INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Tratamento de Imagem e detecção de objectos

JOÃO DAVID GUIMARÃES FIGUEIREDO DIAS
novembro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Pré-processamento de imagem e detecção de objectos em imagens captadas por uma câmara de plâncton

João David Dias

Mestrado em Engenharia Electrotécnica

Orientador: Eduardo Silva

November 20, 2019

À minha tia Nazaré, que não tinha o mínimo interesse em ler isto, mas tinha todo o interesse em que eu o escrevesse.

Esta Página
Foi Intencionalmente
Deixada Em Branco

Resumo

Em missões com o objectivo de captar imagens subaquáticas, a necessidade de uma maior eficiência na obtenção de resultados é um problema recorrente. Este projecto tem como objectivo contribuir para a o robustecimento da câmara de plâncton do projecto *MarinEye* e solucionar os problemas de armazenamento causados pela necessidade de uma elevada resolução nas imagens captadas. Tendo isso em conta, foi desenvolvido um sistema de rápido processamento de imagens responsável pela selecção de informação relevante para recolha. Foram estudados métodos de melhoramento de imagem, extracção de características, *edge detection* e detecção de objectos. Foram também abordados métodos de detecção de objectos e melhoramento de imagem com redes neuronais. Nesta tese foi ainda estudado o estado da arte do campo da visão subaquática no que diz respeito a detecção de objectos e melhoramento de imagem. Esta tese implementa normalização para o melhoramento de imagens e uma cadeia de processos para detecção de objectos que contém *canny edge detection*, transformações morfológicas, detecção de contornos e detecção de áreas de modo a avaliar conteúdo de potencial interesse para pós processamento. A solução foi implementada em tempo real recorrendo a ROS e atinge 12 *frames* por segundo no seu estado actual de implementação. Foram realizados testes em ambiente semi-controlado e no rio Douro que validaram a solução desenvolvida. A solução apresentada melhora o contraste de imagens subaquáticas, detecta potenciais objectos de interesse, processa a informação em tempo real e é capaz de comunicar com a câmara de plânction do *MarinEye*, contribuindo para o *Technology Readiness Level* deste sistema.

Esta Página
Foi Intencionalmente
Deixada Em Branco

Abstract

In missions with the objective of gathering underwater images the need for better efficiency when it comes to results is a recurrent problem. This project's main objective is to contribute to MarinEye's plankton camera improvement and solve the storage problem caused by the high quality of image needed for this project. Having that in mind it was developed a fast image processing system responsible for selecting relevant information to gather. There was a research regarding methods of image enhancement, feature extraction, *edge detection* and object detection. There were also studied methods of object detection and image enhancement with neural networks. This thesis explores the state of the art of the field of underwater vision regarding image enhancement and object detection. This thesis implements normalisation for image enhancement and a series of processes for object detection like *canny edge detection*, morphological transformations, contour detection and blob detection in order to evaluate potential areas of interest for post-processing. This solution was implemented in real time with ROS and goes up to 12 frames per second in the current state of implementation. There were conducted tests in a semi-controlled environment and in Douro river which validated the solution presented in this thesis. The solution improves the contrast of underwater images, detects potential areas or objects of interest, processes that information in real time and it is capable of communicating with MarinEye's plankton camera contributing for the *Technology Readiness Level* of the system.

Esta Página
Foi Intencionalmente
Deixada Em Branco

Agradecimentos

Agradeço ao meu orientador Eduardo Silva pela forma como me ajudou a concluir este projecto, pelos conselhos que me deu e pela forma como trata todos os alunos, algo que torna o LSA num lugar diferente de qualquer sítio onde tinha estado até ao momento a estudar. Ao José Eduardo Silva, por toda a ajuda que me deu ao longo de toda a minha estadia pelo ISEP e na vida em geral, desde que o conheci, ao Pedro Guedes por me obrigar a trabalhar e ser crucial para acabar muitas cadeiras e ao Daniel Freire pela paciência de me explicar conteúdos de cadeiras que foram muito importantes para realizar esta tese.

À Bertil Marques por me receber no auditório, a melhor coisa que me aconteceu no ISEP. À minha família e à minha namorada, Mariana.

João Dias

Esta Página
Foi Intencionalmente
Deixada Em Branco

Conteúdos

Lista de Figuras

Lista de Abreviaturas

CIIMAR	**C**entro **I**nterdisciplinar de **I**nvestigação **M**arinha e **A**mbiental
FPGA	***F**ield **P**rogrammable **G**ate **A**rray*
FCUP	**F**aculdade de **C**iências da **U**niversidade do **P**orto
FPS	***F**rames **P**er **S**econd*
GPU	***G**raphics **P**rocessing **U**nit*
IDS	**I**maging **D**evelopment **S**ystems
INESC TEC	**I**nstituto de **E**ngenharia de **S**istemas e **C**omputadores, **Te**cnologia e **C**iência
IPMA	**I**nstituto **P**ortuguês do **M**ar e da **A**tmosfera
ISEP	**I**nstituto **S**uperior de **E**ngenharia do **P**orto
LoG	***L**aplacian **o**f **G**aussian*
MARE	**C**entro de **C**iências do **M**ar e do **A**mbiente
Mb	**M**ega**b**yte
ROS	***R**obot **O**perating **S**ystem*
SIFT	***S**cale-**I**nvariant **F**eature **T**ransform*
SURF	***S**peeded **U**p **R**obust **F**eatures*
TPU	***T**ensor **P**rocessing **U**nit*
TRL	***T**echnology **R**eadiness **L**evel*
USB	**U**niversal **S**erial **B**us
YOLO	***Y**ou **O**nly **L**ook **O**nce*

Esta Página
Foi Intencionalmente
Deixada Em Branco

Capítulo 1

Introdução

1.1 Contextualização e Motivação

O intuito deste trabalho é, contribuir para o crescimento do TRL(*Technology Readiness Level*)[1] do projecto *MarinEye*[2], um projecto que visa o desenvolvimento de um sistema autónomo para monitorização química, física e biológica de ambientes oceânicos. O sistema *MarinEye* é composto por quatros subsistemas, nomeadamente captação de imagens de alta resolução para a avaliação de plâncton, sistemas de sonar (passivos e activos), filtragem autónoma e um sistema de *samples* de DNA e RNA. O sistema é compacto e preparado para ser usado em plataformas fixas ou móveis[2]. A figura 1.2 mostra algumas das detecções feitas pelo sistema *MarinEye*, classificações efectuadas por biólogos do CIIMAR e o resultado obtido pelo sistema de melhoramento de imagem desenvolvido nesta tese.

Mais especificamente, o trabalho desta dissertação contribui para o melhoramento do sistema de aquisição de imagens de plâncton onde foram identificadas algumas limitações relativamente ao funcionamento autónomo. Essa limitação ocorre fundamentalmente pela necessidade de se adquirir imagens com elevada resolução, que resulta num problema de dimensão de armazenamento. Consequentemente, no caso da aplicação remota deste sistema resulta um problema de armazenamento, um problema de comunicação de informação e um problema de tempo de processamento. Esta tese pretende contribuir para a resolução desses problemas. O primeiro ponto deve-se ao facto da necessidade das imagens terem de ser capturadas com alta qualidade com dimensões de 4912 x 3684 pixeis que, a uma taxa de aquisição de cinco *frames* por segundo, resulta na necessidade de 172 *Mbytes* de armazenamento a cada segundo, que rapidamente encheria o dispositivo de armazenamento. A solução para este problema é não guardar todas as imagens captadas, mas só as que contêm informação potencialmente relevante. Com esta solução prolongar-se-ia o tempo possível de missão, já que seria necessário mais tempo para que o dispositivo de

armazenamento chegasse ao seu limite. O segundo ponto refere-se à deficiência no acesso ao sistema de captação, que devido à forma como pode ser controlado (via *ethernet*) tem algumas restrições de velocidade e a acessibilidade ao utilizador não é total, já que para aceder ao *hardware* é necessário desmontar o sistema. A única forma "não intrusiva" de verificar a qualidade e o conteúdo das imagens que estão a ser captadas é via *ethernet*. Para este problema ser ultrapassado é necessário que exista um sistema que faça uma selecção automática das imagens que são guardadas de forma a não ser necessário interagir com o sistema autónomo que as está a capturar e que exista o mesmo grau de confiança nessa selecção que se teria ao verificar manualmente os resultados. O último ponto incide sobre o facto de imagens captadas debaixo de água sofrerem algumas perturbações que resultam de problemas tais como: *blur*, turbidez e falta de luz, o que implica tempo de processamento mais elevado. O *blur* caracteriza-se pelo aspecto de imagem desfocada causado pelo movimento da água, a turbidez caracteriza-se pela obstrução óptica que existe em ambiente não controlado devido a detritos que, no caso concreto de detecções, podem causar falsos positivos. Os objectos são, ainda, menos visíveis devido à fraca iluminação causada pela impossibilidade de penetração da luz na água. Para este problema é necessário desenvolver um método de melhoramento de imagens de forma a melhorar o contraste da imagens para pós processamento.

