, Y e Z e apresenta-se o erro relativo médio face aos valores medidos no local para cada uma destas coordenadas. O par de imagens 2 foi mantido apenas para demonstrar a consistência dos valores de profundidade nas medições de grandes distâncias (10 m); a medida de X apresenta para este par de imagens um erro grande que pode perceber-se se considerarmos que para aquela distância Z, 9.5m, um desvio da orientação do par estereoscópico de cerca de 6° gera uma coordenada X idêntica à registada.

As medições efetuadas apresentam valores de erro relativo menores para as imagens com maior resolução. Mesmo assim, a diferença não é grandemente relevante no caso das imagens analisadas. Daqui não decorre, como seria expectável, que a qualidade da medição seja independente da resolução com que se faz a aquisição. O gráfico logarítmico da Figura 3.8 apresenta, para duas resoluções de imagem distintas, a relação disparidade-profundidade deste par estereoscópico, **Z(i)**, e também uma medida do máximo erro numa medida para uma dada disparidade, ΔZ , que, para o valor de disparidade **i** provém da diferença entre **Z(i-1)** e **Z(i)**. Assinaladas a preto, duas



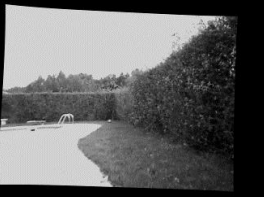



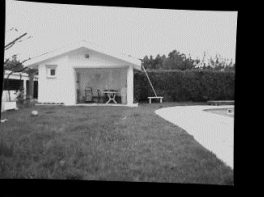




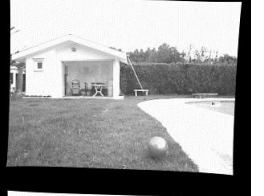








Par de imagens	Imagem original		Imagem remapeada	
	Esquerda	Direita	Esquerda	Direita
2				
3				
5				
7				
8				

Tabela 3.6: Pares de imagens

medições de uma mesma distância usando resoluções distintas revelam a possibilidade de se cometer um erro quatro vezes maior com o par de imagens de pior definição; este facto não é alheio à relação de 1 para 4 entre as áreas, em píxeis, das imagens de menor e maior resolução.

par de imagens	medição no terreno (m) (aproximadas)			medição stereo (m) (resolução: 640x480)						medição stereo (m) (resolução: 320x240)					
	X	Y	Z	X	Y	Z	disp. (px)	coordD(x,y)	coordE(x,y)	X	Y	Z	disp. (px)	coordD(x,y)	coordE(x,y)
2	0	-1,2	9,5	1,03	-1,35	9,58	-13	(269,298)	(282,297)	1,09	-1,44	10,3	-6	(136,148)	(142,148)
3	3	-0,75	5	2,8	-0,71	5,19	-24	(70,295)	(94,294)	3,01	-0,73	5,6	-11	(38,146)	(49,146)
5	-0,5	-0,75	1,6	-0,68	-0,65	1,75	-71	(426,397)	(497,396)	-0,68	-0,66	1,76	-35	(213,198)	(248,198)
7	2,3	1,4	9,7	1,4	1,44	10,35	-12	(261,176)	(273,175)	1,1	1,21	8,9	-7	(131,88)	(138,88)
8	0,95	-0,75	1,55	0,71	-0,67	1,83	-68	(93,396)	(161,396)	0,74	-0,7	1,93	-32	(50,196)	(82,196)
erro relativo (%) - (sem par 2)				8,77	6,62	9,49				9,49	8,73	9,57			
erro relativo (%) - (com par 2)				12,99	2,80	7,76				12,41	2,98	9,34			

Tabela 3.7: Medições com par estereoscópico

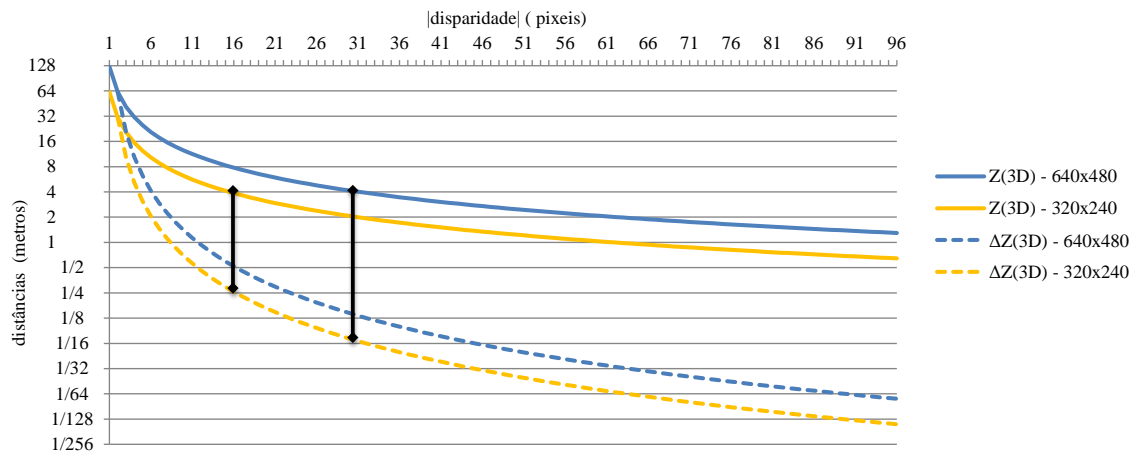


Tabela 3.8: Influência da resolução na qualidade da medição

Capítulo 4

Sistema de Condução Visual

Este capítulo apresenta o sistema de aquisição de imagem, detalhando individualmente cada um dos seus constituintes. São apresentados vários algoritmos que foram desenvolvidos para se perceber a viabilidade da sua aplicação, bem como a solução final, incluindo as interações com o *hardware e framework* previamente existentes na embarcação.

4.1 Visão Stereo

O desvio de obstáculos, função de importância crítica num veículo autónomo, depende em grande medida não só da capacidade de identificação dos mesmos como da estimação *à priori* de um valor da distância a que se encontram. A visão estereoscópica é uma das formas de extrair essa informação do ambiente, tendo relativamente a outros modos de deteção a vantagem de fornecer informação de cor para cada ponto da cena capturada. A valia deste método é, conforme explicado anteriormente, precisamente o facto de extrair informação de profundidade de um par de imagens 2D: dado um ponto numa imagem desse par, encontrar o correspondente na outra de forma a que ambos representem projeções de um mesmo ponto no mundo. A técnica pressupõe que as duas imagens da cena sejam obtidas em sincronismo por duas câmaras, separadas por uma *baseline*. O comprimento desta baseline tem de ser definido com base num *tradeoff*: aumentá-la resulta numa melhor estimação de distâncias aos pontos do mundo mas numa maior dificuldade em encontrar correspondência de pontos entre imagens, potenciando-se a geração de um maior número de falsos positivos. A melhor qualidade da medição de maiores profundidades justifica-se com a relação mais favorável entre comprimento da baseline e a profundidade medida: pontos mais distantes encontram-se sobre linhas que fazem ângulos muito pequenos com o eixo ótico das câmaras; quanto mais pequena a baseline, menores serão as distâncias a partir das quais, do seu ponto de vista, o objeto se encontra no infinito. O reverso da medalha acontece por força dos efeitos que esta maior diferença de perspetiva tem na forma como o objeto é percecionado por cada

câmara. Diferenças na iluminação podem, do ponto de vista de cada uma das câmaras, tornar as duas representações (uma em cada câmara) de um mesmo objeto muito distintas.

numa zona de derivada acentuada do gráfico distância/disparidade apresentado na secção anterior pelo que o erro da medição se eleva de forma muito acentuada

4.2 Hardware

4.2.1 Sistema computacional Raspberry Pi

O Raspberry PI (Figura 4.1) é um pequeno computador lançado em 2012 cujo desenvolvimento partiu do departamento de Ciência dos Computadores da Universidade de Cambridge.

Da constatação de que os conhecimentos de programação dos novos alunos vinham diminuindo desde a década de 90, nasceu a ideia de criar uma máquina versátil e barata, numa plataforma aberta que permitisse a experimentação. O resultado foi um sistema do tamanho de um cartão de crédito, baseado no processador multimédia SOC (*System-on-Chip*) BCM2845 da Broadcom, assente em tecnologia ARM de 11^a geração.

Foram originalmente previstos dois modelos, A e B, com 128 e 256MB de RAM respetivamente, nunca chegando esse modelo A a ser comercializado. O modelo B foi entretanto melhorado, vindo elevada para 512MB a sua memória RAM, sem que isso tenha tido impacto no preço original de comercialização, de cerca de 35 euro; a versão A tem agora 256MB e custa menos de 30 eur. Nenhuma delas possui capacidade de armazenamento de dados, pelo que o seu registo é feito num cartão SD de até 32GB.

Para este trabalho foi utilizado um modelo B de 256MB que, face ao mais económico, oferece uma porta USB extra e porta Ethernet 10/100. Foi ainda utilizado um cartão SD Classe 10 de 8GB.

Alem destas características, o Raspberry PI inclui 8 I/O digitais, barramentos I2C e SPI, UART, saída audio e saídas video analógica e HDMI, com uma GPU (*Graphics Processing Unit*) capaz de decodificação 1080P.

É alimentado com 5V via entrada microUSB, conseguindo consumos em torno dos 3W.

Segundo a página oficial, a performance do sistema deve aproximar-se da de um Pentium II a 300MHz, com gráficos de qualidade superior, equivalentes ao de uma XBOX1.

O processador de virgula flutuante ARM1176JZF5, que funciona habitualmente a uma frequência a 700MHz, sofreu para esta aplicação um *overclock*, passando a funcionar a 900MHz. Após esta modificação, o sistema não evidenciou nenhuma alteração da sua estabilidade geral.

A comunidade Raspberry PI é dinâmica e já de dezenas de milhares de utilizadores, desenvolvendo, na sua maioria, aplicações nas linguagens Python e C/C++. Estão disponíveis, com atualizações frequentes, vários sistemas operativos, entre distribuições de Linux, Android e, mais recentemente, uma versão do RISC OS, desenhado de raiz para processadores ARM.

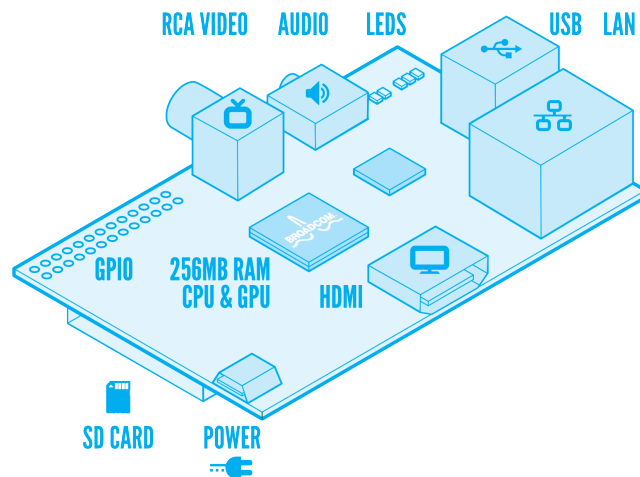


Figura 4.1: Diagrama do Raspberry PI utilizado [<http://www.raspberrypi.org/>]

4.2.2 Câmaras

De entre os sensores disponíveis para executar a tarefa de identificação de obstáculos e alvos, a escolha recaiu sobre visão estereoscópica baseada em câmaras de baixo custo.

Ao contrário do que acontece com modelos de centenas ou milhares de euros, além do problema de não estarem calibradas, as webcams USB não têm nenhuma entrada de *trigger*. O trigger permite impôr a uma ou mais câmaras uma seqüência de aquisições simultâneas. Associar um mesmo sinal de trigger a um par estereoscópico garante a sincronização do hardware, primeiro preceito de um verdadeiro sistema de aquisição *stereo*.

Sistemas como o descrito em [14] recorrem a um par de câmaras não sincronizado, utilizando uma estratégia de 3 aquisições sucessivas, num esquema **esquerda -> direita -> esquerda** (Figura 4.2).

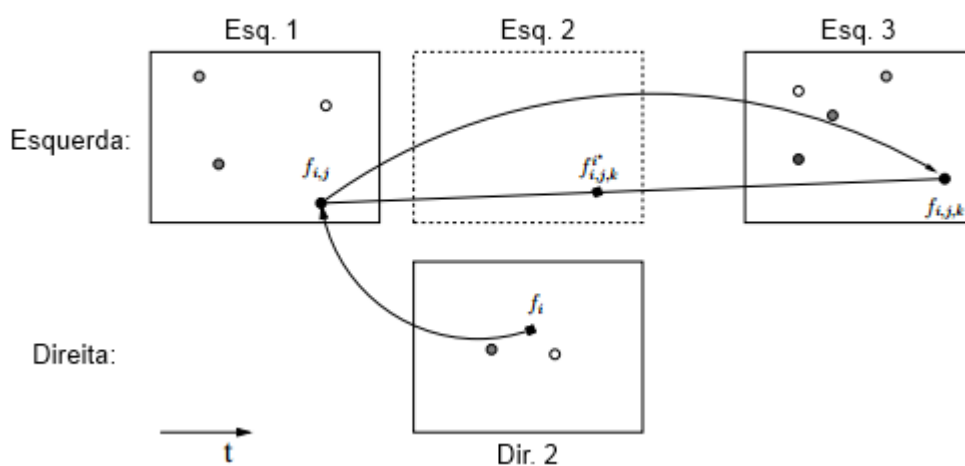


Figura 4.2: Sistema Dessincronizado - Exemplo de Sequência de Aquisição (adaptado de [14])

Como é notório, por força de o sistema não estar sincronizado a imagem **Dir. 2** não encontra par na câmara esquerda. A estimação das coordenadas de um dado ponto de interesse na "virtual" imagem **Esq.2** é feita através de uma interpolação para o *timestamp* de **Dir. 2**, após um processo que estabelece a correspondência do ponto de interesse nas três imagens adquiridas. Esta é uma entre várias formas de lidar com o problema da não sincronização de imagem. Em [48] sugere-se que se tire partido dos microfones frequentemente incorporados em câmaras de baixo custo - algo de que dispunham os três modelos em que houve oportunidade de fazer experiências - para, utilizando o canal audio, estabelecer o sincronismo do canal vídeo.

O recurso a estas estratégias acontece de forma a evitar a aquisição de equipamento mais dispendioso, com capacidade de sincronização.

O objetivo deste trabalho foi, contudo, tentar sincronizar um par de câmaras.

A escolha do Raspberry Pi como unidade computacional responsável pelo processamento de imagem impõe restrições e exige critério nos métodos utilizados.

Partindo do pressuposto errado de que seria exequível um sistema de processamento de imagem assente em Raspberry Pi com *framerates* da ordem das poucas dezenas de *frames* por segundo, a escolha da câmara fez-se com base nos critérios indicados no capítulo de revisão bibliográfica. A abundância de métodos de visão *pseudo* estereoscópica, *framerate*, a análise de vídeos disponíveis no *Youtube* que permitiu perceber a resposta de alguns dos modelos listados a diferentes condições ambientais, o fator baixo custo, tónica do trabalho desenvolvido, e a geometria levaram a crer que a escolha do par de câmaras **Philips SPZ-5000** seria a melhor. Rapidamente se concluiu que, em especial com duas câmaras, a taxa de aquisição do par não chegaria à dezena de *fps*. A estratégia subsequente foi uma tentativa de sincronização do par.

Algumas *webcams* (Figura 4.3) incluem pares de sensor / *bridge processor* com possibilidade de, por adaptação de hardware, serem sincronizadas. Essa possibilidade foi explorada no caso das câmaras Philips, tendo sido inspecionado o circuito de uma delas.

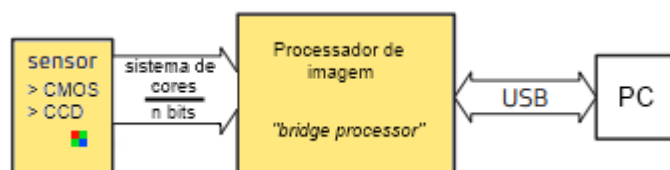


Figura 4.3: Webcam - Esquema simplificado da estrutura interna

O *bridgeprocessor* desta câmara, o **SN9c250** da Sonix, permitiu, com o uso de um osciloscópio, descobrir um sinal de sincronização vertical. Este sinal tem a forma de um pente de impulsos e, nos três modelos de câmaras analisados ao longo do trabalho, uma frequência igual ao *framerate* escolhido.

O sinal de sincronização vertical, VSYNC (Figura 4.4), é o sinal que despoleta a aquisição de um *frame*. No exemplo, referente à aquisição de uma imagem VGA (resolução de 640x480 pixels), é visível o varrimento linha a linha. Dentro do período do sinal VSYNC cabe a aquisição dos

640 pixels de uma linha (t_{Line}) e uma folga inicial e final de $15 \times t_{Line}$. Um clock comum governa o sistema e estes sinais são enviados pelo sensor ao *bridgeprocessor*, para com ele sincronizar a transferência de cada pacote de dados.