Esta dissertação tem como objectivo contribuir para a solução dos problemas identificados.

1.2 Objectivos

Como referido anteriormente, o objectivo principal deste projecto é desenvolver um sistema de detecção de indivíduos (plâncton) em imagens recolhidas no âmbito do projecto *Marineye* para posterior categorização por uma rede neuronal. O sistema desenvolvido tem como propósito ser um filtro entre a fonte (imagens adquiridas no âmbito do projecto *Marineye*) e o destino (rede neuronal) visto que, até ao momento, as imagens recolhidas eram gravadas sem qualquer filtragem e escolhidas após análise, uma a uma. Este projecto pretende reduzir o tempo de análise descartando imagens que não contenham os objectos de interesse e prevenir que o dispositivo de armazenamento ocupe todo o seu espaço com informação desnecessária aumentando, assim, o tempo possível de uma missão, já que o esgotamento do dispositivo de armazenamento ocorreria mais tarde, e a qualidade de informação trazida de volta para o laboratório. Os tópicos seguintes devem ser atingidos para esse efeito:

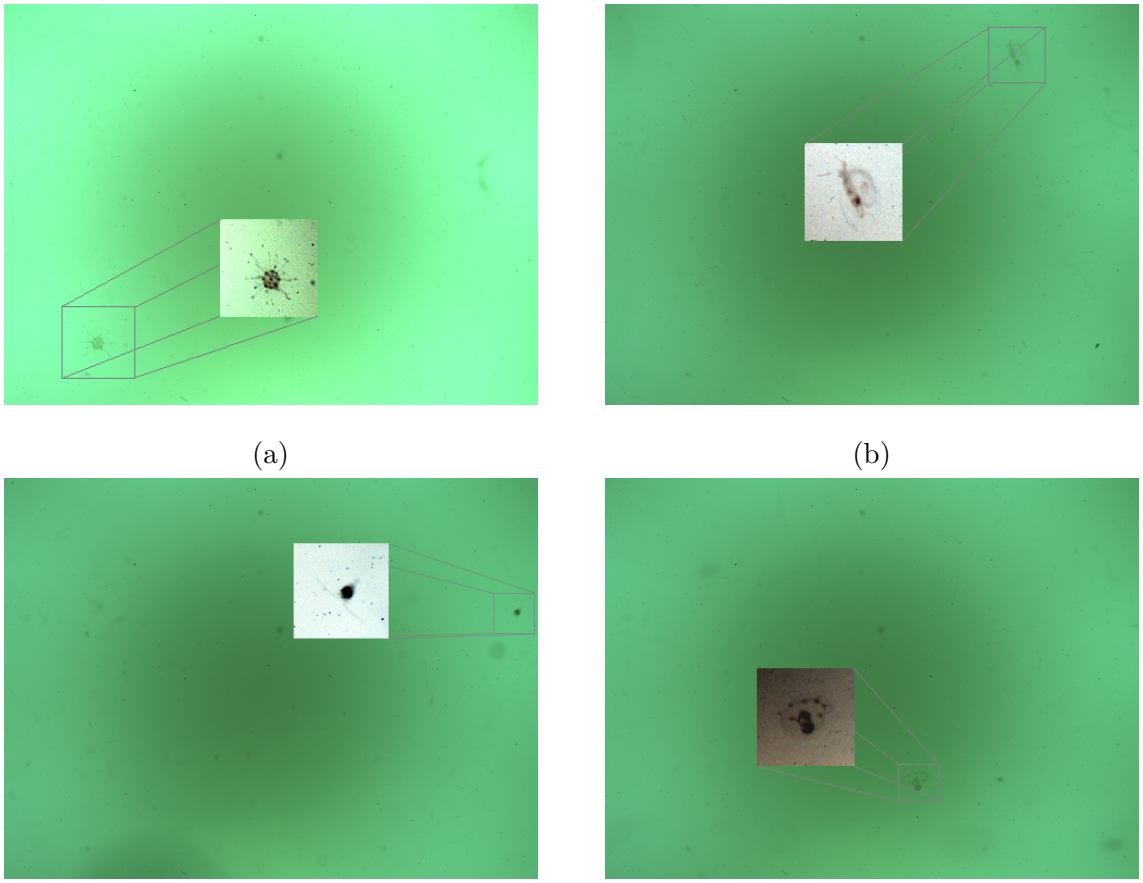
- Desenvolver uma ferramenta de melhoramento de imagem para imagens subaquáticas

- Identificar potenciais objectos de estudo e seleccionar as imagens que os contenham
- Implementar o sistema em ROS
- Desenvolver um processo de calibração autónoma do sistema

Figura 1.1: Objectivo Geral do Projecto

1.3 Estrutura da Tese

Esta tese está dividia em sete capítulos.
O capítulo 1 apresenta a motivação, uma breve introdução ao problema e os objectivos a cumprir no final deste projecto.
O capítulo 2 incide sobre o estado da arte no que diz respeito a técnicas de visão e detecção de objectos.
O capítulo 3 explica uma das funções principais do programa desenvolvido responsável pela implementação de um algoritmo de melhoramento de imagens.
O capítulo 4 detalha a função de detecção de objectos em imagens melhoradas pela função do capítulo anterior.
O capítulo 5 apresenta a solução implementada para que o programa desenvolvido nos capítulos 3 e 4 seja implementado em tempo real.
O capítulo 6 demonstra os testes feitos no terreno com o sistema implementado a processar imagens em tempo real.
O capítulo 7 fecha esta tese apresentando uma conclusão sobre o trabalho feito e trabalho futuros.



(a) (b)

(c) (d)

Figura 1.2: Imagens originais captadas pelo sistema *MarinEye* com imagens em destaque resultantes do melhoramento de imagem do sistema desenvolvido nesta tese. (a) Fitoplancton, (b) Larvacea, (c) Hydrozoa, (d) Copepod

Capítulo 2

Estado da Arte

2.1 Introdução

Este capítulo aborda soluções de processamento de imagem, versa sobre métodos de detecção de objectos, indivíduos ou grupos e métodos de melhoramento de imagem.

2.2 Fundamentos

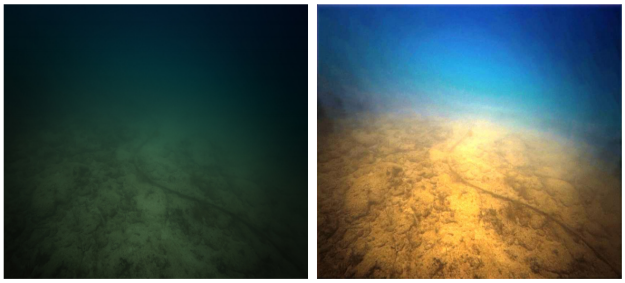
2.2.1 Normalização

Normalização é um método de processamento de imagem que muda o espectro de valores de intensidade que um pixel pode ter. É um método utilizado para melhorar fotografias com mau contraste(demasiada luz ou falta dela), algo que é muito comum em imagens captadas debaixo de água. O processo de normalização pode também ser referido como um processo espalhar o contraste de uma imagem ou espalhar um histograma, no sentido de estender, alongar[3]. O método identifica os pixeis de maior intensidade, por exemplo, 138 e passa-o para 255 e de menor intensidade, por exemplo, 38 e passa-o a 0 de forma a destacar os pontos de interesse numa imagem.

2.2.2 Características de Interesse (*Features*)

Algumas das principais características que abrangem a maior parte das imagens são arestas (*edges*), cantos ou áreas. A aresta é a característica mais básica de uma imagem e a mais simples de detectar com um algoritmo e não é mais do que um grupo de pixeis que muda de cor ou intensidade repentinamente. *Edge detection* é um dos métodos para detectar e segmentar as arestas de uma imagem baseando-se na descontinuidade de pontos cinzentos[5].

Um canto é o encontro de duas *edges*, duas semi-rectas com uma intercepção. Uma área, *blob*, é um conjunto de pixeis próximos uns dos outros. Neste projecto, a detecção



(a) (b)

Figura 2.1: Exemplo do efeito de normalização aplicado a uma imagem subaquática. (a) imagem original (b) imagem normalizada[4]

foi efectuada maioritariamente recorrendo a *edge detection* e detecção de áreas.

2.2.3 *Edge Detection*

O método de *edge detection* é uma técnica de segmentação de imagem que detecta uma linha, uma fissura ou o fim de um segmento (*edge*) e assinala estes grupos de pontos. O principal propósito deste método é simplificar a informação contida numa imagem de forma a aumentar a velocidade de processamento da mesma[6].

2.2.4 *Canny Edge Detection*

Este método executa 4 passos: É Feito *denoise* (redução de ruído) da imagem antes de detectar as *edges* utilizando o filtro gaussiano(*Gaussian Smoothing Filter*, filtro de Sobel) presente na equação 1.2 para reduzir o ruído. De seguida, é calculado o gradiente de amplitude e de direcção, este segundo, por norma, utiliza os ângulos 0, 45, 90 e 135, como ilustra a equação 1.2 onde Gx e Gy são um par de *convolution arrays*.

$$G(\mathrm{x}, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Equação 1.1: Gaussian Smoothing Filter[5]. x,y posição de um pixel na imagem, σ desvio padrão

$$G = \sqrt{G_{\mathrm{x}}^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Equação 1.2: Gradiente de amplitude (G) e de direcção (θ) [5], filtro de Sobel

O passo seguinte é supressão de todos os pixeis que não fazem parte de *edges*, o que

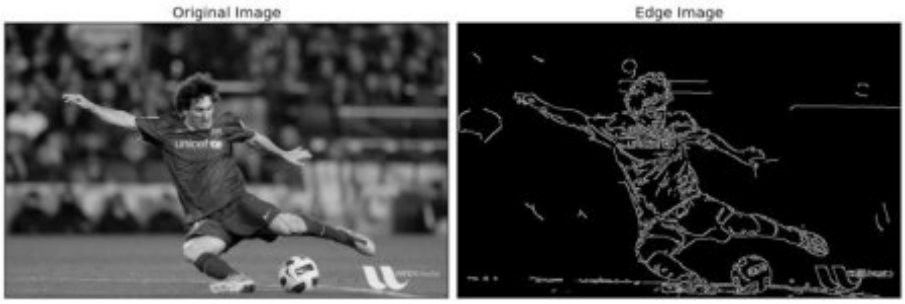


Figura 2.2: Exemplo de Implementação de *Canny Edge Detection* [5]

deixa a imagem com algumas linhas.

Por último, é aplicado o *threshold* e um múltiplo inteiro desse valor introduzido pelo utilizador como valor mínimo e máximo da histerese que escolhe manter ou excluir pixeis, o que resulta na imagem final apenas com detecções de *edges*. Se a amplitude da posição de um pixel é maior que o *threshold* máximo, o pixel é mantido e categorizado como *edge*, se a amplitude de um pixel é menor que o *threshold* máximo, o pixel é eliminado. Se a amplitude da posição de um pixel está entre os dois *thresholds*, o pixel só é mantido se estiver junto a um pixel maior que o *threshold* máximo[7].

2.2.5 Extracção de características

Scale Invariant Feature Transform

O SIFT (*Scale Invariant Feature Transform*) é um detector de características locais para imagens[8] e tem esse nome porque utiliza diferença de Gaussianas[9] e transforma a informação contida numa imagem em coordenadas invariantes em escala (*scale-invariant*) relativamente às características locais detectadas.

Esta técnica tem a vantagem de gerar (detectar) um grande número de características, por exemplo, uma imagem de 500 por 500 pode gerar 2000 características estáveis se o conteúdo o proporcionar (em imagens sem grandes características visíveis detectar-se-iam menos). A quantidade de características detectadas é importante para a detecção de um objecto já que, para um objecto pequeno numa imagem com bastante ruído (fundo repleto de outros elementos distintivos facilmente detectáveis) é necessário que, pelo menos, 3 características locais sejam correspondestes a esse objecto para uma identificação fiável[10]. Outro factor que torna esta técnica viável é o facto desta abordagem transformar a imagem analisada numa biblioteca de características locais invariantes a transformações, rotações e mudanças na iluminação[8].

O SIFT utiliza um método semelhante *scale-space filtering*, método que utiliza LoG (*Laplacian of Gaussian*) com vários valores de σ, *Gaussian kernel of width.* a utilização de Log funciona como um detector de áreas (*blob detector*) que detecta área com o tamanho

definido por σ. O σ funciona como um parâmetro de escala, neste caso. O SIFT utiliza diferença de Gaussianas, uma aproximação de LoG, porque o método unicamente com LoG requer um processamento elevado. O método utilizado pelo SIFT é obtido aplicando *blur* a duas imagens com diferentes σ e vendo a diferença entre elas. Este processo é feito com *Gaussian Pyramid*[10][11], processo ilustrado pela figura 2.3.

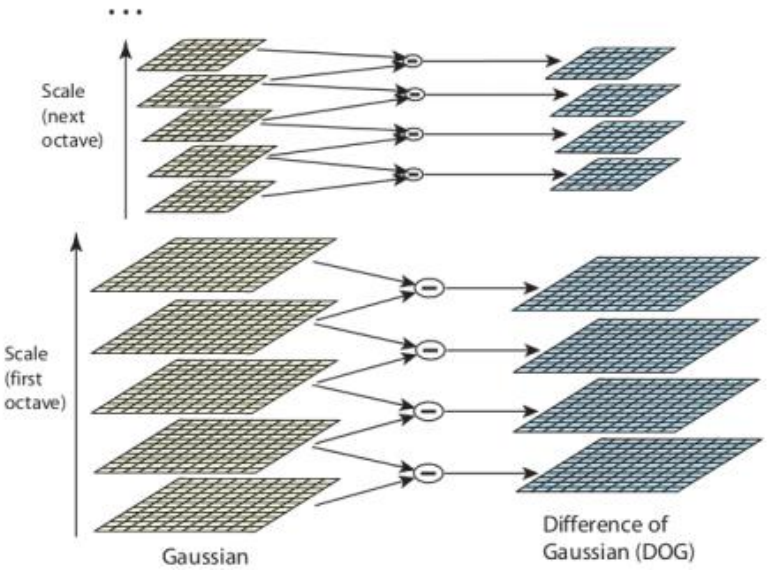


Figura 2.3: *Gaussian Pyramid*:Para cada degrau de *scale space* é utilizado *smoothing* com gaussianas na imagem inicial de forma a criar um conjunto de *scale spaces* com imagens apresentadas à esquerda. Imagens que sejam adjacentes são subtraidas e o resultado é a diferença de gaussianas, imagem apresentada à direita. A cada iteração as imagens são reduzidas com um factor de dois e o processo é repetido [10]

Depois da diferença de gaussianas ser feita, os pixeis de uma imagem são comparados com os pixeis vizinhos e com os pixeis do degrau anterior, por exemplo, num conjunto de pixeis 3 por 3, um dos pixeis é comparado com os oito que estão à sua volta e com os nove da camada anterior. Se forem iguais, o ponto comparado é um potencial *keypoint*, o que significa que pode existir um ponto de interesse nessa imagem desse degrau.

Speeded Up Robust Features

O SURF (*Speeded Up Robust Features*) é um detector de características que, tal como o SIFT utiliza aproximações de LoG, mas recorrendo a *Box Filters*, filtros que atribuem a um grupo de pixeis o valor do pixel com maior intensidade[12], o que faz com que o tempo de processamento seja menor que com o SIFT. A figura 2.5 apresenta a utilização de *box filters* de 9 por 9.

Figura 2.4: Exemplo de SIFT. Acima as Imagens Modelo e Cenário de Estudo, Abaixo o Resultado do Reconhecimento do Modelos: os Contornos e as Características Chave Assinalados que Foram Usados para Fazer a Relação Entre Imagens[8]

2.2.6 Background Subtraction

Background subtraction é um método de detecção de objectos que se baseia no facto do fundo de um cenário de estudo como uma rua, por exemplo, não mudar, o que muda são os objectos, por exemplo, x pessoas a passar, e, com este método, é possível distinguir esses objectos e assinalá-los sem ter qualquer conhecimento ou informação prévia do que são ou da sua forma [13].

Esta técnica gera uma máscara para o *foreground* (uma imagem binária que contem pixeis correspondentes a objectos que se estão a mover à frente do fundo).Como o nome sugere, a técnica de *background subtraction* calcula a máscara ilustrada na figura 2.7 subtraindo ao *frame* que contem o objecto alvo o *frame* do fundo[14].