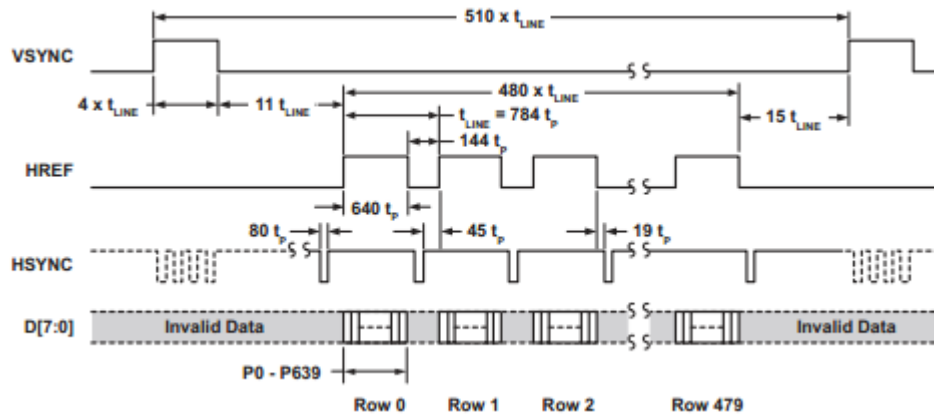


Figura 4.4: Aquisição de *frame* VGA - exemplo de diagrama temporal

É exatamente este sinal VSYNC que, em certos casos, pode ser usado para sincronizar as câmaras. Um sistema numa estrutura *master-slave*, com duas (ou mais) câmaras funcionando com a mesma frequência de aquisição e em que a (ou as) câmaras *slave* tenham um sensor com possibilidade de sincronização de *frames* (como, por exemplo, o OmniVision OV9713) torna possível fazê-lo.

A *datasheet* do *bridgeprocessor* da câmara Philips não estava disponível no site do fabricante. Foi tentado por duas vezes o contacto, com o intuito de a obter. Não houve resposta.

Um outro modelo de câmara disponível, de marca branca, foi inspecionado. O fabricante do processador **ARK3299**, a ARKmicro, foi contactado e também não respondeu. Neste caso, mais tarde descobriu-se que o sensor CMOS era o OmniVision **OV7660** e confirmou-se na *datasheet* que não dispunha entrada de sincronização de frames.

Alguma pesquisa online mostrou que o sensor CMOS **OV7720** da OmniVision - uma marca a que, ao contrário das outras, é de agradecer a abundância de informação técnica - além de ter essa capacidade, está disponível numa câmara que custa 15 euros : a **PS3 EyeCam**.

Fez-se a adaptação das duas câmaras, deixando um terminal acessível do exterior em cada uma delas (Figura 4.5).

No final da adaptação, foi feita prova da sincronização das câmaras. Os sinais apresentados abaixo (Figuras 4.6 e 4.7) são resultado da aquisição do sinal VSYNC em ambas as câmaras.

Utilizando o software gratuito *Kinovea*, que permite análise de vídeo frame a frame e sincronização de vídeo, foram gravadas seqüências de vídeo de um cronómetro disponível *online* com resolução de 1 ms. Mesmo tendo o vídeo sido adquirido de um monitor, ou seja, mesmo estando a taxa de refrescamento dos valores do cronómetro limitada à taxa do monitor, e notória a diferença entre o caso sincronizado e não sincronizado.

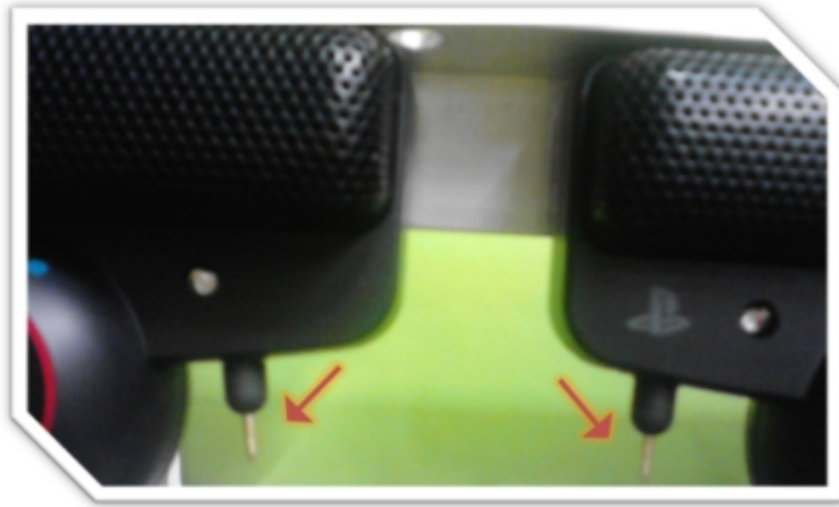


Figura 4.5: Par de PS3 EyeCam após modificação

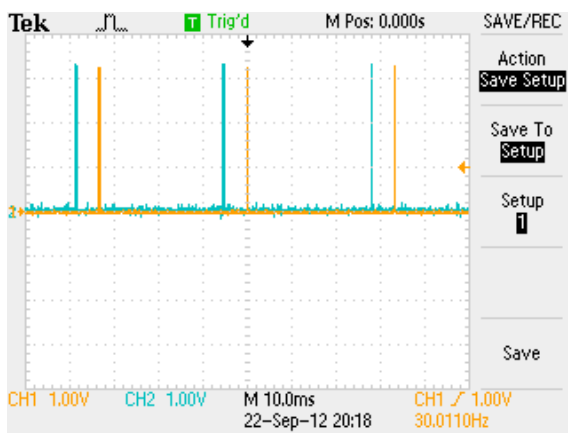


Figura 4.6: Câmaras Não Sincronizadas - Sinais VSYNC de ambas

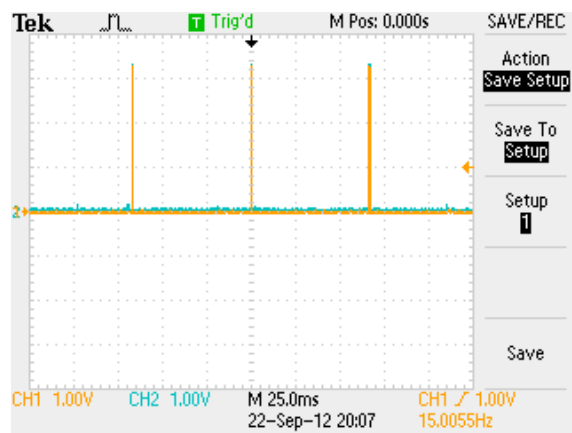


Figura 4.7: Câmaras Sincronizadas - Sinais VSYNC de ambas

No par não sincronizado (Figura 4.8) é visível, logo no primeiro *frame*, que a aquisição na câmara esquerda ocorreu ligeiramente mais cedo do que na direita. Adiante, no *frame* 16, os instantes de aquisição são já bem distintos.

Com as câmaras sincronizadas, (Figura 4.9), os instantes de aquisição são em cada *frame*, como esperado, precisamente os mesmos para as duas câmaras.

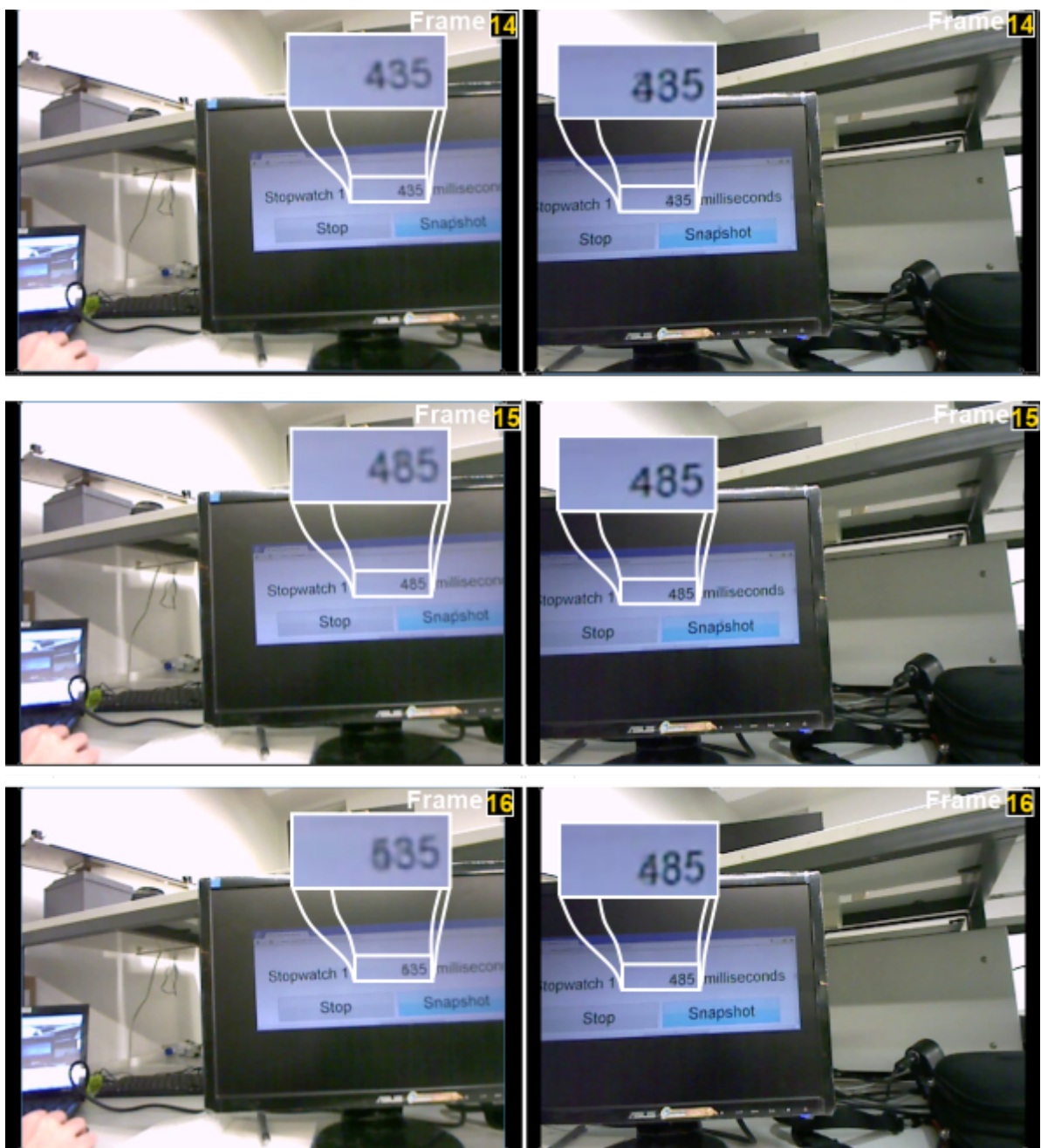


Figura 4.8: Aquisição com par não sincronizado (sequência)

4.2.3 Sensor de Ultrassons

O sensor de ultrassons HC-SR04 (Figura 4.10) encontra-se ligado a uma entrada digital do Raspberry Pi e funciona da seguinte forma:

- Um pulso de pelo menos 10 *us* é fornecido à entrada *trigger*
- O emissor envia 8 pulsos de 40 KHz e o recetor espera recebê-los de volta

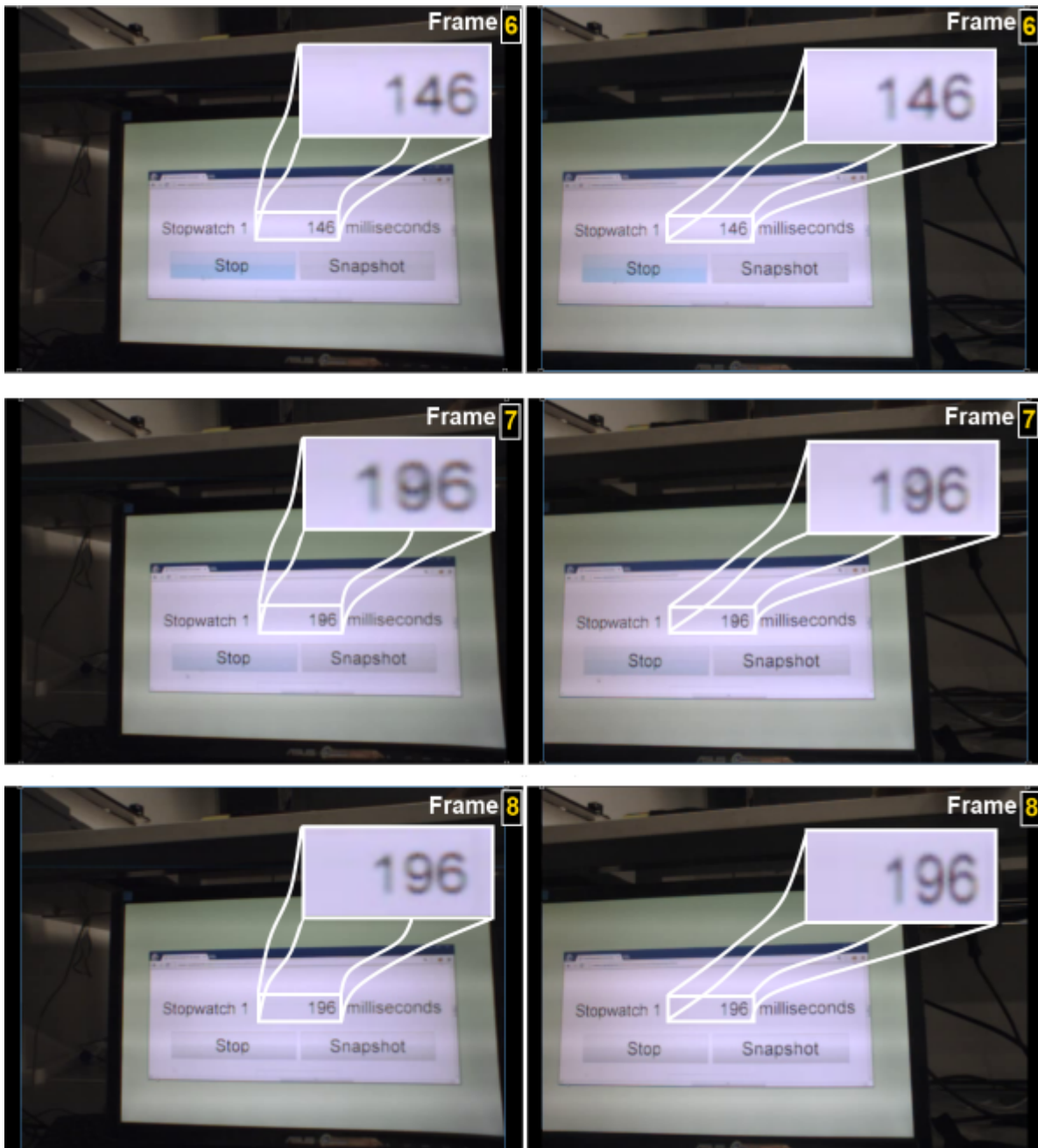


Figura 4.9: Aquisição com par sincronizado (sequência)

- Se houver retorno, a saída *Echo* fica no nível alto por durante o tempo de viagem do sinal
- A distância a que se encontra o objeto detetado é igual a $\frac{\text{tempo}_{\text{nível alto}} * V_{\text{som}}}{2}$

Este modelo é alimentado com um nível de tensão de 5V e comumente utilizado em aplicações envolvendo sistemas Arduino.

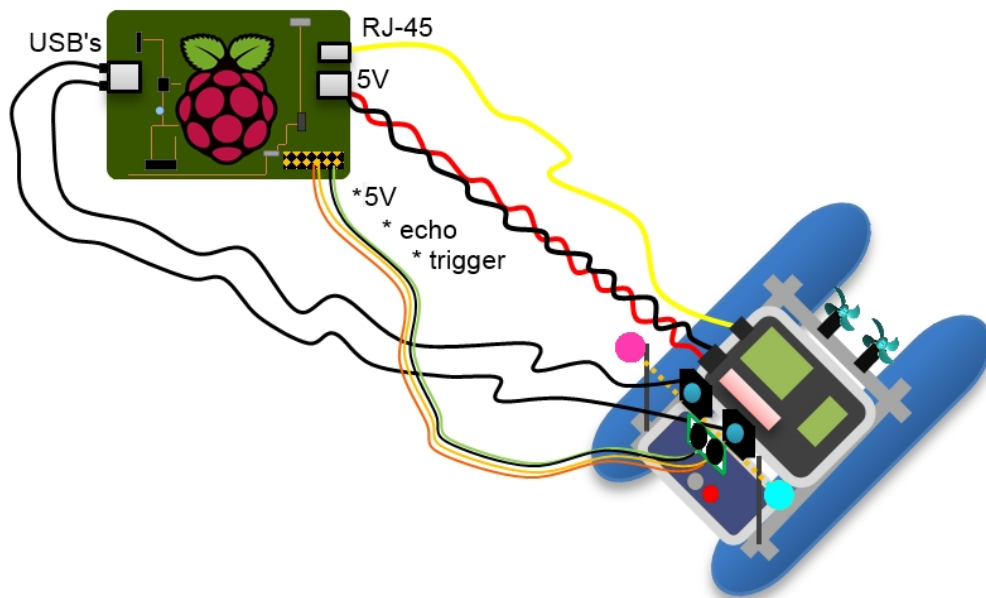


Figura 4.10: Sensor HCSR04

4.2.4 Montagem Final

A estrutura final do sistema e respetivas ligações são apresentadas na Figura 4.11. Por motivos explicados na secção 6.1 foi decidido instalar o Raspberry PI num compartimento separado do restante *hardware* da embarcação. Assim, a comunicação feita entre as duas unidades computacionais via ligação *ethernet* bem como a alimentação do Raspberry PI fez-se através das tomadas estanques existentes na caixa hermética do ASV.