2.2.7 *GrabCut*

O *GrabCut* é um método de segmentação de objectos em imagens que se baseia em cortes gráficos que funciona, em termos de resultados, como o método de *background*

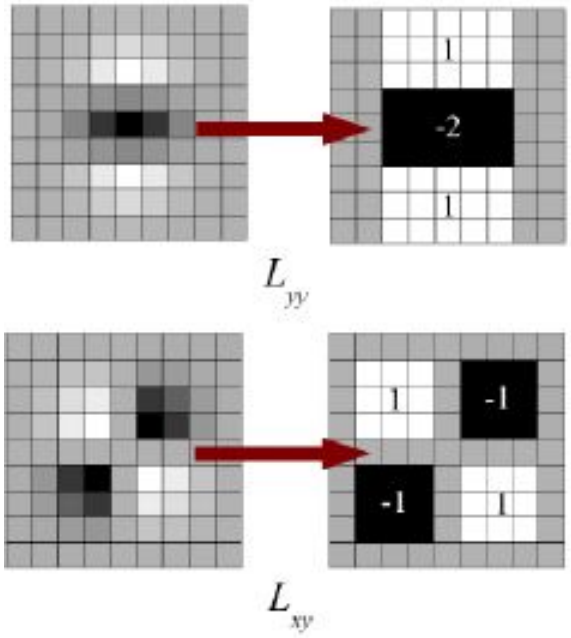


Figura 2.5: Esquerda: Derivada Gaussian Parcial de Segunda Ordem na direcção yy e xx, Direita: Aproximação Utilizando *Box Filters*. As Regiões a Cinzento São 0 (Zero).[9][12]

Figura 2.6: Exemplo de SURF. Detecção de características numa imagem[9]

subtraction, numa única imagem. Para isolar um objecto o utilizador, manualmente, assinala o objecto desejado com um rectângulo fazendo com que dentro do segmento fiquem presentes o objecto e parte do fundo a eliminar. O algoritmo baseia-se na distribuição de cores entre o objecto e o fundo utilizando um modelo de mistura de gaussianas (*gaussian mixture model*), por isso, o algoritmo tem tendência a ser mais eficaz quando há uma diferença suficientemente grande entre o primeiro plano, que contém o objecto-alvo, e o fundo. O algoritmo tem algumas falhas quando analisa padrões camuflados.[15]

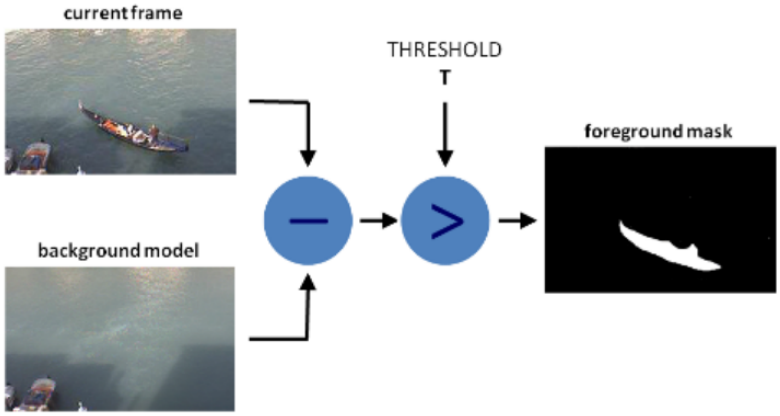


Figura 2.7: *Background Subtraction*: Máscara para o *foreground* [14]

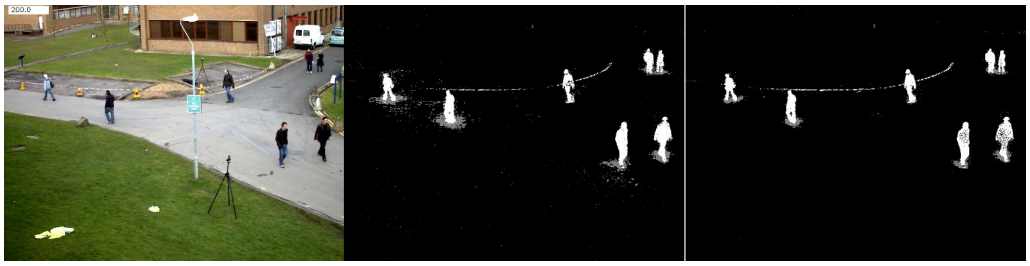


Figura 2.8: a) *Frame* Original, b) Máscara Gerada, c) Objectos Detectados [14]

Figura 2.9: Dois exemplos de *GrabCut* . O utilizador selecciona a imagem com um rectângulo à volta da imagem, incluindo o fundo, e objecto é extraído automaticamente[15]

2.2.8 Janela Deslizante

O método de janela deslizante passa por comparar uma imagem onde o objecto de interesse está contida, com uma imagem do universo onde o queremos encontrar. Este método detecta o objecto de interesse procurando características distintivas desse objecto(cantos, *edges* ou áreas) na imagem total.

Um problema comum deste método é o facto de falhar em cenários onde o objecto de referência é parecido com os restantes da imagem, por exemplo, uma estante de livros onde o objecto de interesse é um dos livros[16].

No caso deste exemplo, é necessário procurar outras características que não as referidas anteriormente, como a figura 2.10 ilustra. Um outro problema comum deste método é o facto de quando tem de percorrer uma imagem por completo, de forma a comparar todos os pontos da imagem com a imagem referência, pode demorar substancialmente mais que outros métodos.

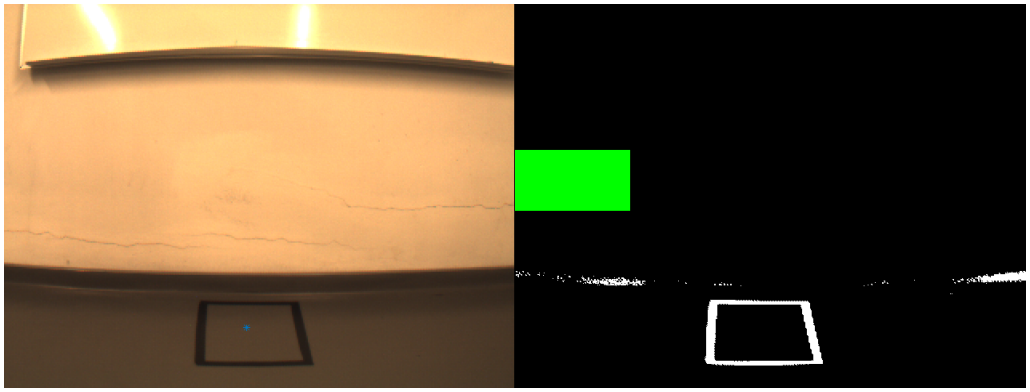
A figura 2.11 mostra um teste com janela deslizante numa simulação de um lugar de estacionamento. A janela contém um recorte da imagem original e é comparada com a imagem da direita, uma fotografia tirada nas mesmas circunstâncias, mas com um ângulo ligeiramente diferente que tem aplicadas técnicas de *edge detection* simples de modo a isolar o lugar de estacionamento. O segmento da imagem original tem os mesmos filtros aplicados.

(a) (b)

Figura 2.10: Exemplo de Detecção de Catacterísticas. (a) Imagem Original (b) Detecção de 500 Pontos de Características que não *Edges* [16]

2.2.9 Abordagens heurísticas

Abordagens heurísticas são técnicas utilizadas para informar uma rede neuronal do que é o quê dentro de uma biblioteca de imagens de forma a poder ser treinada. Este método tem a desvantagem de ter de ser feito à mão, o que acaba por ser dispendioso em termos de tempo visto que, para obter um grau de confiança aceitável depois do treino da uma rede, é necessário categorizar cerca de 300 imagens de modo a aplicar uma função de rotação a cada 10 graus para obter cerca de 10000 imagens e, aí, ter uma biblioteca com imagens suficientes para a rede ter sucesso de aprendizagem, porque para cada classe categorizada são necessárias 1000 imagens[17].



(a) (b)

Figura 2.11: (a) Imagem Original (b) Janela Deslizante a Comparar um Segmento da Imagem (a) com uma Imagem Diferente

O LabelImg é um categorizador gráfico de imagens (*graphical image annotation tool*), é escrito em *Python* e usa *Qt* na sua interface gráfica. As classificações são guardadas num ficheiro XML em fortato *PASCAL VOC*[19], o formato usado pela *ImageNet*, uma biblioteca com centenas de milhar de imagens para treino de redes neuronais[20], ou no formato da rede Yolo[18].

2.2.10 Redes Neuronais

Redes neuronais são um conjunto de algoritmos inspirados no funcionamento de um cérebro que replicam aproximadamente o funcionamento de neurónios. Estes algoritmos

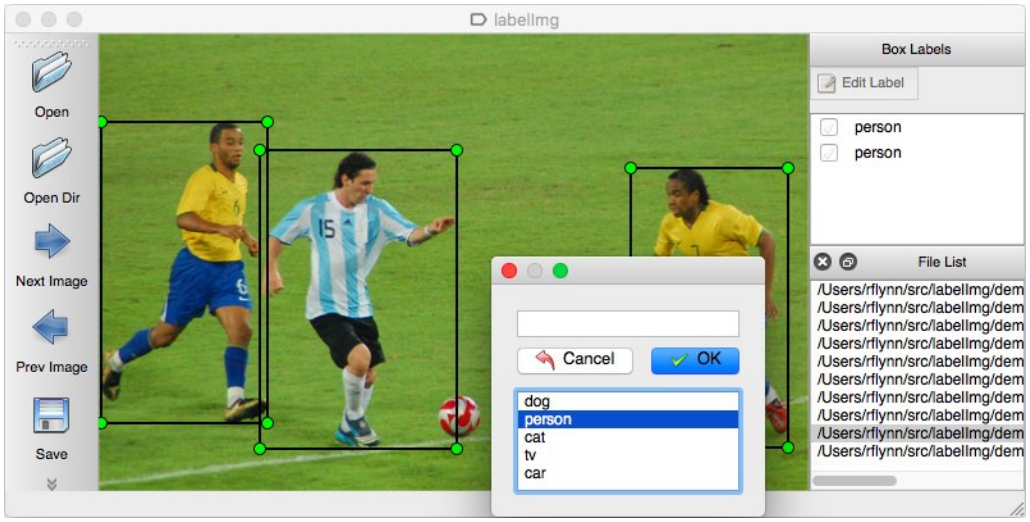


Figura 2.12: Exemplo do Uso do *LabelImg* [18]

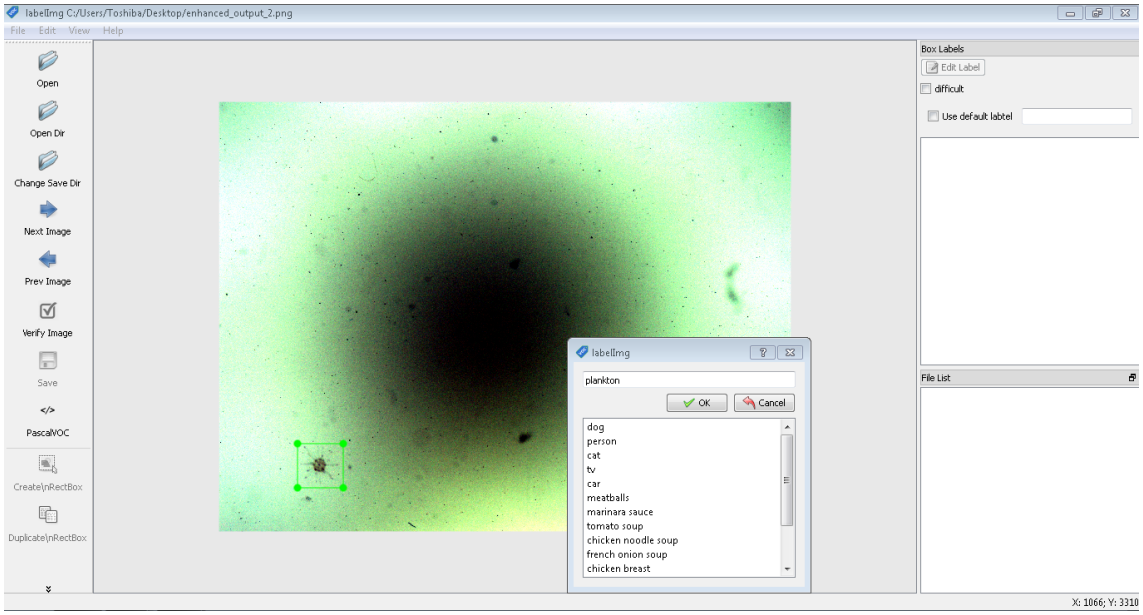


Figura 2.13: Exemplo de Categorização de Plâncton com o *LabelImg*

podem ser usados num vasto leque de áreas como processamento de imagens e vídeo, navegação, controlo, entre outras. A flexibilidade deste sistema deve-se ao facto de cada neurónio do sistema conter uma função não linear em relação à soma das suas entradas[21]. A figura 2.14 apresenta um exemplo de uma rede neuronal com duas camadas. Os pontos pretos representam os neurónios que contêm a informação e passam-na à camada do centro que, posteriormente, comunica com a última, da direita. Os neurónios decidem baseando-se em pesos: cada neurónio soma o peso dos *inputs* (entradas), aplica uma função não linear e passa o resultado à camada seguinte até à saída da rede neuronal, *output*.

Uma rede neuronal aprende ajustando os pesos da informação que recebe baseando-se em dados de treino (como um conjunto de imagens com dados caracterizados, como referido na secção anterior) e informa o utilizador da percentagem de probabilidade de sucesso consoante os pesos finais. Neste ponto, quando maior a quantidade de informação para treino, melhor serão os resultados[21], como abordado na secção anterior. A principal desvantagem de redes neuronais é a capacidade de processamento necessária para obter resultados aliada ao tempo, sendo que um factor está dependente do outro.

O *Yolo* utiliza uma única rede convolucional para prever simultaneamente múltiplas *bounding boxes* e probabilidades dos objectos dentro dessas *bounding boxes* estarem dentro de uma classe existente. Esta rede treina em imagens sem compressão e automaticamente optimiza a performance na detecção de objectos de interesse. O método usado tem melhores resultados que métodos tradicionais de detecção de objectos como *edge detection* ou janela deslizante.

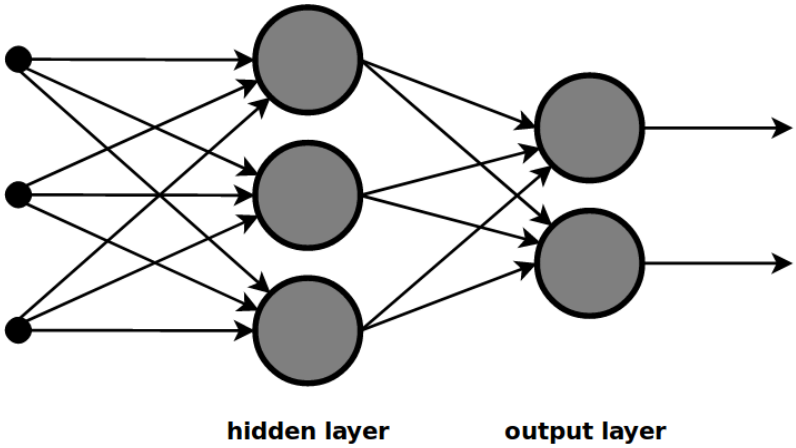


Figura 2.14: Exemplo de uma rede neuronal de duas camadas(imagem de domínio público *inWikimedia Commons*)

A versão base da rede analiza imagens a 45 fps (*frames per second*) numa Titan X GPU e a versão *fast* analiza imagens a 150 fps o que torna possível detectar objectos em tempo real em vídeos com latência perto dos 25 milissegundos. Contrariamente ao método de janela deslizante, o Yolo vê a imagem total enquanto treina e é testada, não só segmentos, por isso detecta sempre o objecto de interesse, assim como todas as outras classes presentes na imagem[22].

A figura 2.15 mostra o processo de detecção do *Yolo*. O sistema redimensiona a imagem a analisar para 448 por 448, utiliza uma rede convolucional em cima dessa imagem e apresenta o resultado através de um modelo de confiança.

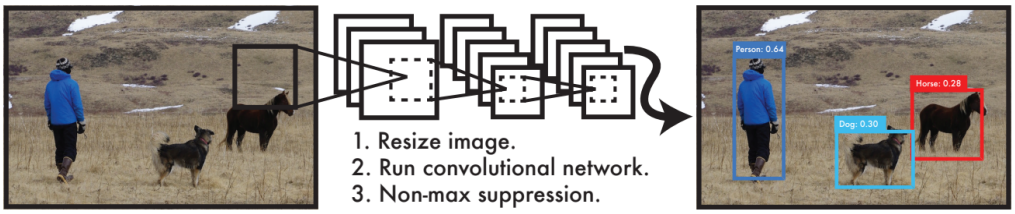


Figura 2.15: Sistema de Detecção do YOLO [22]

Ambientes que permitem treinar redes neuronais

Tensorflow

O *Tensorflow* é um sistema de machine learning open-source desenvolvida pela Google para computação numérica[23], que usa gráficos de fluxo de dados (*dataflow graphs*) para

representar operações entre estados, *shared states* e operações que mudam esses estados. Com a informação contida nos gráficos o tensorflow mapeia nós em várias máquinas num cluster de modo a ser processado por diferentes máquinas onde se incluem CPUs multicore, GPUs e TPUs (*Tensor Processing Units*).