Com esta montagem pode fazer-se aquisição de pares de imagem de $320 \times N$ pixels (N é a altura desejada), algo que, com os algoritmos utilizados, pode ocorrer a uma taxa de 2 ou 3 fps.

Figura 4.11: Montagem e ligações entre *hardware*

4.3 Software

4.3.1 Aquisição e Processamento de Imagem

O OpenCV, através das suas funções de captura de imagem, provou-se ineficaz na aquisição de pares de imagem sincronizados. Quer utilizando uma captura ininterrupta de vídeo, quer tentando controlar a aquisição de cada par de frames, despoletando-a em momentos precisos, a performance dessas funções evidenciou sempre um comportamento insatisfatório, chegando a provocar desfazamentos de várias dezenas de ms entre as imagens do par. A resolução do problema acabou por se fazer por via de uma API de Linux para controlo e acesso a dispositivos Video, o *Video4Linux2*.

4.3.1.1 Video4Linux2

O Video4Linux2 [49] é uma interface desenvolvida pela comunidade Linux que pretende unificar o acesso a dispositivos video. No presente, grande parte das câmaras disponíveis têm compatibilidade com V4L2. Com esta API, pode fazer-se troca de equipamentos sem necessidade de modificação do código de aquisição de imagem, dado que ela negocia com o hardware a forma de operar, tendo por base as características de cada câmara compatível. Por norma, os programas de aquisição devem seguir uma estrutura como a seguinte:

- *Abrir o dispositivo*
- *Negociação de propriedades*
- *Negociação de formato de imagem*
- *Método de I/O*
- *Loop de execução de código sobre a imagem*
- *Fechar o dispositivo*

Com o Video4Linux2 é possível controlar todo o tipo de parâmetros da câmara ([50] e [51]), desde que disponíveis no hardware. A título de exemplo, é possível controlar-se o ganho, o tempo de exposição ou o tipo de focagem.

Embora de início o kernel da versão *Debian* disponibilizada para o Raspberry PI não trouxesse o V4L2 compilado, esse problema resolveu-se nas versões mais recentes.

A forma de resolver o problema dos atrasos entre frames foi fazendo acessos ao *buffer* de ambas as câmaras quase simultâneos, lendo a informação contida em ambos e convertendo-a numa mesma função, de YUYV para RGB. A informação saída do buffer das *webcams* vem no formato YUV:422 (Figura 4.12), um formato que separa a informação de cor da informação de brilho e é comumente usado em *pipelines* de video. A função apresentada no segmento de código abaixo converte sequências de 4 bytes de cada câmara para dois valores RGB, i.e. 6 bytes. Esta função prevê a possibilidade de descodificação de áreas limitadas da imagem para uma dada resolução:

escolhido um tamanho de imagem (*largura*, *altura*), pode definir-se um valor de *offset* para tratar apenas uma parte da altura da imagem. Justifica-se esta possibilidade pela aplicação e sistema computacional a que se destina, tendo em conta que (1) qualquer operação global efetuada sobre a imagem consome, no mínimo, um tempo diretamente proporcional à dimensão da imagem tratada e que (2) a finalidade do sistema é principalmente identificar objetos que se encontram abaixo da linha do horizonte, por norma próxima do meio da altura da imagem.

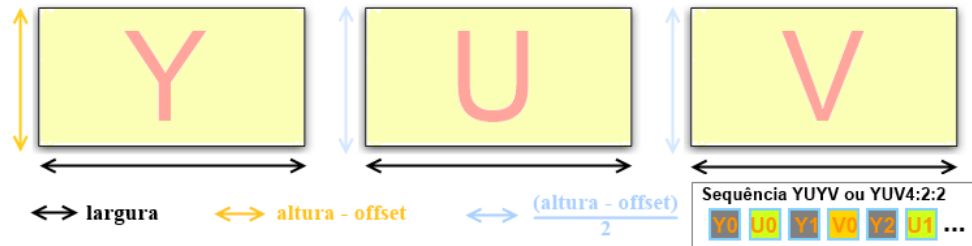


Figura 4.12: YUV

```

1
2 void YUV2RGB (char *buffer1, char *buf_rgb1, char *buffer2, char *buf_rgb2)
3 {
4     int x, y;
5     for(y = offset; y < altura; ++y) {
6         for(x = 0; x < largura; x+=2) {
7
8             int i = (largura * y + x);
9
10            int y0 = buffer1[2*i]; // camara esquerda
11            int u = buffer1[2*i+1];
12            int y1 = buffer1[2*i+2];
13            int v = buffer1[2*i+3];
14            int y0_2 = buffer2[2*i]; //camara direita
15            int u_ = buffer2[2*i+1];
16            int y1_2 = buffer2[2*i+2];
17            int v_ = buffer2[2*i+3];
18
19            int r1 = (y0-16)*1164/1000 + (v-128)*1596/1000; // conversao
20            int r1_ = (y0_2-16)*1164/1000 + (v_-128)*1596/1000;
21            int r2 = (y1-16)*1164/1000 + (v-128)*1596/1000;
22            int r2_ = (y1_2-16)*1164/1000 + (v_-128)*1596/1000;
23
24            int g1 = (y0-16)*1164/1000 - (v-128)*813/1000 - (u-128)*391/1000;
25            int g1_ = (y0_2-16)*1164/1000 - (v_-128)*813/1000 - (u_-128)*391/1000;
26            int g2 = (y1-16)*1164/1000 - (v-128)*813/1000 - (u-128)*391/1000;
27            int g2_ = (y1_2-16)*1164/1000 - (v_-128)*813/1000 - (u_-128)*391/1000;
28
29            int b1 = (y0-16)*1164/1000 + (u-128)*2018/1000;
30            int b1_ = (y0_2-16)*1164/1000 + (u_-128)*2018/1000;
31            int b2 = (y1-16)*1164/1000 + (u-128)*2018/1000;
32            int b2_ = (y1_2-16)*1164/1000 + (u_-128)*2018/1000; // fim da conversao
33
34            LIMITADOR(r1) //limita valores entre 0..255. Sem este controlo,
35            LIMITADOR(g1) // muitos pixels da imagem aparecem saturados.
36            LIMITADOR(b1)
37            LIMITADOR(r2)
38            LIMITADOR(g2)
39            LIMITADOR(b2)
40            LIMITADOR(r1_)
41            LIMITADOR(g1_)
42            LIMITADOR(b1_)
43            LIMITADOR(r2_)
44            LIMITADOR(g2_)
45            LIMITADOR(b2_)
46
47            buf_rgb1[3*i] =b1; //frame 1 - RGB

```

```

48     buf_rgb1[3*i+1]=g1; //
49     buf_rgb1[3*i+2]=r1; //
50     buf_rgb1[3*i+3]=b2; //
51     buf_rgb1[3*i+4]=g2; //
52     buf_rgb1[3*i+5]=r2; //
53
54     buf_rgb2[3*i]  =b1_; //frame 2 - RGB
55     buf_rgb2[3*i+1]=g1_; //
56     buf_rgb2[3*i+2]=r1_; //
57     buf_rgb2[3*i+3]=b2_; //
58     buf_rgb2[3*i+4]=g2_; //
59     buf_rgb2[3*i+5]=r2_; //
60 }
61 }
62 }

```

4.3.1.2 Extração de Características - CVBlobs

O CVBlobs é uma biblioteca de OpenCV destinada à extração de *blobs* (objetos binários de grande dimensão) de conectividade 8 (Figura 4.13) em imagens binárias ou de escala de cinzento. Com esta biblioteca é possível filtrar, manipular e extrair diversas características de cada objeto com estas características presente na imagem.

A análise do contorno de cada objeto é feita utilizando vizinhança 8, isto é, considerando os 8 pixels em torno do pixel do contorno em análise. Desta forma, um pixel é candidato a pertencer ao contorno do objeto se é adjacente, em qualquer das 8 direções, ao pixel em análise e as suas características são similares (frequentemente, se a sua cor for próxima da do analisado). É também comum utilizar-se conectividade 4, que tipicamente analisa os pixels com coordenadas $(0,-1);(1,0);(0,1);(-1,0)$ da Figura 4.13. Este método produz contornos mais espessos: no exemplo da figura, teria sido incluído no contorno o pixel $(0,1)$.

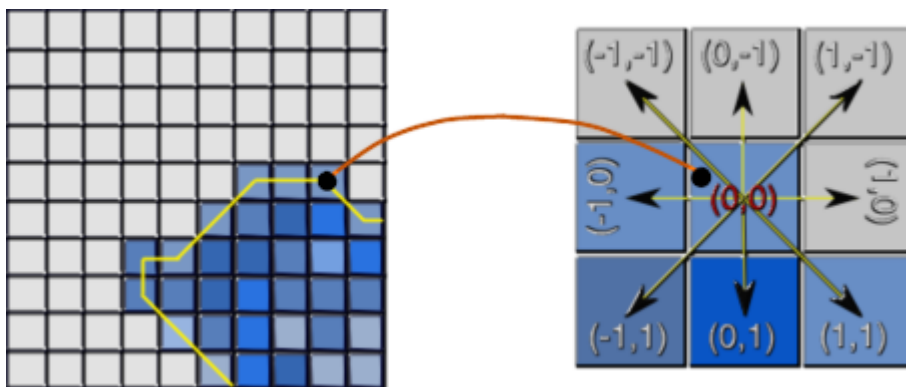


Figura 4.13: Detecção de Contorno - Conectividade 8

Esta biblioteca foi utilizada para tentar a detecção de boias. Para esse efeito, desenvolveu-se um primeiro algoritmo que segue os passos apresentados na Figura 4.14, programa que explora apenas a perspetiva da visão monocular mas que viria, mais tarde, a servir de base ao programa final.



Figura 4.14: CvBlobs - Sequência de Exec. no Algoritmo de Det. de Boias

Foi utilizada uma imagem da piscina de testes com resolução de 320x240 pixels (Figura 4.15 a)), com uma envoltória repleta de equipamentos e materiais sem relevância. Na piscina, entre outros, encontravam-se os objetos de interesse - as bolas. Esta resolução foi escolhida quer pelo custo computacional do tratamento de imagens de maior dimensão, quer por problemas de sincronismo detetados numa das câmaras do par aquando das tentativas de aquisição com resolução superior. A análise da imagem faz-se apenas na metade inferior da imagem (Figura 4.15 b)) onde estarão as bolas (boias) a detetar . Desta forma, e porque será necessário fazer sucessivos varrimentos à imagem, economiza-se tempo de processamento.

A conversão de cor faz-se por motivos práticos. Note-se o objeto da Figura 4.16 cuja cor à primeira vista facilmente identificamos como sendo azul. Se for considerado o sistema de cores RGB para fazer a sua deteção, veja-se que a variação em cada um dos três canais cobre cerca de 60% do intervalo. Desta forma, bastará um cenário de pouca complexidade para que a correta deteção dos seus limites seja de extrema dificuldade sendo de todo impossível se for considerado um cenário real. Pelo contrário, observando os valores no sistema HSV, identifica-se uma ampla variação nos canais S (saturação) e V (valor) sendo contudo notório que o canal H mantém um valor praticamente imutável entre os dois limites testados (varia 0.5% em todo o objeto). É justamente esta característica deste sistema de cores que nos permitirá, adiante, identificar com maior fidelidade os objetos de interesse¹.

A definição de *thresholds* tem de ser ajustada ao ambiente de funcionamento do sistema. Por esse motivo, cada nova cor de boia a detetar implica a chamada da função `deteta_boias()` . Esta função recebe como parâmetros a imagem e quatro parâmetros de *threshold*: `min,max,rmin,rmax`. Cada um destes parâmetros contém três elementos - H, S e V - e está armazenado numa estrutura

¹Numa nota paralela, referir que alguns sistemas de visão são redundantes e utilizam, na deteção de objetos, *thresholds* em mais do que um sistema de cores .

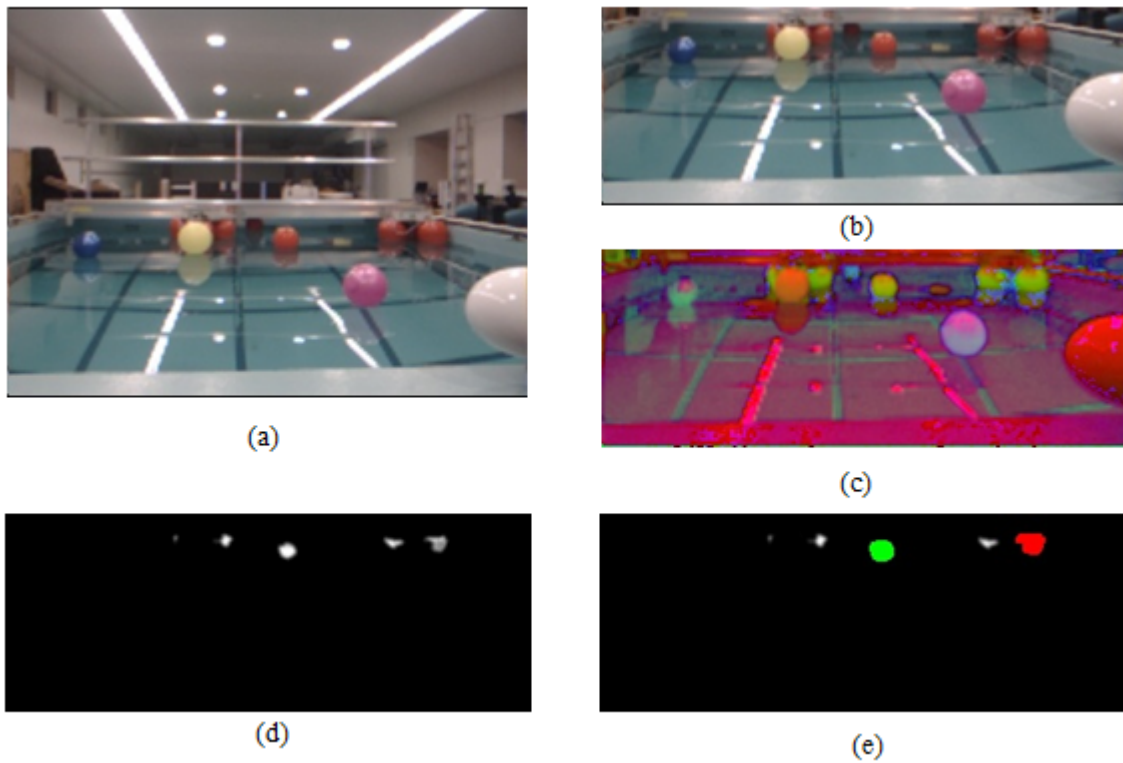


Figura 4.15: CvBlobs - Resultados Parciais do Algoritmo de Det. de Boias

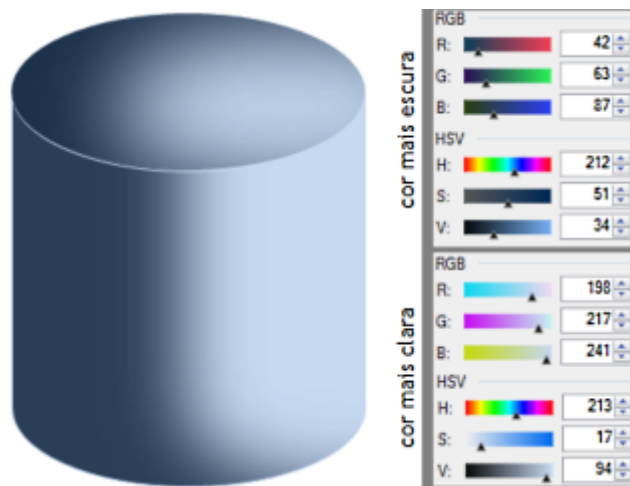


Figura 4.16: Objeto com Sombra

cvScalar. A necessidade de quatro parâmetros de *threshold* prende-se com a repartição dos vermelhos pelos dois extremos do intervalo da matiz (H) (ver Figura 4.16). Este problema poderia ter sido resolvido de outra forma: entendendo o H como um valor angular sobre uma circunferência, adotando um sentido de rotação (por exemplo, direto) e definindo o princípio de que o primeiro

valor, num par de valores de *threshold*, corresponde ao início de um intervalo angular e o segundo ao fim desse mesmo intervalo (por exemplo [330, 10] corresponde a um intervalo de 40°).