Esta arquitectura dá ao utilizador uma flexibilidade de abordagem a um problema. Ao passo que, em abordagens de *parameter server* a gestão de shared states é algo inerente ao sistema, o *tensorflow* permite ao utilizador experimentar outras abordagens de optimização e de algoritmos de treino de redes. O tensorflow pode ser usado em vários cenários, mas foi desenvolvido com foco no treino de *deep neural networks*[24].

Caffe

O *Caffe* é uma biblioteca *open-source* de *deep architectures*.É escrito em C++ e implementada em *Matlab* e *Python*. Foi desenvolvido por Y. Jia, BVLC *center* e oferece ao utilizador uma rede indicada para investigação devido à sua separação de definição de rede neuronal e implementação da mesma[25].

2.3 Técnicas Aplicadas

2.3.1 Detecção de Objectos Debaixo de Água

Alex Raj , Claris Jose e Supriya M. H. utilizam *canny edge detection* em imagens captadas e processadas por uma FPGA *Altium NanoBoard 3000*. Os resultados disponíveis mostram que a técnica é testada em imagens simples de apenas um objecto: um peixe ou uma alforreca como pode ser visto nas figuras 2.19 e 2.20. O algoritmo aplicado para obter estas detecções está presenta na figura 2.16. Este método tem a vantagem de não ser computacionalmente exigente, visto que *canny edge detection*, só por si, é bastante leve, o que faz com que seja possível o processamento das imagens ser efectuado por uma FPGA e em tempo real. A maior desvantagem deste método é a complexidade do desenvolvimento em FPGA. Um factor que faz com que este método não seja muito versátil é o facto de não ser possível implementar ROS em FPGAs sem recorrer a um sistema externo complementar(algo que comunicasse com a FPGA por RS232, por exemplo, e processasse a informação, passando-a posteriormente para ROS), e ROS é algo comum na área da robótica para que sistemas corram em tempo real.

Fabio Oleari, Fabjan Kallasi, Dario Lodi Rizzini e Jacopo Aleotti implementam ROS em projectos que utilizam imagens subaquácticas. A solução implementada no artigo utiliza detecção de objectos e ROS. A segmentação de um objecto é apresentada nas figuras 2.21 e 2.22, onde são isolados dois canos numa imagem subaquática. Para a aquisição de imagens

stereo foi utilizado um *driver* de uma câmara integrado em ROS e, de forma a etiquetar as imagens cronologicamente, foi utilizado um *timer* de ROS para ser possível colocar um *timestamp* em cada *frame*[26]. Esta solução tem como vantagem a utilização de ROS para captar imagens, o que faz com que a velocidade de captação seja elevada, algo que tem em comum com o *MarinEye* (o *setup* actual do *MarinEye* consegue captar imagens até 7 *frames* por segundo).

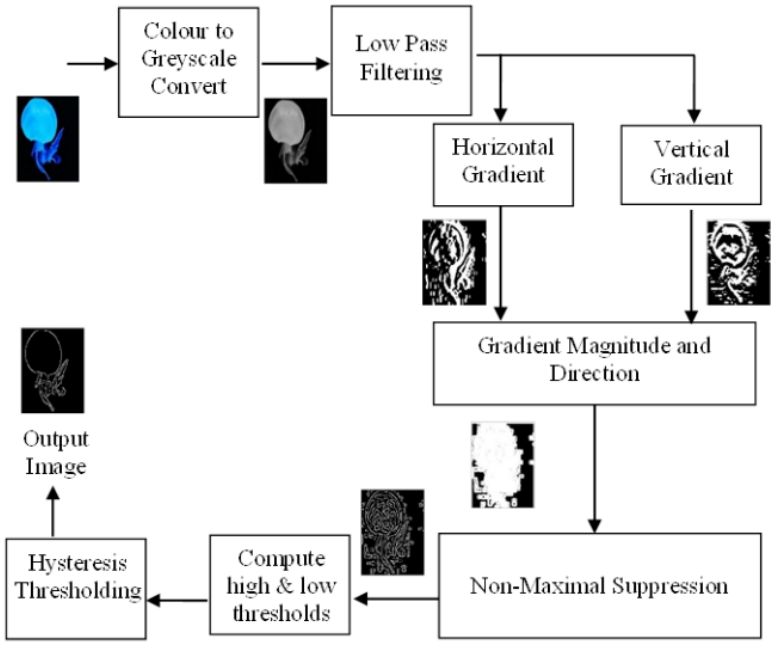


Figura 2.16: Algoritmo de detecção com *canny edge detection*[27]

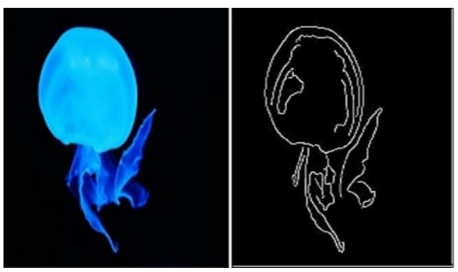


Figura 2.17: *Canny edge detection* aplicado debaixo de água a uma alforreca[27]

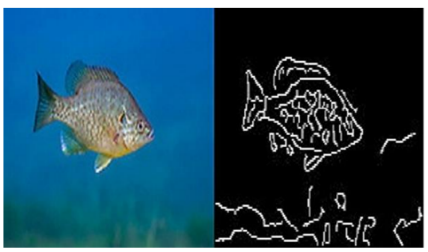


Figura 2.18: *Canny edge detection* aplicado debaixo de água a um peixe[27]

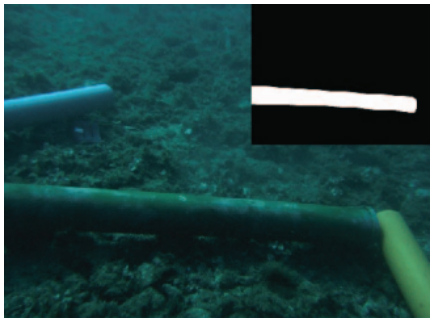


Figura 2.19: Segmentação de um cano debaixo de água[26]

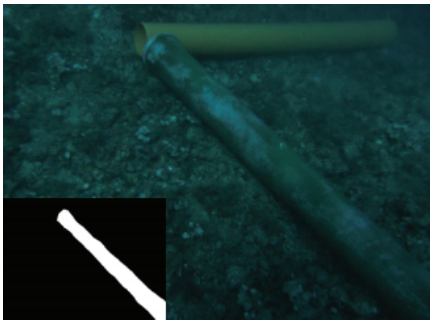


Figura 2.20: Segmentação de um cano debaixo de água[26]

Wenwei Xu e Shari Matzner utilizam algumas técnicas que utilizam redes neuronais, mais concretamente a terceira versão do YOLO, Yolov3, e o *Tensorflow* para estudar os peixes detectados em vídeos debaixo de água. Os resultado das detecções pode ser observado na figura 2.21, que mostra detecções correctas e falsos positivos[28]. Esta técnica tem a vantagem de ter muito bons resultados, principalmente se forem captadas mais imagens e forem introduzidas na biblioteca de imagens de estudo da rede neuronal. Eventualmente, todos os peixes seria detectados por este método. A grande desvantagem deste método é a necessidade de *hardware* com um elevada capacidade de processamento, o que impossibilita a implementação em sistemas de baixo custo como sistemas embebidos ou mesmo um computador comum se o projecto tiver em vista a implementação em tempo real.

Zhe Chen, Zhen Zhang, Yang Bu e Fengzhao Dai utilizam étodos de extracção de características de interesse[29] de forma a obter um resultado semelhante ao de *backgroud subtraction*. O resultado é visível na figura 2.23.