Na Figura 4.15 d), apresenta-se a imagem binarizada da pesquisa de boias vermelhas. No instante da aquisição da imagem original, outros objetos da mesma cor estavam presentes no tanque e apenas uma boia vermelha. Vários desses objetos surgem ainda representados após a binarização. Uma análise de características de redondez, rugosidade, perímetro, área, alongação, *bounding box* e diversas razões entre medidas de cada um deles permitem discriminar os que são realmente boias. O algoritmo faz essa distinção preenchendo a verde as manchas das boias e a vermelho outros objetos (Figura 4.15 e). Objetos de pequena dimensão são filtrados pela função *Filter* da biblioteca. A experiência de utilização desta biblioteca fez-se com o fulcro de se obter o centro de cada um dos objetos de interesse, para posterior utilização no âmbito da estereoscopia. As classes *CBlobGetXCenter* e *CBlobGetYCenter* fornecem essa informação diretamente. Algumas funções importantes utilizadas no algoritmo apresentam-se e descrevem-se na Tabela 4.1

<i>CBlobGetMajorAxisLength()</i>	Classe que calcula o comprimento do eixo maior da elipse que envolve o objeto.
<i>CBlobGetMinorAxisLength()</i>	Classe que calcula o comprimento do eixo menor da elipse que envolve o objeto.
<i>CBlobGetOrientation()</i>	Classe que devolve a orientação, em radianos, da elipse que envolve o objeto.
<i>CBlobGetRoughness()</i>	Classe que devolve a rugosidade do objeto, i.e., a razão entre o perímetro e o perímetro das saliências do objeto.
<i>CBlobGetElongation()</i>	Classe que calcula a alongação do objeto, i.e., a razão entre comprimento e largura
<i>CBlobGetArea()</i>	Classe que devolve a área do objeto.
<i>CBlobGetPerimeter()</i>	Classe que devolve o perímetro do objeto.
<i>GetBoundingBox().width</i>	Obtém a largura do retângulo envolvente do objeto. Igualmente válido para a altura (<i>GetBoundingBox().height</i>)

Tabela 4.1: Funções da Biblioteca CvBlobs Utilizadas

4.3.1.3 Determinação de *thresholds*

Para definir os *thresholds* de cada uma das cores a identificar, foi desenvolvido um *script* em *Matlab*.

Dado que as condições de iluminação no ambiente de funcionamento são variáveis, para melhorar a definição dos limites de *threshold* adquirem-se várias imagens do objeto a seguir, exposto a condições de iluminação distintas. Posteriormente, converte-se o espaço de cores das imagens utilizando um algoritmo que incorpora a função **cvCvtColor()** de OpenCV e faz conversão de RGB para HSV. Numa terceira fase, usando um editor de imagem extrai-se de cada foto tirada a zona que contém o objeto de interesse, conservando alguma distância aos limites do mesmo para

evitar a inclusão de cores indesejadas. As novas imagens, contendo apenas o objeto de interesse, são submetidas ao *script* que devolve os histogramas de Matiz, Saturação e Valor (Figura 4.17) e uma tabela com máximos e mínimos, obtidos considerando valores que surjam mais do que um dado número de vezes em torno da zona de maior concentração do histograma (para evitar picos).

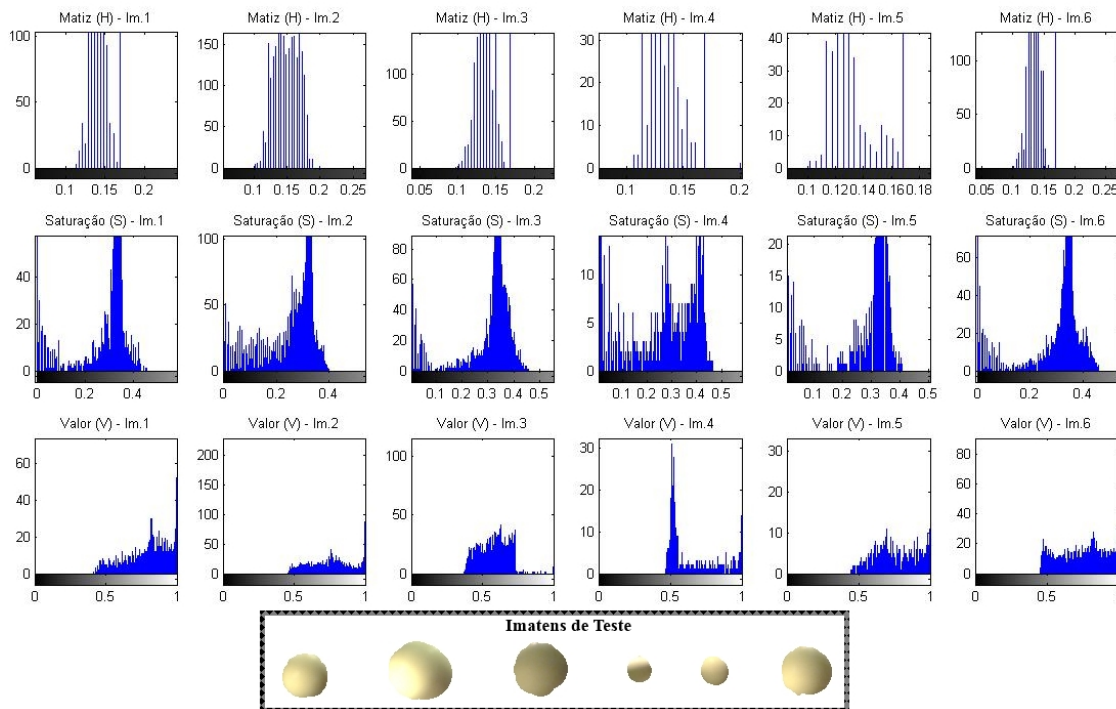


Figura 4.17: Imagens teste e Histogramas

4.3.1.4 Reconversão de mapas - remapeamento de pontos

Para ser possível trabalhar as imagens sem distorção o OpenCV permite, através da função *cvInitUndistortRectifyMap*, a geração de mapas de remapeamento, conforme referido na secção sobre o processo de calibração. Estes mapas servem para uma reconstrução completa da imagem, sem distorções (Figuras 4.18 e 4.19).

Os mapas gerados, **mx** e **my**, são armazenados sob forma de matrizes bidimensionais com as dimensões da imagem a remapear. Cada elemento **mx[i,j]** guarda um valor que representa a coordenada vertical do pixel a copiar da imagem original, reposicionando-o, na nova imagem, com as coordenadas [i,j]. O mesmo acontece com os elementos do mapa **my[i,j]**, que guardam a coordenada horizontal.

A Figura 4.20 descreve este processo.



Figura 4.18: Imagem original



Figura 4.19: Imagem remapeada

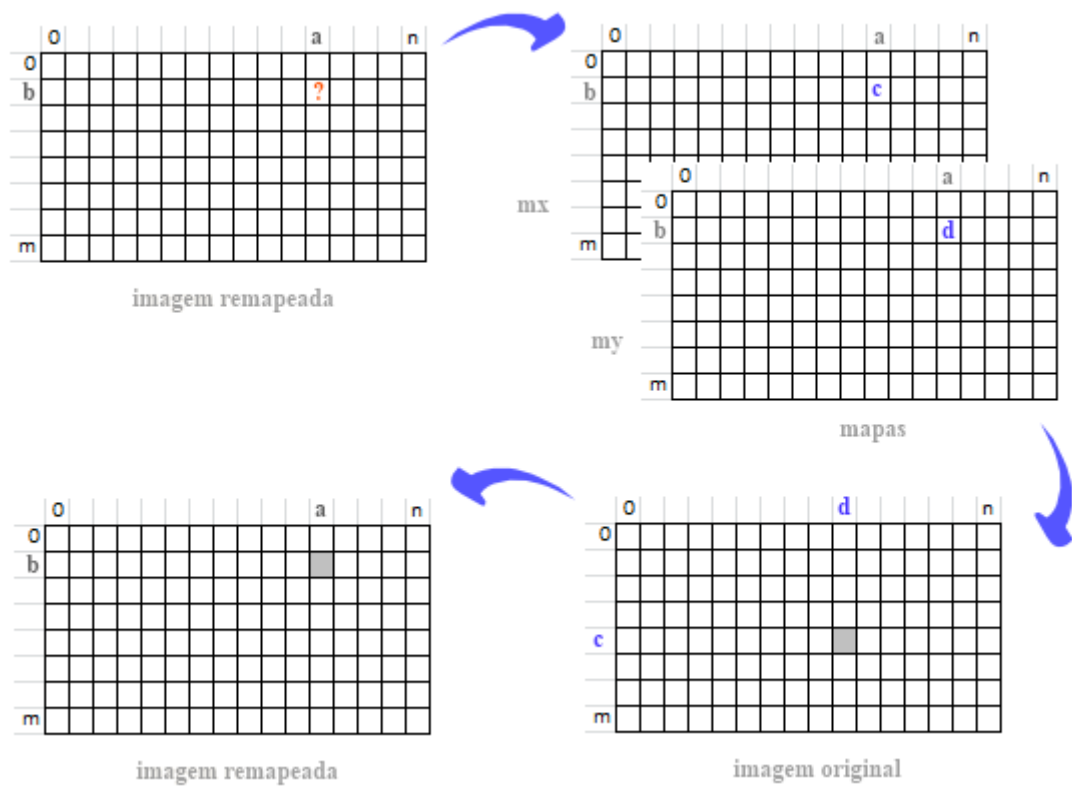


Figura 4.20: Processo de remapeamento de imagens

O processamento de imagem é extremamente exigente do ponto de vista computacional. Tendo em atenção a decisão de implementar o sistema de aquisição e processamento de imagem num Raspberry PI, importa ter especial preocupação com a utilização pouco cuidada de recursos.

Esta conversão de imagens é feita com base na função *cvRemap*, que faz o varrimento de toda a imagem, recriando-a sem deformações.

A distância a medir é entre o centro de massa do objeto de interesse e o referencial do par estereoscópico da embarcação. Não estando a cena captada repleta de objetos de interesse, donde

é desnecessário o remapeamento de todos os pixels do par de imagens, uma forma de economizar tempo de processamento é substituir o processo de remapeamento de ambas as imagens por outro que apenas corrija as coordenadas do centro de cada objeto de interesse. Para esse efeito, foi criada a função *convertemapas()*, que reorganiza os mapas **mx** e **my**: novos mapas, **mRx** e **mRy**, contêm em cada uma das suas posições as coordenadas corrigidas dessas mesmas posições.

A Figura 4.21 ilustra a transformação.

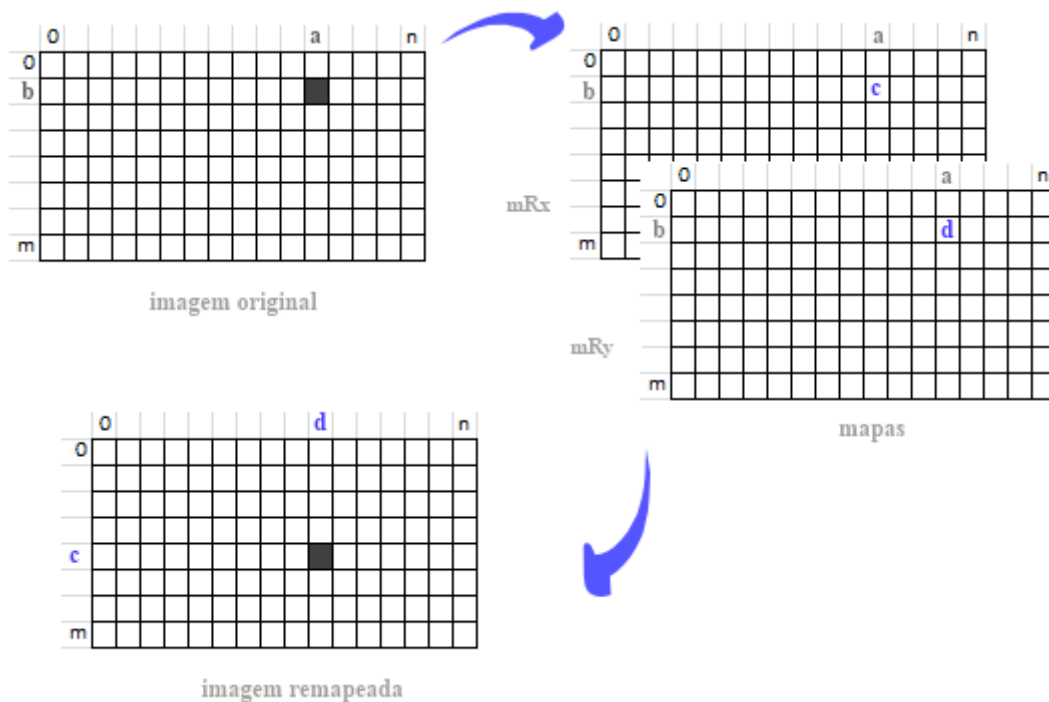


Figura 4.21: Remapeamento de pontos

O resultado desta conversão gera por vezes zonas de vazio. Se atentarmos à figura 4.22 a), em que a intensidade da cor do mapa é proporcional à coordenada (+ branco para coordenadas maiores), nota-se que surge uma espécie de grelha, reflexo do processo de calibração e eliminação de distorções nos mapas originados pela função *cvRemap*. Isto também acontece porque os mapas utilizados provindos da função *cvRemap* eram mapas com valores float; os de destino, são mapas com valores inteiros. A interpolação (Figura 4.22 b) teve de se fazer para impedir situações em que o centro de um qualquer objeto detetado pelo par estereoscópico ocorresse num desses pontos, gerando coordenadas (0,y) ou (x,0) erradamente. O contorno do mapa interpolado evidencia ainda indícios da grelha, devendo-se isso à dimensão do filtro aplicado no processo de interpolação.

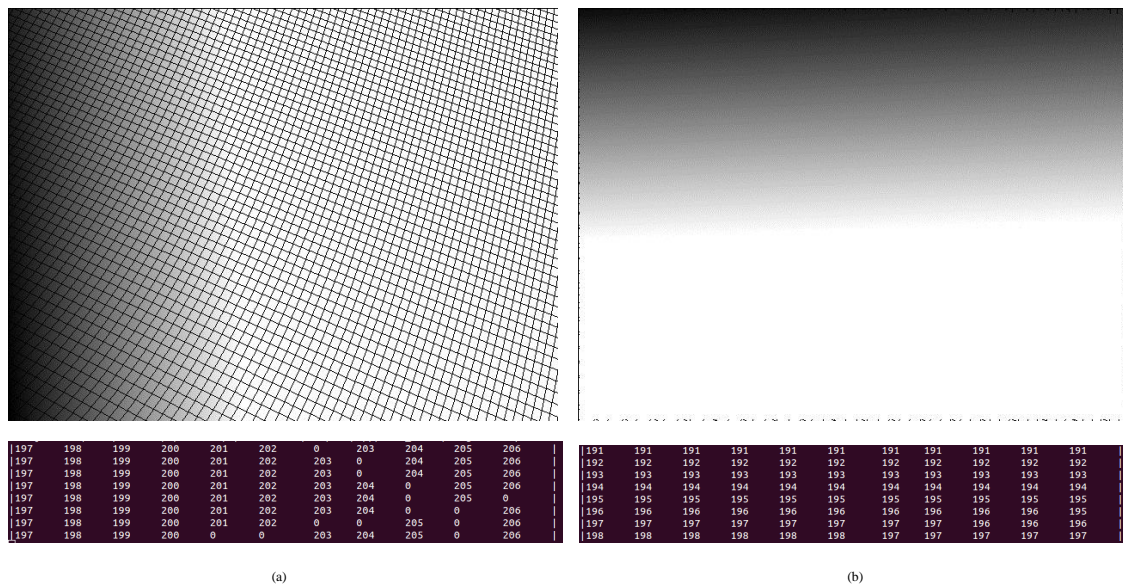


Figura 4.22: Resultados da função *convertmapas* - a) sem e b) com interpolação

4.3.1.5 Programa de seguimento e detecção de boias

O algoritmo final segue o esquema de funcionamento representado na Figura 4.23.

Recebe como parâmetros os seguintes valores:

- **Temp_min_frames** - Limita inferiormente o tempo entre aquisições de pares de frames (int)
- **Tolerância epipolar** - Tolerância para correspondência Direita - Esquerda, ΔY ; máxima diferença de coordenada vertical admitida entre objetos correspondentes (int)
- **Offset de captura** - Define a altura da captura ($h = \text{resolução vertical} - \text{offset}$) (int)
- **A_Orientação** - Fator aditivo de correção de valores de orientação (int)
- **M_Orientação** - Fator multiplicativo de correção de valores de orientação (int)
- **A_Distância** - Fator aditivo de correção de valores de distância (int)
- **M_Distância** - Fator multiplicativo de correção de valores de distância (int)
- **Modo Gráfico** - Variável booleana para ativação/desativação da apresentação de janelas de cena (bool)
- **Dist_Obstáculo** - Distância mínima de perigo (em metros)- sensor de ultrassons (float)

Importa aqui principalmente detalhar o funcionamento do algoritmo de detecção de boias e seguimento. Apenas referir que enquanto ele corre, o processo de aquisição (e qualquer outro que pudesse estar a ser executado) está impedido de modificar o conteúdo do par de *frames* direita e

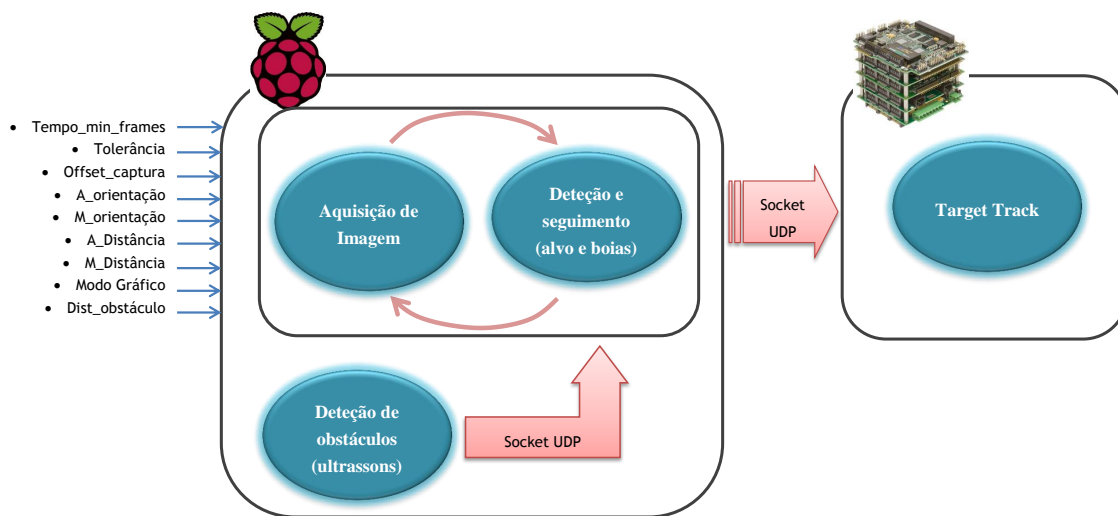


Figura 4.23: Algoritmo de seguimento e detecção de boias

esquerda em utilização, por força de o seu funcionamento ocorrer dentro de um *mutex* (Pthreads, C). a função de detecção de alvo e boias recebe como parâmetros os seguintes valores:

- **Frame Esquerda** - Imagem adquirida pela câmara esquerda (IplImage*)
- **Frame Direita** - Imagem adquirida pela câmara direita (IplImage*)
- **Threshold (mínimo)** - Valor Mínimo de *threshold* (HSV) (CvScalar)
- **Threshold (máximo)** - Valor Máximo de *threshold* (HSV) (CvScalar)
- **Threshold Vermelho (mínimo)** - Valor Mínimo de *threshold* para tons vermelhos (HSV) (CvScalar)
- **Threshold Vermelho (máximo)** - Valor Máximo de *threshold* para tons vermelhos (HSV) (CvScalar)
- **Forma** - Identificador de forma (0 - Alvo a seguir, 1 - Boia Direita, 2 - Boia Esquerda) (int)

O funcionamento da função está descrito no esquema da Figura 4.24. Para cada par de *frames* adquirido, a função *detetaalvoboia()* é executada até 3 vezes. No mínimo, tem de haver uma execução para forma=0 (alvo a seguir). Pode ainda executar-se até mais duas vezes, tendo em conta boias delimitadoras de um dado trilho a seguir. Para forma=1, interpretará as deteções como boias direitas, a serem contornadas pela esquerda; para forma=2, assume que objetos são boias delimitadoras esquerdas, a contornar pela direita. No final de cada execução destas funções, o sistema confirma a informação mais recente do sensor de ultrassons: havendo deteção de obstáculos a uma distância inferior a *Dist_Obstáculo*, ainda que tenha sido detetado um alvo a seguir, a informação de alvo contida na estrutura "**cena**", a enviar ao programa *target_track()*, é substituída por um

valor interpretado pelo ASV como ordem de inversão de sentido de rotação aos motores, de forma a impor uma paragem rápida do veículo.

Abaixo expõe-se a estrutura de informação trocada entre o Raspberry PI e a embarcação.

```
1 struct cena {  
2     char target_flag;  
3     double target_angle;  
4     double target_distance;  
5     char right_flag;  
6     double right_angle;  
7     double right_distance;  
8     char left_flag;  
9     double left_angle;  
10    double left_distance;  
11 } atual;
```

4.3.1.6 Target Track

O programa Target Track é a interface entre o Raspberry PI e a *framework* em que se baseia o funcionamento dos veículos do OceanSys.

Os algoritmos em funcionamento no Raspberry PI geram um fluxo contínuo de informação de orientação e distância de objetos de interesse, armazenados na estrutura de dados "cena", que existe de ambos os lados.

O Target Track mantém em permanência uma *thread* a receber essa informação, gerando com esses dados uma referência de erro de posição e orientação para a embarcação.

A *framework* em funcionamento nos ASV's tem uma camada de controlo organizada da seguinte forma:

- Dispatcher: responsável pelo controlo da missão. Recebe, envia e toma decisões sobre a manobra a executar em cada instante de tempo. É com este programa que outros processos devem interagir para enviar comandos à camada de controlo.

- Controlo: recebe instruções do Dispatcher sobre a manobra a executar em cada momento. Sob autorização do Dispatcher, pode receber referências de pose (posição+ângulos), de velocidades, de erros de velocidades/poses ou combinações de todas estas. Essas referências são tipicamente passadas pela estrutura MotionRef ou através de strings que o Dispatcher lhe envia. Essas referências entram nos loops de controlo que se encarregam de fazer convergir o sistema. Neste nível, existem controladores de posição e de velocidade que usam o modelo cinemático e dinâmico do veículo, respetivamente, para gerar os comandos para os motores.

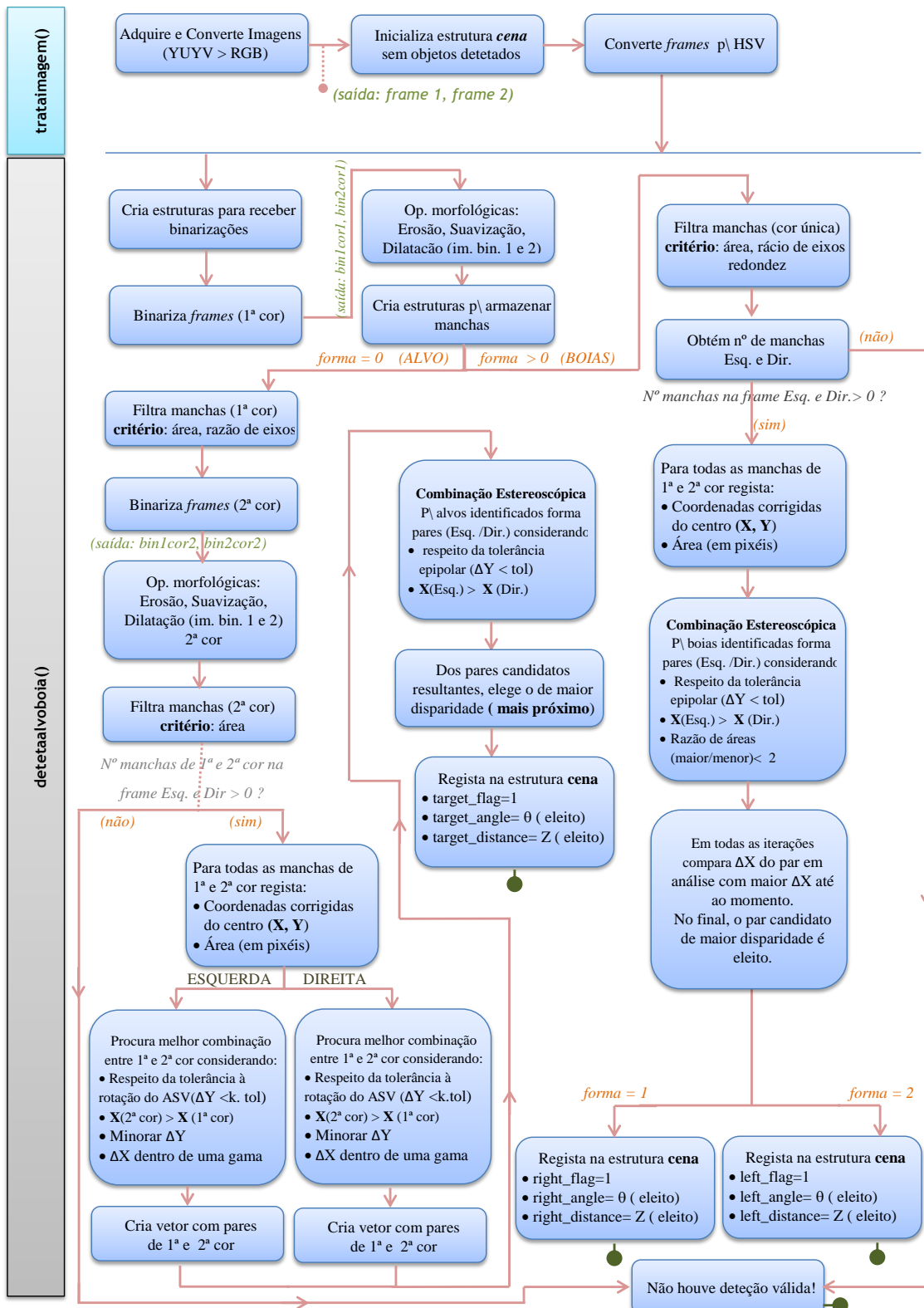


Figura 4.24: Sequência do algoritmo de seguimento de alvo e deteção de boias

4.3.2 Outros algoritmos

4.3.2.1 Transformada de Hough

A Transformada de Hough é um método que permite a detecção de formas que possam descrever-se através de equações paramétricas. Na sua forma mais elementar permite a detecção de linhas embora sejam também muito comuns implementações para detecções de círculos e elipses.

No caso da detecção de linhas, a ideia da Transformada de *Hough* é de que "qualquer dos pontos de uma imagem binária pode fazer parte de um conjunto de linhas" [45]. Os pontos do referencial cartesiano são transformados para o plano de *Hough*, também chamado de plano acumulador. Atente-se na imagem 4.25:

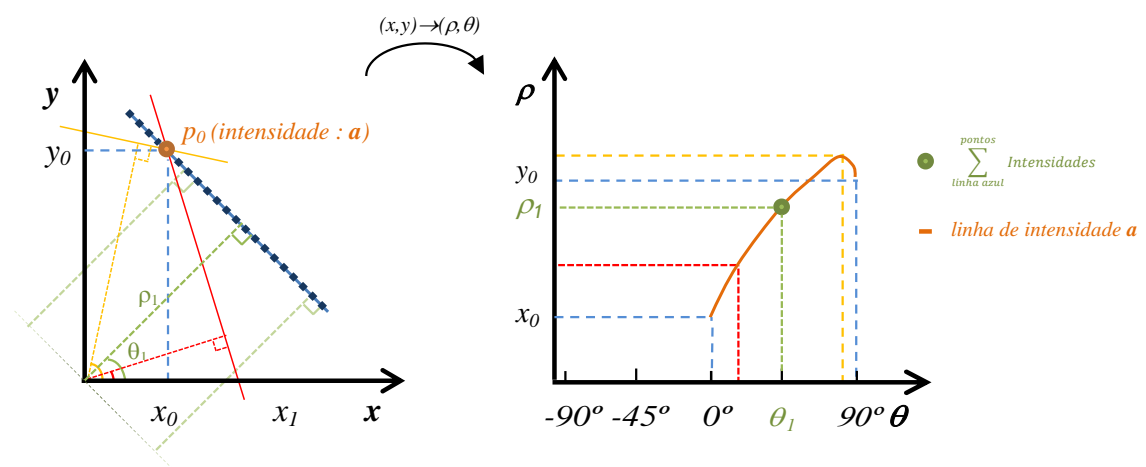


Figura 4.25: Transformada de *Hough* (pontos e linhas)

No caso da transformação de pontos e linhas, o plano de *Hough* é bidimensional. Os parâmetros ρ e θ correspondem, respetivamente, ao comprimento e ao ângulo (com o eixo das ordenadas) da linha perpendicular- linha verde - à linha em análise -linha azul - no plano (x,y) que une a uma outra, também perpendicular, que cruza a origem do referencial. Todos os pontos que se encontrem sobre uma mesma linha no plano cartesiano são convertidos num unico ponto no plano de *Hough*, com um valor de intensidade que resulta das contribuições individuais de todos eles; em contrapartida, pontos no plano x,y são transformados em linhas no plano de *Hough*.

Para o trabalho desenvolvido, interessou testar o funcionamento da Transformada de *Hough* na detecção de círculos. O princípio de funcionamento é idêntico ao descrito acima embora haja um acréscimo na exigência computacional para este caso, pela existência de um terceiro parâmetro que, em princípio, obrigaria o plano acumulador do caso anterior a transformar-se num espaço acumulador. O problema é normalmente resolvido fixando o raio e tratado, novamente, no plano. Para um raio r , todos os pontos da circunferência no plano (x,y) passam a ser o centro de uma nova circunferência no plano (a,b) ; o uso de *thresholds* permite detetar o centro do círculo original (indicado a amarelo na Figura 4.26), tipicamente um máximo, por via da acumulação de contributos de todas essas circunferências.

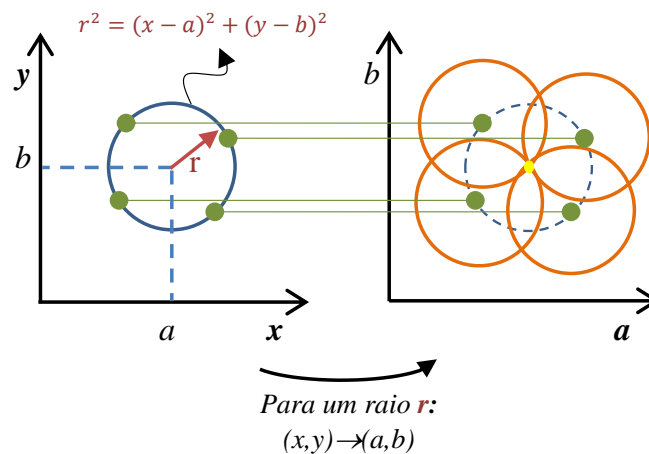


Figura 4.26: Transformada de *Hough* (círculos)

O OpenCV dispõe de implementações da transformada de Hough, quer para linhas, quer para círculos. No segundo caso, no sentido de evitar ter de trabalhar em R^3 , OpenCV implementa um método de gradiente, detalhadamente descrito em [45]. Para garantir eficiência no funcionamento, esta implementação permite ao utilizador que defina raios máximos e mínimos dos círculos a detetar, sendo o custo computacional da sua execução função da amplitude deste intervalo. Como desde início vêm sendo utilizadas para o trabalho bolas e boias como elementos presumivelmente adjuvantes ao cumprimento de trajetórias, foi desenvolvido um algoritmo baseado na função *cvHoughCircles* cuja sequência se encontra descrita na Figura 4.27.

O algoritmo desenvolvido utiliza os mapas convertidos segundo a descrição da Secção 4.3.1.4. São definidos logo de início valores de *threshold* para as cores a detetar e importados os mapas resultantes da conversão - dois por câmara - e a matriz \mathbf{Q} proveniente do processo de calibração.

Para cada par de imagens adquirido, o tratamento das imagens esquerda e direita é feito em paralelo. A função *cvHoughCircles* é aplicada individualmente e armazena os resultados numa estrutura do tipo *cvSeq* de OpenCV. Na estrutura, são guardadas ordenadamente para cada círculo as coordenadas do centro e respetivo raio. É possível definir-se, conforme referido acima, raios mínimo e máximo dos círculos a detetar, distância mínima entre círculos e limites de *threshold* para imagens de 8bits. Esta função lida com imagens em escala de cinzento mas a imagem submetida é binária; o que motiva a binarização prévia das imagens partindo do formato HSV é a necessidade de garantir que apenas bóias das cores definidas são incluídas no processo de deteção de círculos.

Partindo das coordenadas do centro de cada círculo detetado, recorre-se aos mapas convertidos (\mathbf{mRx} e \mathbf{mRy}) para obter as coordenadas corrigidas do centro do círculo. O passo seguinte é a obtenção das coordenadas tridimensionais de cada círculo bem como a sua orientação face ao referencial do par estereoscópico, utilizando a informação da matriz \mathbf{Q} e das relações enunciadas em 3.9 e tendo em conta as restrições epolares. Uma boia que surja na imagem esquerda na coordenada (x_1, y) tem coordenadas (x_2, y) na imagem direita. Os resultados da execução do algoritmo apresentam-se na Figura 4.28.

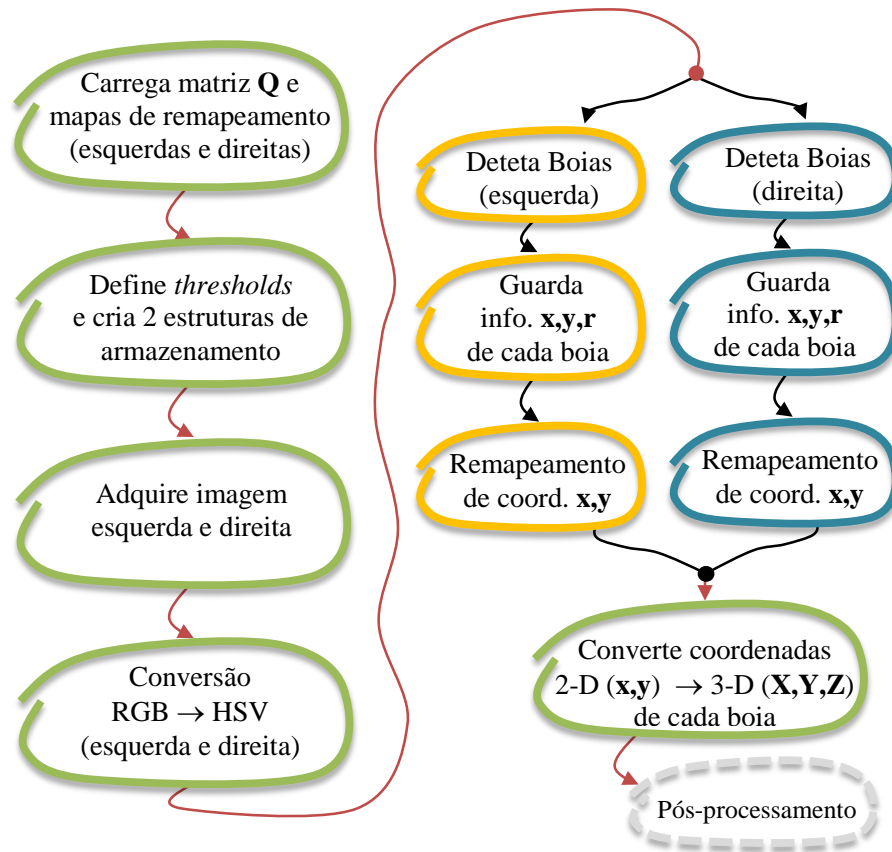


Figura 4.27: Algoritmo baseado em Transformada de *Hough* (círculos)

Como é observável nos resultados, a resolução utilizada foi superior aos 320x240 de captura usados em 4.3.1.2. Testes efetuados com resoluções inferiores aos 640x480 aqui utilizados revelam grande dificuldade mesmo na detecção de boias a curtas distâncias (ca. 2 m) sendo ainda frequente o surgimento de falsos positivos. Como adiante se concluirá, a montagem de hardware utilizada tem dificuldade trabalhar com esta resolução pelo que se decidiu enveredar por outro tipo de algoritmo de detecção.