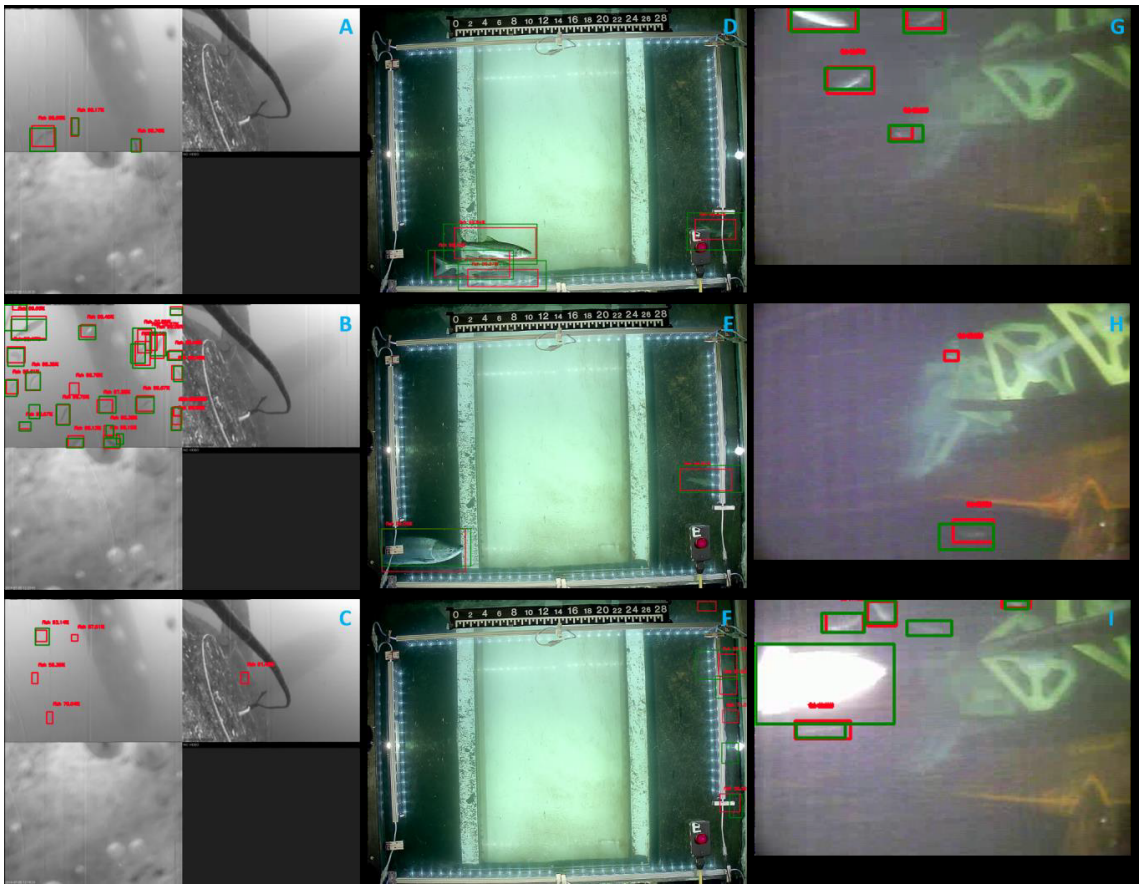


Figura 2.21: As caixas verdes são categorizações feitas previamente, as caixas vermelhas são resultados da rede neuronal com peso considerado viável. As figuras A, B, C, G, H e I mostram imagens desfocadas (devido ao *blur* natural de imagens tiradas debaixo de água) que contêm peixes pequenos algo difíceis de distinguir do fundo. As figuras D e E mostram detecções correctas. A figura F mostra a dificuldade de detecção da rede quando existe apenas uma imagem parcial do objecto alvo[28].

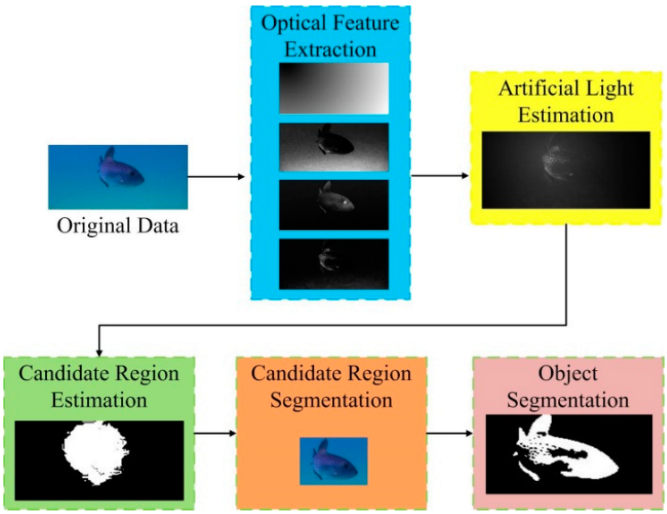


Figura 2.22: Sequência de métodos de segmentação de objectos debaixo de água testados[29]

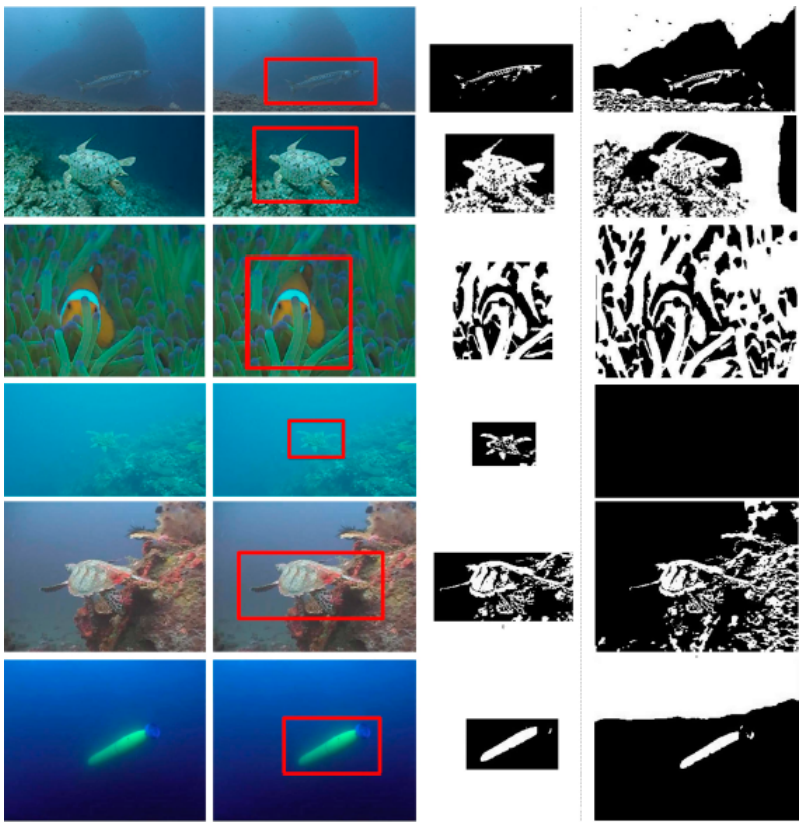


Figura 2.23: Fases do processo de isolamento de um objecto da esquerda para a direita: imagem original, candidato a região de interesse manualmente destacada, objecto isolado resultante, objecto isolado sem região destacada [29]

2.3.2 Tratamento de imagens subaquácticas

Saeed Anwar, Chongyi Li e Fatih Porikli utilizam combinações de técnicas de melhoramento de imagens a redes neuronais de forma a calibrar os parâmetros necessários para obter um resultado ideal para cada cenário[4]. Neste artigo pode observar-se o melhoramento substancial da qualidade obtida no produto final quando existe uma calibração de parâmetros feita por redes neuronais. Os resultados finais podem ser observados nas figuras 2.26 e 2.27. A primeira parte deste método, a normalização, pode ser executada sem uma necessidade elevada de processamento e, se a aparência das imagens alvo for semelhante, algo que acontece nas imagens recolhidas com o *MarinEye*, os resultados são bastante positivos e este método pode ser viável para o projecto desenvolvido nesta tese.

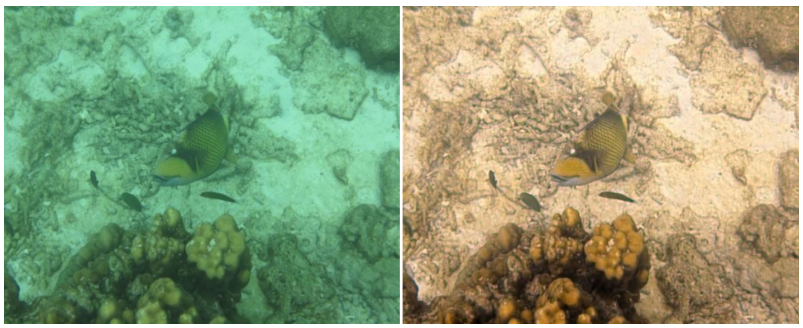


Figura 2.24: (a) Imagem original (b) Resultado do algoritmo de tratamento de imagem[4]

(a) (b)

Figura 2.25: (a) Imagem original (b) Resultado do algoritmo de tratamento de imagem[4]

Codruta O. Ancuti , Cosmin Ancuti, Christophe De Vleeschouwer e Philippe Bekaert utilizam normalização para melhorar imagens subaquáticas captadas por um par *stereo* e SIFT para detectar características de interesse[30]. Esta solução tem como pontos positivos o uso de normalização, pela sua eficácia, e a qualidade dos resultados, mas a utilização do SIFT requer algum poder de computação, o que pode ser um entrave quando se quer manter o sistema leve. Os resultados deste artigo podem ser vistos na figura 2.26.