4.3.2.2 Filtros de Variância Local

No sentido de experimentar a validade e possível aplicabilidade dos filtros de variância local, conforme sugerido em 2.2, adquiriu-se, na albufeira da barragem de Crestuma-Lever, uma sequência de imagens do ambiente de funcionamento previsto para o sistema.



Figura 4.28: T. *Hough*: Resultados de execução do algoritmo

O algoritmo de filtro de variância local, adaptado de [57], apresenta-se abaixo.

```

1 FILTRO DE VARIANCIA LOCAL
2
3 L=5
4 IMv=zeros(240,320);
5 for y=1:320
6     for x=1:240
7         sx=0;sxx=0;cnt=0;
8
9         for j=(y-L):(y+L)
10            for i=(x-L):(x+L)
11                if ((i>0) && (j>0) && (i<240) && (j<320))
12                    sx=sx+VARcres(i,j);
13                    sxx=sxx+VARcres(i,j).*VARcres(i,j);
14                    cnt=cnt+1;
15                end
16            end
17        end
18
19        IMv(x,y)=(sxx-(sx*sx)./cnt)./cnt;
20    end
21 end

```

	Medida	céu	Construções e vegetação	Água
Crestuma.jpg (RGB)	Brilho(med.)	167.4660	87.8266	111.3089
	Brilho(desv.pad.)	28.4583	62.4845	49.2978
	Saturação(med.)	0.0374	0.0333	0.0409
	Saturação(desv.pad.)	0.0887	0.0481	0.0862
Crestuma3.jpg (RGB)	Brilho(med.)	163.4250	107.0547	116.4293
	Brilho(desv.pad.)	34.4622	39.6725	27.8355
	Saturação(med.)	0.0478	0.0617	0.0248
	Saturação(desv.pad.)	0.1306	0.0852	0.0132
Crestuma2.jpg (RGB)	Brilho(med.)	160.8602	88.5505	99.0046
	Brilho(desv.pad.)	28.9258	54.8031	28.8158
	Saturação(med.)	0.0512	0.0338	0.0358
	Saturação(desv.pad.)	0.0896	0.0402	0.0843
Crestuma4.jpg (RGB)	Brilho(med.)	210.8565	106.0828	114.9280
	Brilho(desv.pad.)	30.4199	59.8777	31.6369
	Saturação(med.)	0.0209	0.0436	0.0200
	Saturação(desv.pad.)	0.0154	0.0340	0.0091



Tabela 4.2: Média e Desvio Padrão dos parâmetros Brilho e Saturação

Extraídas algumas imagens com elementos de três zonas distintas (Água - Céu - Vegetação/-Construções), observaram-se os resultados da tabela 4.2.

A tabela permite concluir que o brilho médio é sempre mais elevado no céu, sendo que o da água tem sempre o valor intermédio (de esperar pelas suas propriedades refletoras). Confirma-se também que o desvio padrão do brilho é mais elevado para a Vegetação/Construções.

A aplicação do algoritmo de filtro de variância local a imagens da cena resulta numa delimitação muito clara das três zonas. Na maioria dos casos, um filtro de $L=1$ (janela de 3×3) é suficiente. Veja-se a figura 4.29

Experimentou-se ainda perceber se pode retirar-se informação de distância tendo por base a relação $\frac{\text{saturacao}}{\text{brilho}}$. Nesta fase, percebeu-se que os sensores das câmaras utilizadas, na resolução com que foi adquirido o vídeo (320x240) degradam bastante o conteúdo da imagem. Mesmo assim, é perceptível que aquela razão, no caso do plano de água, decresce de valor com o aumento da distância. Note-se esse mesmo efeito numa imagem adquirida em Crestuma e uma outra, obtida online, com melhor qualidade (Figura 4.30 a) e b)).

A combinação da informação obtida por estes processos fornece pistas relevantes quer para a classificação de terreno, quer para a clarificação do tipo de objetos que a embarcação encontra

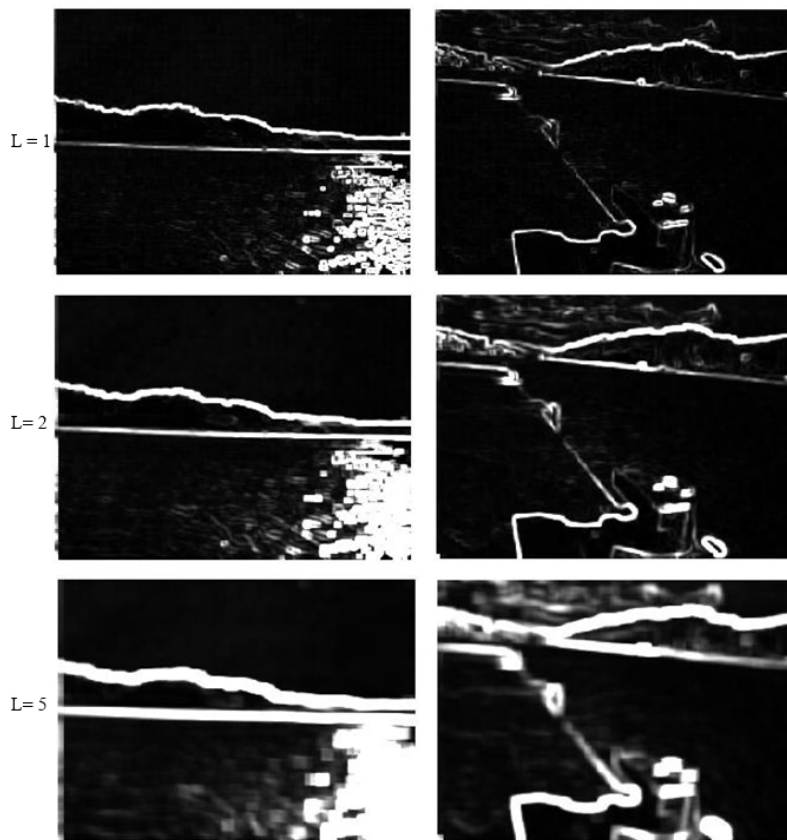


Figura 4.29: Aplicação do filtro de variância local (L=1,2,5)

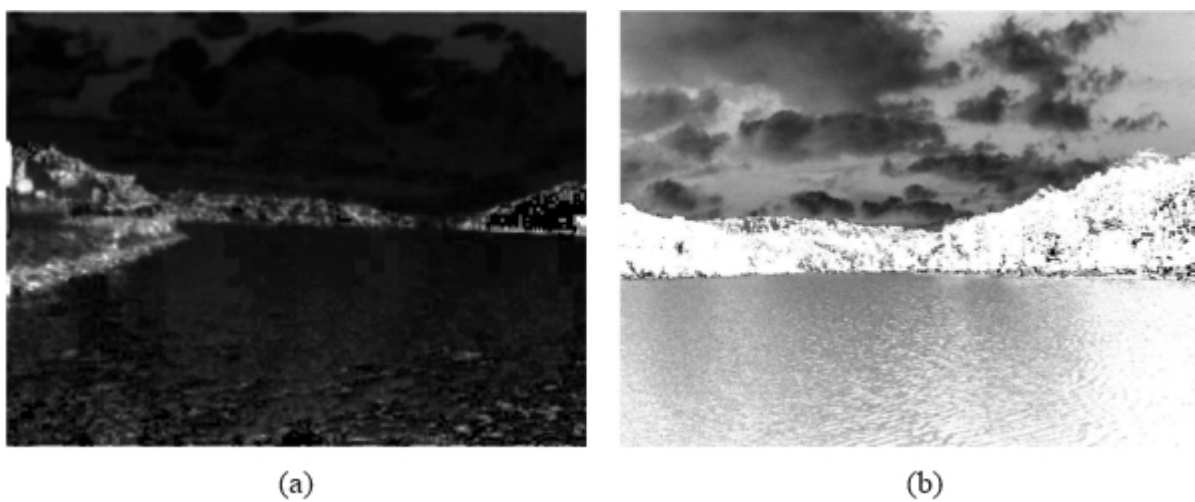


Figura 4.30: Superfície da Água - Efeito da profundidade na razão saturação/brilho

pela frente. No caso da navegação, a possibilidade de limitar a área de pesquisa apenas a zonas do *waterplane* pode permitir economia de recursos computacionais e limitar classificações erradas, especialmente se cruzada com a informação tridimensional da visão estereoscópica.

4.3.2.3 Matlab - Simulink Support Package for Raspberry Pi Hardware

A versão mais recente do software *Matlab*, da *Mathworks* - **2013a** - passou a incluir entre as suas *Toolbox* de *Simulink* uma que permite estabelecer ligação com o Raspberry Pi. À semelhança do que já acontecia com plataformas como *Arduino* ou *Beagleboard*, agora também é possível desenvolver de forma expedita aplicações para esta plataforma.

Para lhe aceder é necessária uma instalação, conseguida após uma sequência de passos simples. É imprescindível um cartao de memória de pelo menos 4GB visto que o processo instala nesse cartão uma imagem do Raspbian idêntica em tudo à distribuição mais utilizada de Linux para Raspberry.

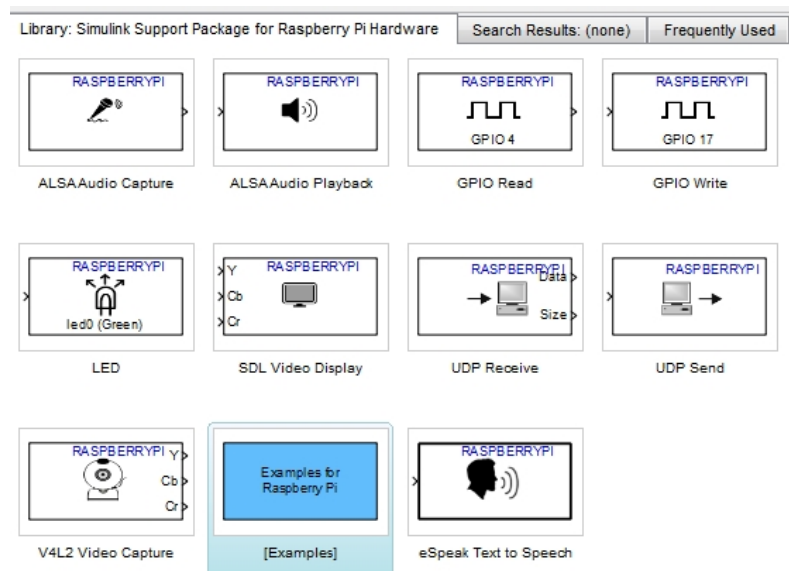


Figura 4.31: *Simulink Support Package for Raspberry Pi Hardware*

Criando um novo modelo de *Simulink*, basta aceder ao menu **Tools -> Run on Target Hardware -> Install/Update Support Package ...**, escolher a opção **Internet** e, avançando para o passo seguinte, escolher *Support package for Raspberry Pi* avançando novamente. Daqui em diante, a instalação dá instruções claras até à sua finalização.

Para testar esta *toolbox* foram experimentados algoritmos elementares de aquisição. A possibilidade de portar para C, em especial para Raspberry PI, quer *scripts*, quer modelos *Simulink* é uma valência que pode ter grande utilidade. As experiências feitas usando esta *toolbox* mostraram que podemos, por exemplo, utilizar blocos de "buffer" para controlar o *framerate* das câmaras, sendo possível definir qualquer valor (algo que, por exemplo, não possível foi com os controlos do bloco V4L2 da biblioteca Sdpo (5dpo) de *Lazarus* [52]); por outro lado, as funções de conversão de

sistemas de cor não compilam diretamente. Na tentativa de resolver esta questão, experimentou-se desenvolver um *script* que fizesse essa conversão (no caso, RGB->HSV). Na prática, abandonou-se a ideia de utilizar a *toolbox* para processamento estereoscópico porque o tratamento, em ramos paralelos, da imagem adquirida, gerava frequentemente diferenças temporais inaceitáveis entre ambos. Esta é, contudo, uma solução a ponderar para trabalhos com visão monocular, também pela possibilidade de utilização de ferramentas da *DSP System Toolbox*.

Capítulo 5

Testes Finais

Os testes ao sistema implementado foram feitos em laboratório. Por motivos logísticos, não houve, a partir do momento em que o sistema ficou terminado, possibilidade de levar a embarcação para ensaio na albufeira.

O *setup* final tem o aspeto da Figura 5.1.

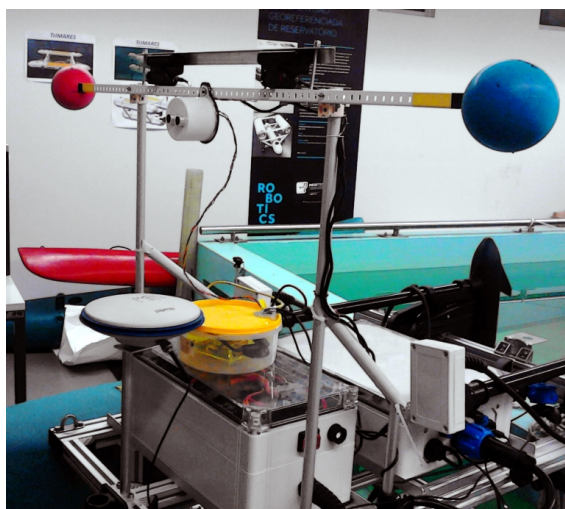


Figura 5.1: *Setup* Final

Utilizando um alvo bicolor, conforme apresentado na Figura 5.2, colocou-se a embarcação a uma distância de mais de 5m, o limite de funcionamento previsto para o sistema. O alvo tem, por força da baixa resolução utilizada, uma dimensão de cerca de 50x30 cm e, em cenário de utilização, deve acoplar-se à traseira do veículo seguido.

Com o algoritmo de deteção e seguimento em funcionamento, fez-se rodar a embarcação e obteve-se o resultado da Figura 5.5. São apresentadas as coordenadas (X,Y) corrigidas, na imagem esquerda e direita, a distância ao alvo, a orientação da embarcação relativamente a ele e o número de deteções de manchas de 1ª e 2ª cor em cada momento. Note-se que o sistema apresenta, para o alvo detetado, coordenadas Y_e e Y_d muito próximas, confirmando o cumprimento

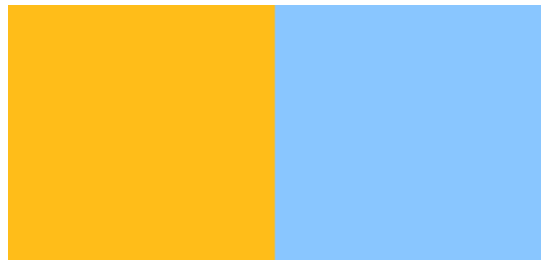


Figura 5.2: Alvo a seguir

da tolerância epipolar (10, no caso desta execução). As discrepâncias podem atribuir-se a diferenças na forma do objeto detetado por cada câmara após binarização, algo que desloca o centro de gravidade do objeto. Questões relacionadas com imprecisões na calibração também influenciam.

Numa outra execução do algoritmo, o alvo foi mantido fixo e fez-se variar a distância da embarcação ao alvo. O *printscreen* na figura 5.6 atesta o comportamento do algoritmo. O programa responde às alterações da posição do alvo face à embarcação, notando-se alguma oscilação de valores quando se encontra parada. Isto acontece porque em binarizações sucessivas de uma mesma cena, a imagem fornecida pela câmara não se mantém precisamente igual e isso tem efeito no resultado da aplicação das operações morfológicas sobre a imagem. São também visíveis períodos em que o alvo não é detetado, sendo por isso desejável que o tempo mínimo entre frames seja tão baixo quanto possível facilitando, nestas situações, uma reidentificação do alvo mais célere.

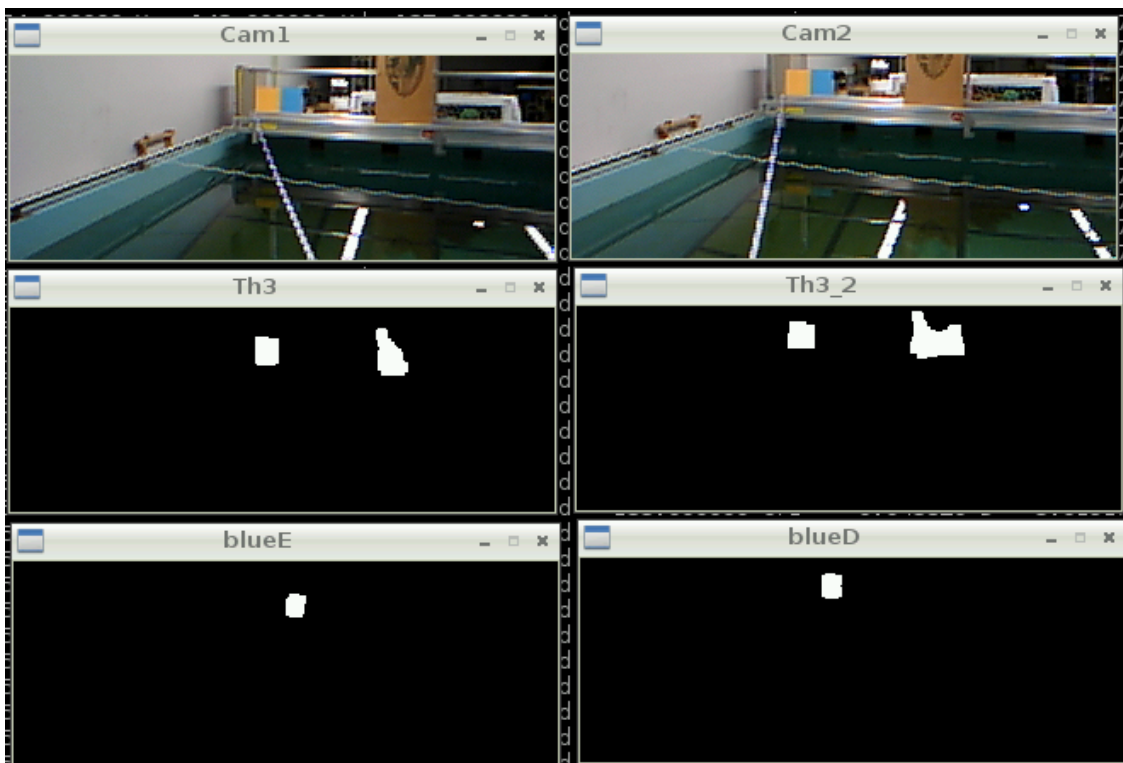
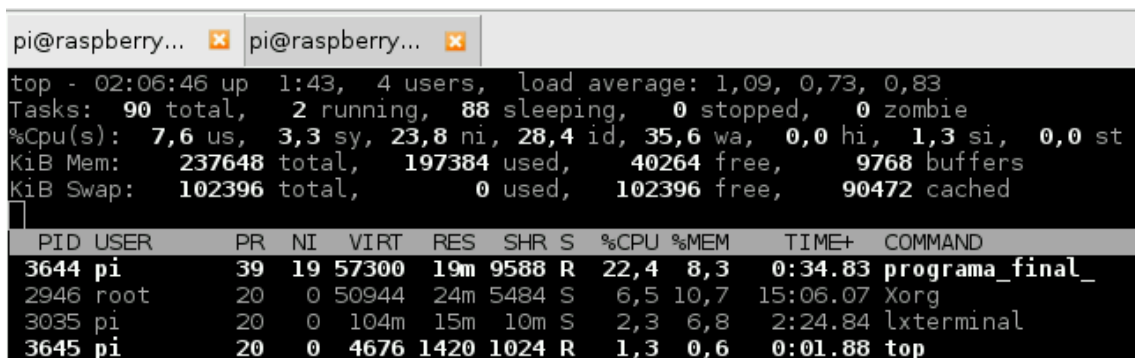


Figura 5.3: Cena de teste (orientação)

A Figura 5.3 mostra o sistema em funcionamento no laboratório. Colocou-se propositalmente um objeto parcialmente detetado como sendo de 1ª cor, para confirmar o bom funcionamento da detecção do alvo àquela distância. Veja-se que o sistema o reconhece mas não o combina com o objeto de 2ª cor. Note-se ainda que o caixote, que pretende simular parte de um falso alvo, é detetado de forma bastante diferente por ambas as câmaras. Isto acontece porque os limites de *threshold* não foram afinados para aquela cor.

Na mesma figura, as imagens Cam1 e Cam2 estão visivelmente desalinhadas na vertical. Isso acontece porque, conforme já explicado acima, não se aplica o algoritmo de remapeamento a toda a imagem, convertendo-se apenas os centros dos objetos de interesse.

O programa esteve aqui em execução com um tempo mínimo entre aquisições de pares de 150 ms. Isto traduz-se, na prática, por cerca de 3 fps. Funcionando desta forma, a Figura 5.4 atesta que absorve cerca de 30% do espaço de processamento do processador. Este valor sobe para uma média de mais de 40% se reduzirmos esse tempo mínimo para 80ms.



```

pi@raspberr... x pi@raspberr... x
top - 02:06:46 up 1:43, 4 users, load average: 1,09, 0,73, 0,83
Tasks: 90 total, 2 running, 88 sleeping, 0 stopped, 0 zombie
%Cpu(s): 7,6 us, 3,3 sy, 23,8 ni, 28,4 id, 35,6 wa, 0,0 hi, 1,3 si, 0,0 st
KiB Mem: 237648 total, 197384 used, 40264 free, 9768 buffers
KiB Swap: 102396 total, 0 used, 102396 free, 90472 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU  %MEM    TIME+  COMMAND
 3644 pi         39   19 57300  19m 9588 R   22,4   8,3    0:34.83 programa_final_
 2946 root        20    0 50944  24m 5484 S    6,5  10,7   15:06.07 Xorg
 3035 pi         20    0 104m  15m 10m S    2,3   6,8    2:24.84 lxterminal
 3645 pi         20    0  4676  1420 1024 R    1,3   0,6    0:01.88 top

```

Figura 5.4: Peso computacional

ALVO A SEGUIR	-> Xe = 136.000000	Ye = 171.000000	Xd = 106.000000	Yd = 167.000000	Ori = -0.116897	D = -3.228930M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 139.000000	Ye = 171.000000	Xd = 111.000000	Yd = 168.000000	Ori = -0.104783	D = -3.416402M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 141.000000	Ye = 171.000000	Xd = 112.000000	Yd = 167.000000	Ori = -0.096690	D = -3.320022M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 139.000000	Ye = 171.000000	Xd = 110.000000	Yd = 167.000000	Ori = -0.104783	D = -3.320022M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 143.000000	Ye = 178.000000	Xd = 112.000000	Yd = 171.000000	Ori = -0.088584	D = -3.142704M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 140.000000	Ye = 179.000000	Xd = 108.000000	Yd = 174.000000	Ori = -0.100738	D = -3.060963M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 143.000000	Ye = 181.000000	Xd = 110.000000	Yd = 178.000000	Ori = -0.088584	D = -2.983366M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 139.000000	Ye = 182.000000	Xd = 106.000000	Yd = 176.000000	Ori = -0.104783	D = -2.983366M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 139.000000	Ye = 182.000000	Xd = 104.000000	Yd = 179.000000	Ori = -0.104783	D = -2.839406M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 2)
ALVO A SEGUIR	-> Xe = 141.000000	Ye = 186.000000	Xd = 106.000000	Yd = 184.000000	Ori = -0.096690	D = -2.839406M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 0)
Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 0, 0)							
Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 0, 1)							
Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 0, 1)							
ALVO A SEGUIR	-> Xe = 141.000000	Ye = 148.000000	Xd = 89.000000	Yd = 145.000000	Ori = -0.096690	D = -2.013533M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 141.000000	Ye = 148.000000	Xd = 89.000000	Yd = 145.000000	Ori = -0.096690	D = -2.013533M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 2)
ALVO A SEGUIR	-> Xe = 140.000000	Ye = 148.000000	Xd = 89.000000	Yd = 145.000000	Ori = -0.100738	D = -2.048583M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 2, 2)
ALVO A SEGUIR	-> Xe = 154.000000	Ye = 136.000000	Xd = 118.000000	Yd = 133.000000	Ori = -0.043820	D = -2.772513M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 158.000000	Ye = 133.000000	Xd = 111.000000	Yd = 129.000000	Ori = -0.027489	D = -2.201899M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 158.000000	Ye = 136.000000	Xd = 124.000000	Yd = 132.000000	Ori = -0.027489	D = -2.909606M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 158.000000	Ye = 136.000000	Xd = 124.000000	Yd = 132.000000	Ori = -0.027489	D = -2.909606M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)							
Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)							
ALVO A SEGUIR	-> Xe = 144.000000	Ye = 137.000000	Xd = 101.000000	Yd = 136.000000	Ori = -0.084526	D = -2.380021M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 141.000000	Ye = 137.000000	Xd = 99.000000	Yd = 137.000000	Ori = -0.096690	D = -2.429147M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 146.000000	Ye = 142.000000	Xd = 99.000000	Yd = 139.000000	Ori = -0.076403	D = -2.201899M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 163.000000	Ye = 130.000000	Xd = 139.000000	Yd = 126.000000	Ori = -0.007055	D = -3.865233M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 163.000000	Ye = 129.000000	Xd = 139.000000	Yd = 124.000000	Ori = -0.007055	D = -3.865233M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 128.000000	Xd = 137.000000	Yd = 124.000000	Ori = -0.015230	D = -3.865233M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 162.000000	Ye = 127.000000	Xd = 139.000000	Yd = 124.000000	Ori = -0.011143	D = -3.996493M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 127.000000	Xd = 139.000000	Yd = 123.000000	Ori = -0.015230	D = -4.136982M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 143.000000	Yd = 122.000000	Ori = -0.015230	D = -4.813869M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 144.000000	Yd = 122.000000	Ori = -0.015230	D = -5.019176M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 143.000000	Yd = 122.000000	Ori = -0.015230	D = -4.813869M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 144.000000	Yd = 122.000000	Ori = -0.015230	D = -5.019176M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 144.000000	Yd = 122.000000	Ori = -0.015230	D = -5.019176M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)
ALVO A SEGUIR	-> Xe = 161.000000	Ye = 126.000000	Xd = 144.000000	Yd = 122.000000	Ori = -0.015230	D = -5.019176M	Objetos (1 cE, 1 cD, 2 cE, 2 cD) = (1, 1, 1, 1)

Figura 5.6: Testes de detecção e seguimento (distância) - Raspberry PI

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Problemas no desenvolvimento

O desenvolvimento deste trabalho encontrou diversos problemas. Em primeiro lugar, a necessidade de trabalhar com uma resolução relativamente baixa faz com que objetos de interesse que se encontrem a uma distância de poucos metros possam confundir-se com ruído da imagem, dificultando a classificação. A título de exemplo, a imagem abaixo demonstra o aspecto de uma bola de 20 cm de diâmetro a uma distância de 3 e 5m (Figura 6.1). A 5m, a bola têm uma área de apenas 11x12 pixels, já de si pequena para a detecção. Esta é mais dificultada ainda pelas deformações visíveis, mesmo num cenário de iluminação controlada.



Figura 6.1: Bola a cerca de 3m e 5m, respetivamente

Dada a possibilidade de definir a área de imagem a decodificar, a dimensão demasiado pequena de objetos a curtas distâncias levou a experiências com resoluções de captura maiores. Tentada a aquisição com 640x480 pixels, mesmo utilizando apenas cerca de 1/4 da altura da imagem, i.e., quarto inferior da imagem, com 640x120 pixels, problemas com sinais de sincronismo de cada câmara, mesmo sem que ambas estivessem ligadas entre si, impediram a utilização desta resolução 6.2. A inconstância dos "saltos" da imagem, tanto segundo o eixo horizontal como vertical, não revelou nenhum padrão que pudéssemos identificar, pelo que a hipótese de aumentar a resolução foi descartada.



Figura 6.2: Aquisição defeituosa a 640x120 pixels

A troca de informação entre o Raspberry Pi e a stack PC-104 da embarcação também não foi direta. Os valores recebidos do lado da embarcação não tinham sentido, algo que aconteceu por estarem a interagir arquiteturas computacionais distintas. O problema resolveu-se modificando o *byte packing* do compilador, utilizando a diretiva `#pragma pack` e impondo um alinhamento de dados de 4 bytes.

Um outro problema, na fase de testes, foi a integração do Raspberry Pi juntamente com o restante hardware do ASV. O facto de utilizarmos componentes *off-the-shelf*, sem nenhum tipo de isolamento eletromagnético levou a que as interferências dos outros dispositivos do barco contaminassem os sinais de sincronismo das câmaras. Por causa disto, a imagem da câmara direita aparecia completamente destruída, levando a que o algoritmo de deteção de alvos/obstáculos detetasse objetos sem sentido e gerando ordens erradas de movimento para o ASV.

Um problema detetado tardiamente, que teve influência na forma como estava a ser feita a deteção de cores, tem que ver com as diferenças entre as imagens convertidas pela função de OpenCV `cvCvtColor()` e a função `rgb2hsv()` de *matlab*. A partir da identificação deste problema, as conversões de imagens para efeito de determinação de threshold passaram a ser feitas usando a função de OpenCV. Esta diferença é também reforçada pelo facto de o aspeto de uma imagem adquirida com V4L2 e em ambiente Windows não ser exatamente igual (Figura 6.3). Um dos motivos pelos quais isto poderá ocorrer é a diferença de tratamento que os drivers dão à informação vinda das câmaras em cada um dos sistemas operativos.

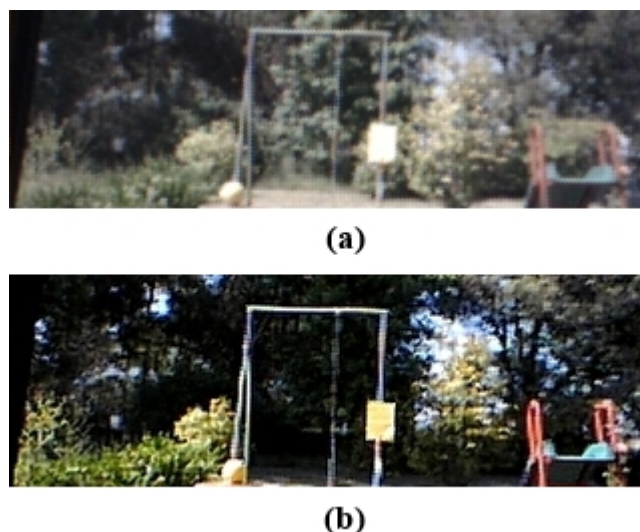


Figura 6.3: Imagem adquirida em (a) Windows (b) V4L2

Outro problema ainda, surgido por altura dos testes ao algoritmo final, foi a avaria de uma das câmaras. Foi necessário encomendar uma nova câmara e repetir todo o processo de adaptação de hardware e calibração, individual e do par. Uma inspeção com um osciloscópio permitiu detetar uma possível fonte para o problema: um pico de tensão de muito alta frequência (Figura 6.4),

com cerca de 2 Vpp, estava a ocorrer no barramento de 5V da embarcação; acabou por se perceber que o problema ocorria em ambos os ASV's (Zarco e Gama) e se devia a um sensor instalado em ambos.

O facto de, embora tendo uma GPU potente, não a ter até ao momento acessível, torna o Raspberry PI uma base bastante limitada para o processamento de imagem. Até ao momento, a comunidade continua a desenvolver pequenas aplicações de visão passando à margem da GPU VideoCore IV e deixando todo o trabalho ao processador ARM.

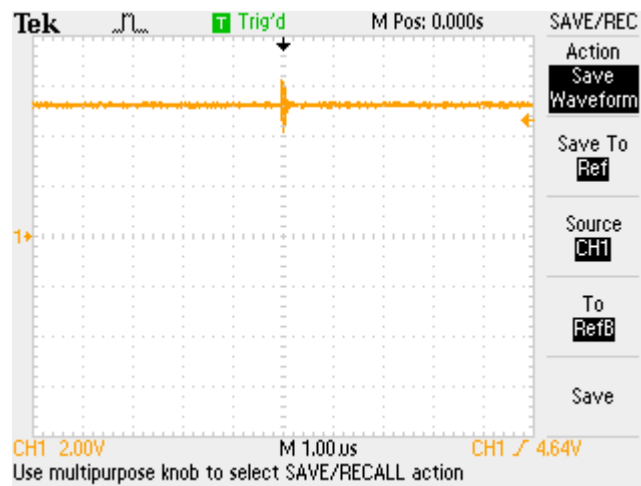


Figura 6.4: Pico de Tensão no barramento de 5V do ASV

6.2 Satisfação dos Objectivos

Os objetivos iniciais foram globalmente atingidos. Foi seleccionado hardware de baixo custo para um sistema de condução visual e fez-se um estudo abrangente da área do processamento de imagem, tendo sido exploradas as possibilidades de implementação de diversos métodos de detecção de objetos. Foi desenvolvida uma estrutura apta a fornecer pistas e informação estereoscópica do ambiente em que a embarcação opera, tendo sido testado o seu funcionamento para distâncias de até 5m em laboratório. O sistema, por ser composto por equipamentos de uso genérico, obrigou a uma adaptação do hardware adquirido, para permitir a sua sincronização. Do ponto de vista do software, métodos de aquisição genericamente válidos para aplicações de visão monocular revelaram-se ineficazes e não suficientemente rápidos para cumprir as restrições temporais a que este tipo de aplicação obriga; os algoritmos finais utilizaram bibliotecas de OpenCV mas, para a aquisição foi estudada uma solução baseada na API *Video4Linux2*. Se originalmente a pesquisa criou grande expectativa quanto à multiplicidade de métodos a utilizar no sistema de deteção e seguimento, a prática mostrou que a solução teria de ser diferente e passar por métodos computacionalmente menos dispendiosos. Uma parte significativa do tempo foi dispendido a experimentar implementações de *Block Matching*, *Graph Cut* ou SIFT mas, quer pela prestação temporal (no caso de métodos de Graph Cut, uma imagem de 320x240 chega a ultrapassar os 20 segundos na

geração de um mapa de disparidades), quer pela necessidade de variado processamento acessório para lidar com problemas de falsos positivos ou mesmo classificação dos objetos detetados, a opção teve de recair sobre métodos de informação menos densa. Até que a *Broadcom* liberte informação de acesso à GPU do Raspberry PI, algoritmos de processamento mais exigentes, especialmente com estereoscopia, estão vedados para aplicações que funcionem online. Foi explorada a possibilidade de aplicação de filtros de variância local para delimitação de contornos de diferentes tipos de terrenos avistados pelo sistema e testada a viabilidade de obter mais informação sobre o tipo de terreno baseada nas relações entre as componentes de cor. Uma nova Toolbox Matlab/Simulink desenvolvida para interação com Raspberry PI foi experimentada. Nesta fase, a toolbox pode revelar-se útil para visão monocular mas, uma vez mais, é ineficiente para lidar com visão estereoscópica, não garantindo o sincronismo da imagem recebida pelas duas câmaras.

O sistema desenvolvido interage diretamente com a *framework* em utilização nos veículos do OceanSys, tendo isso mesmo sido testado em laboratório embora, por limitações da dimensão da piscina, apenas testes de orientação tenham sido efetuados. Com efeito, fez-se o barco seguir um alvo tendo-se ele orientado no sentido do alvo. Embora lamentando não ter sido possível o teste no exterior, assim que haja disponibilidade, testar-se-á o sistema em funcionamento no exterior.

Parte do trabalho desenvolvido serviu de base para a submissão de um *extended abstract* para a conferência Oceans2013; além de ter sido aceite no programa regular da conferência, a submissão ficou entre as cerca de 20 selecionadas para o *Student Program* da conferência.

6.3 Trabalho Futuro

O trabalho até aqui desenvolvido pode evoluir por várias vias. Para começar, o desenvolvimento centrou-se principalmente nos algoritmos de visão pelo que, embora implementados do lado da visão, i.e., o algoritmo corrido em Raspberry PI envia para o Target Track informação de boias e alvo mas, nesta fase, ele cumpre apenas o seguimento do alvo. A incorporação na embarcação de uma lógica de decisão simples de definição de *waypoints* baseada na largura dos caminhos definidos por boias, quer seguisse ou não um alvo. Também a incorporação de vários sensores de ultrassons na embarcação é também uma forma económica e computacionalmente pouco exigente de a proteger face a perigos repentinos. A informação obtida a partir deles pode ser utilizada para implementar uma metodologia simplificada de COLREGS. Mantendo a configuração de hardware até aqui utilizada - Câmaras e Raspberry PI - poderia testar-se a utilização de outras bibliotecas de funções, por exemplo Point Cloud Library [53], ou explorar funcionalidades oferecidas por *packages* de software para criação de aplicações robóticas, por exemplo ROS - *Robot Operating System* (existe, inclusivamente, uma versão do sistema operativo Raspbian que incorpora ROS, o Raspberry Pi) [54] ou MRPT - *Mobile Robot Programming Toolkit* [55]. No caso destas *packages*, pode haver vantagem em recorrer a unidades computacionais com maior capacidade, por forma a se conseguir a execução de um maior número de tarefas com uma só máquina. Uma máquina quad-core como o *Hardkernel Odroid - U2*, também económica, permitirá certamente melhorar a

caraterização obtida da envolvência da embarcação e, conseqüentemente, a capacidade de detecção de obstáculos do sistema.

Nos últimos meses, surgiu uma câmara feita propositadamente para o Raspberry PI (Figura 6.5). Trata-se de uma câmara com um sensor de 5MP capaz de adquirir 30 *fps* com resolução de 1080p ou 60 *fps* com resolução de 720p. Uma estrutura utilizando 2 Raspberry Pi com duas câmaras deste tipo ligados via Ethernet, utilizando MPI (*Message Passing Interface*) para paralelizar o processamento [56] poderia ser também uma maneira de expandir as valências do sistema.

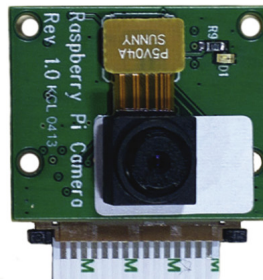


Figura 6.5: Raspberry Pi Cam

Uma possibilidade a explorar pode ser a utilização da visão estereoscópica para docagem dos veículos após cumprimento de missões de varrimento de áreas, também elas baseadas em referências visuais.

Certeza, apenas de que, nas circunstâncias atuais, o Raspberry PI funcionando ao mesmo tempo como gerador de referências para seguimento de outra embarcação e detetando boias direitas e esquerdas se encontra próximo do limite da sua capacidade de processamento. Qualquer função de alguma complexidade que se queira ver executada implicará a retirada de alguma das existentes.

Referências

- [1] Nuno Cruz, Aníbal Matos, Sérgio Cunha, e Sérgio Silva. *Zarco - An Autonomous Craft For Underwater Surveys*. 2007.
- [2] N. Tesla. Method of and apparatus for controlling mechanism of moving vessels or vehicles. 1898.
- [3] J. Boileau. *Fastest In The World: The Saga Of Canada's Revolutionary Hydrofoils*. Formac Publications, 2004.
- [4] G. Hitz, F. Pomerleau, M.-E. Garneau, C. Pradalier, T. Posch, J. Pernthaler, e R.Y. Siegwart. Autonomous inland water monitoring: Design and application of a surface vessel. *Robotics Automation Magazine, IEEE*, 19(1):62–72, march 2012. doi:10.1109/MRA.2011.2181771.
- [5] J. Wilde, D. DiBiaso, e M. Nervegna. Team planning for unmanned vehicles in the risk-aware mixed-initiative dynamic replanning system. Em *OCEANS 2007*, 2007.
- [6] Mini asv- swift application and less manpower, Outubro 2012. <http://www.usm.my/index.php/en/news-archive/6023-MINI-ASV--SWIFT-APPLICATION-AND-LESS-MANPOWER.html>.
- [7] Jie Zhao, Xinbin Zhang, Ning Chen, e Qinmin Pan. Why superhydrophobicity is crucial for a water-jumping microrobot? experimental and theoretical investigations. *ACS Applied Materials and Interfaces*, 4(7):3706–3711, 2012.
- [8] Carol Martínez, Thomas Richardson, Peter Thomas, Jonathan Luke du Bois, e Pascual Campoy. A vision-based strategy for autonomous aerial refueling tasks. *Robotics and Autonomous Systems*, 61(8):876 – 895, 2013. URL: <http://www.sciencedirect.com/science/article/pii/S0921889013000420>, doi:<http://dx.doi.org/10.1016/j.robot.2013.02.006>.
- [9] F. Fraundorfer, L. Heng, D. Honegger, G.H. Lee, L. Meier, P. Tanskanen, e M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. Em *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, páginas 4557–4564, 2012. doi:10.1109/IROS.2012.6385934.
- [10] Tobias Neumann e Alexander Schlaefer. Feasibility of basic visual navigation for small robotic sailboats. Em Colin Sauzé e James Finnis, editores, *Robotic Sailing 2012*, páginas 13–22. Springer Berlin Heidelberg, 2013.
- [11] Larry Matthies, Paolo Bellutta, e Mike Mchenry. Detecting water hazards for autonomous off-road navigation. Em *Proceedings of SPIE Conference 5083: Unmanned Ground Vehicle Technology V*, páginas 263–352, 2003.

- [12] Arturo L. Rankin e Larry Matthies. Daytime water detection based on color variation. Em *IROS*, páginas 215–221. IEEE, 2010. URL: <http://dblp.uni-trier.de/db/conf/iros/iros2010.html#RankinM10>.
- [13] Lawrence B. Wolff. Polarization vision: a new sensory approach to image understanding. *Image and Vision Computing*, 15(2):81 – 93, 1997. URL: <http://www.sciencedirect.com/science/article/pii/S0262885696011237>, doi:10.1016/S0262-8856(96)01123-7.
- [14] M. Svedman, L. Goncalves, N. Karlsson, M. Munich, e P. Pirjanian. Structure from stereo vision using unsynchronized cameras for simultaneous localization and mapping. Em *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, páginas 3069 – 3074, aug. 2005. doi:10.1109/IROS.2005.1545431.
- [15] U. S. o. A. Department of Defense. Unmanned systems integrated roadmap fy2011- 2036, 2011.
- [16] B. Haughton. *Hidden History: Lost Civilizations, Secret Knowledge, and Ancient Mysteries*. Career Press Incorporated, 2007.
- [17] V. Bertram. Unmanned surface vehicles—a survey. 2008.
- [18] P. Bhattacharjee, P. Rakshit, I. Goswami, A. Konar, e A.K. Nagar. Multi-robot path-planning using artificial bee colony optimization algorithm. Em *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, páginas 219 –224, oct. 2011. doi:10.1109/NaBIC.2011.6089601.
- [19] Anan Banharnsakun, Tiranee Achalakul, e Romesh C. Batra. Target finding and obstacle avoidance algorithm for microrobot swarms. Em *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, páginas 1610 –1615, oct. 2012. doi:10.1109/ICSMC.2012.6377967.
- [20] R. Falcon, Xu Li, A. Nayak, e I. Stojmenovic. A harmony-seeking firefly swarm to the periodic replacement of damaged sensors by a team of mobile robots. Em *Communications (ICC), 2012 IEEE International Conference on*, páginas 4914 –4918, june 2012. doi:10.1109/ICC.2012.6363859.
- [21] V. Ganapathy, T.T.J. Jie, e S. Parasuraman. Improved ant colony optimization for robot navigation. Em *Mechatronics and its Applications (ISMA), 2010 7th International Symposium on*, páginas 1 –6, april 2010.
- [22] S. Sanchez, A.A. Solovev, S.M. Harazim, e O.G. Schmidt. Microbots swimming in the flowing streams of microfluidic channels. *Journal of the American Chemical Society*, 133(4):701–703, 2011.
- [23] African Robotics Network. "10 dollar robot" design challenge, 2012. Disponível em http://robotics-africa.org/design_challenge.html, acessado a última vez em 3 de Janeiro de 2012.
- [24] Herbert Bay, Andreas Ess, Tinne Tuytelaars, e Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, Junho 2008. URL: <http://dx.doi.org/10.1016/j.cviu.2007.09.014>, doi:10.1016/j.cviu.2007.09.014.

- [25] Martin A. Fischler e Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, Junho 1981. URL: <http://doi.acm.org/10.1145/358669.358692>, doi:10.1145/358669.358692.
- [26] A. Finn e S. Scheduling. *Developments and Challenges for Autonomous Unmanned Vehicles: A Compendium*. Intelligent systems reference library. Springer London, Limited, 2010.
- [27] J. Borenstein e Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, jun 1991. doi:10.1109/70.88137.
- [28] I. Ulrich e J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. Em *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, páginas 1572–1577 vol.2, may 1998. doi:10.1109/ROBOT.1998.677362.
- [29] I. Ulrich e J. Borenstein. Vfh*: local obstacle avoidance with look-ahead verification. Em *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, páginas 2505–2511 vol.3, 2000. doi:10.1109/ROBOT.2000.846405.
- [30] M.R. Benjamin e J.A. Curcio. Colregs-based navigation of autonomous marine vehicles. Em *Autonomous Underwater Vehicles, 2004 IEEE/OES*, páginas 32–39, june 2004. doi:10.1109/AUV.2004.1431190.
- [31] Yoshiaki Kuwata, Michael T. Wolf, Dimitri Zarzhitsky, e Terrance L. Huntsberger. Safe maritime navigation with colregs using velocity obstacles. Em *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, páginas 4728–4734, sept. 2011. doi:10.1109/IROS.2011.6094677.
- [32] M. Sarkar, D. San Segundo Bello, C. Van Hoof, e A. Theuwissen. Integrated polarization analyzing cmos image sensor for material classification. *Sensors Journal, IEEE*, 11(8):1692–1703, aug. 2011. doi:10.1109/JSEN.2010.2095003.
- [33] J.E. Ahmad e Y. Takakura. Improving segmentation maps using polarization imaging. Em *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 1, páginas I–281–I–284, 16 2007-oct. 19 2007. doi:10.1109/ICIP.2007.4378946.
- [34] M. Ferraton, C. Stolz, e F. Meriaudeau. Surface reconstruction of transparent objects by polarization imaging. Em *Signal Image Technology and Internet Based Systems, 2008. SITIS '08. IEEE International Conference on*, páginas 474–479, 30 2008-dec. 3 2008. doi:10.1109/SITIS.2008.18.
- [35] M. Iqbal, O. Morel, e F. Meriaudeau. Choosing local matching score method for stereo matching based-on polarization imaging. Em *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 2, páginas 334–338, feb. 2010. doi:10.1109/ICCAE.2010.5451536.
- [36] M.Z. Brown, D. Burschka, e G.D. Hager. Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):993–1008, aug. 2003. doi:10.1109/TPAMI.2003.1217603.
- [37] Transformação sift (scale invariant feature transform). URL: http://www.maxwell.lambda.ele.puc-rio.br/9142/9142_4.PDF.

- [38] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, e Richard Szeliski. A database and evaluation methodology for optical flow. *Int. J. Comput. Vision*, 92(1):1–31, Março 2011. URL: <http://dx.doi.org/10.1007/s11263-010-0390-2>, doi:10.1007/s11263-010-0390-2.
- [39] J.L. Barron, D.J. Fleet, S.S. Beauchemin, e T. A. Burkitt. Performance of optical flow techniques. Em *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, páginas 236–242, 1992. doi:10.1109/CVPR.1992.223269.
- [40] M. Durkovic, M. Zwick, F. Obermeier, e K. Diepold. Performance of optical flow techniques on graphics hardware. Em *Multimedia and Expo, 2006 IEEE International Conference on*, páginas 241–244, 2006. doi:10.1109/ICME.2006.262427.
- [41] João Miguel Queirós Magno Leitão. Síntese por Computador de Imagens Estereoscópicas com Elevado Realismo. Tese de mestrado, 1994.
- [42] H. D. Cheng, X. H. Jiang, Y. Sun, e Jing Li Wang. Color image segmentation: Advances and prospects. *Pattern Recognition*, 34:2259–2281, 2001.
- [43] Teledynedalsa. Ccd vs cmos. Disponível em http://www.teledynedalsa.com/corp/markets/CCD_vs_CMOS.aspx, acessado a última vez em 20 de Fevereiro de 2013.
- [44] Stuart A. Taylor. Ccd and cmos imaging array technologies: Technology review. XEROX, 1998.
- [45] G. Bradski e A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. Software that sees. O’Reilly Media, Incorporated, 2008. URL: <http://books.google.pt/books?id=seAgiOfu2EIC>.
- [46] Raspberry pi super computer - university of southampton. Acessado em 13 de Julho de 2013. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/#start.
- [47] J. M. Coelho, M. e Tavares. Toolbox de calibração de câmaras para matlab - relatório interno. Acessado em 13 de Julho de 2013.
- [48] N. Hasler, B. Rosenhahn, T. Thormahlen, M. Wand, J. Gall, e H. P Seidel. Markerless motion capture with unsynchronized moving cameras. Em *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, páginas 224–231, 2009. doi:10.1109/CVPR.2009.5206859.
- [49] Video 4 linux 2 api infrastructure. Acessado em 15 de Junho de 2013. URL: <http://linuxtv.org/downloads/v4l-dvb-apis/>.
- [50] Video 4 linux 2 api infrastructure. Acessado em 15 de Junho de 2013. URL: <http://linuxtv.org/downloads/v4l-dvb-apis/control.html>.
- [51] Video 4 linux 2 api infrastructure. Acessado em 15 de Junho de 2013. URL: <http://linuxtv.org/downloads/v4l-dvb-apis/extended-controls.html>.
- [52] Toolbox de calibração de câmaras para matlab - relatório interno. URL: <http://wiki.freepascal.org/5dpo>.

- [53] Point cloud library. Acedido em 18 de Junho de 2013. URL: <http://pointclouds.org/>.
- [54] Robot operating system. Acedido em 18 de Junho de 2013. URL: <http://ros.org/>.
- [55] Mobile robot programming toolkit. Acedido em 18 de Junho de 2013. URL: <http://www.mrpt.org/>.
- [56] Raspberry pi super computer - university of southampton. Acedido em 27 de Junho de 2013. URL: <http://www.southampton.ac.uk/~sjc/raspberrypi/>.
- [57] M. Haidekker. *Advanced Biomedical Image Analysis*. Wiley, 2011.