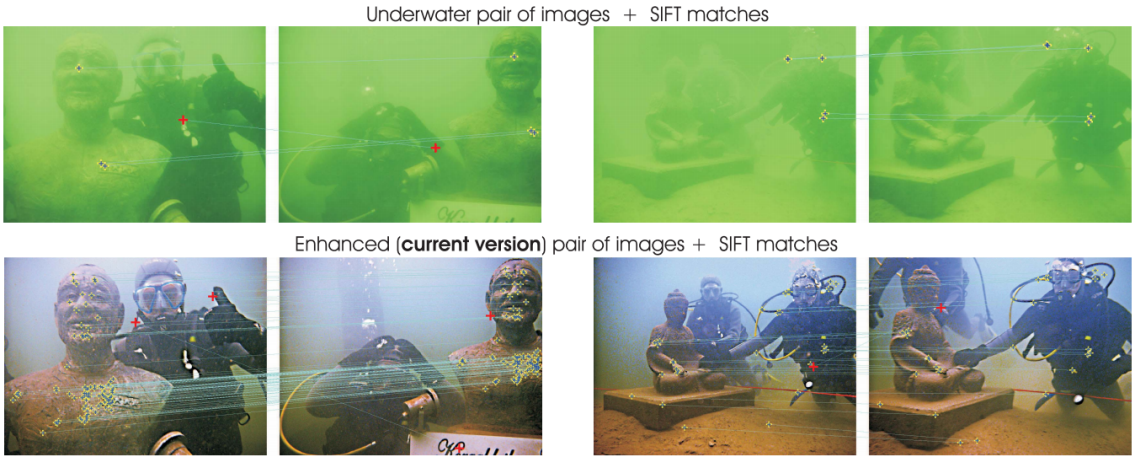


Figura 2.26: Em cima, dois pares de imagens originais. Em baixo, imagens melhoradas e características de interesse detectadas[30]

2.3.3 Sistema de captação de imagens subaquáticas de alta definição para zooplâncton

O *MarinEye* é um projecto resultante de uma parceria do INESC TEC, do ISEP, do CIIMAR, da FCUP, do MARE e do IPMA que consiste na criação de um sistema de captação de imagens subaquáticas e estudo de plâncton recorrendo a sistemas autónomos[31]. No âmbito do projecto, foi desenvolvido um sistema autónomo de captação de imagens subaquáticas presente no artigo *Autonomous High-Resolution Image Acquisition System for Zooplankton*[32] onde João Resende et al utilizam uma *Odroid XU4* para processamento do *software*. Para o controlo da câmara é utilizado um *Single Board Computer* que comunica com a câmara por USB 3.0. A câmara utilziada foi uma IDS (UI-3590CP-C-HQ Rev.2) que consegue atingir 15 *frames* por segundo, mas devido ao facto de utilizar uma *odroid* para o processamento e as imagens que capta têm as dimensões de 4912 por 3684 pixeis, a taxa de aquisição fica reduzida a 5 *frames* por segundo[32].

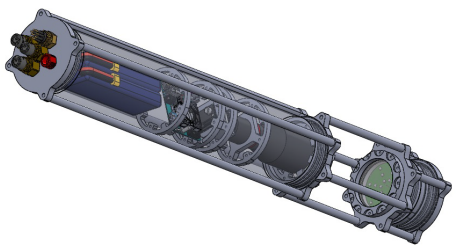


Figura 2.27: *Design* do sistema de captação de imagens subaquáticas de alta definição para zooplâncton[32]

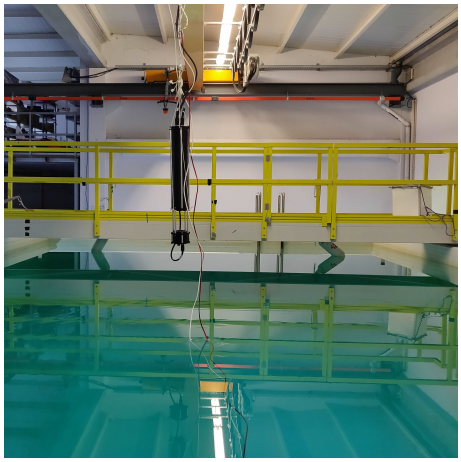


Figura 2.28: Teste com o sistema de captação de imagens desenvolvido no âmbito do projecto *MarinEye*[32]

Figura 2.29: Sistema em acção em modo manual numa missão realizada no rio Douro(Avintes)[32]

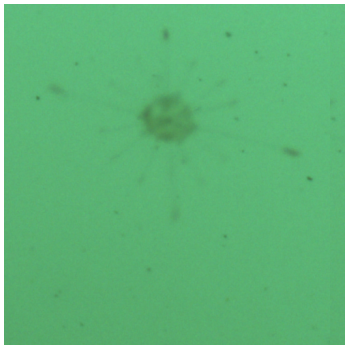


Figura 2.30: Fitoplâncton captado pelo sistema de captação de imagens desenvolvido no âmbito do projecto *MarinEye*[32]

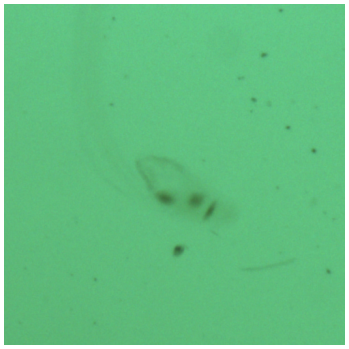


Figura 2.31: Larvacea captada pelo sistema de captação de imagens desenvolvido no âmbito do projecto *MarinEye*[32]

2.3.4 Sistema de detecção de classificação de zooplânction em tempo real

Pedro Geraldes, Joel Barbosa et al[33] utilizam redes neuronais para detectar e classificar plâncton em tempo real com uma velocidade de processamento de 7 *frames* por segundo. Este sistema de pós processamento foi utilizado com modelos anteriores utilizados no projecto *MarinEye*, bem como com imagens captadas pelo sistema desenvolvido descrito no artigo *Autonomous High-Resolution Image Acquisition System for Zooplankton*. Este projecto experimentou com duas redes neuronais: a *Google MobileNetSSD*, uma implementação no *Caffe* que treinou durante três dias numa GPU *Jetson* TX2 e obteve uma média de 75% de precisão; e a ZooplanktoNet que treinou dois dias e obteve uma precisão de 83%[33]. Os resultados podem ser observados nas figuras 2.35 e 2.36 e as curvas de precisão nas figuras 2.37 e 2.38.

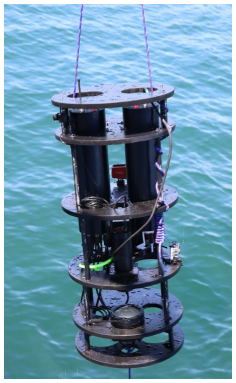


Figura 2.32: *MarinEye* - Sistema de captação de zooplâncton[33]

Figura 2.33: Resultados da detecção e classificação[33]

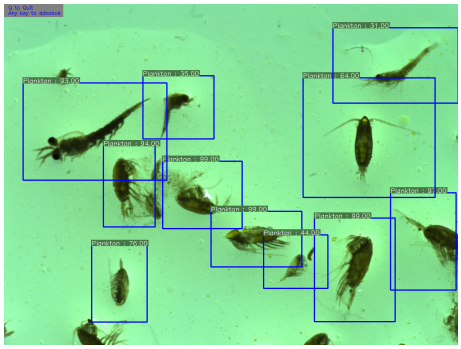


Figura 2.34: Resultados da detecção e classificação[33]

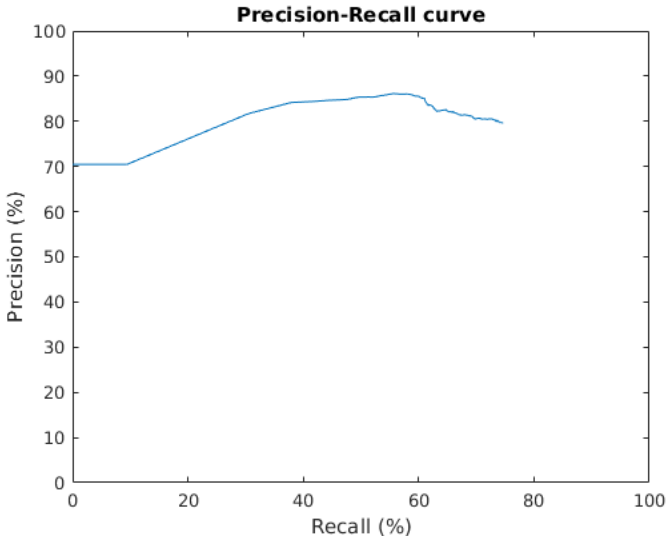


Figura 2.35: Curva de precisão da rede MobileNet-SSD[33]

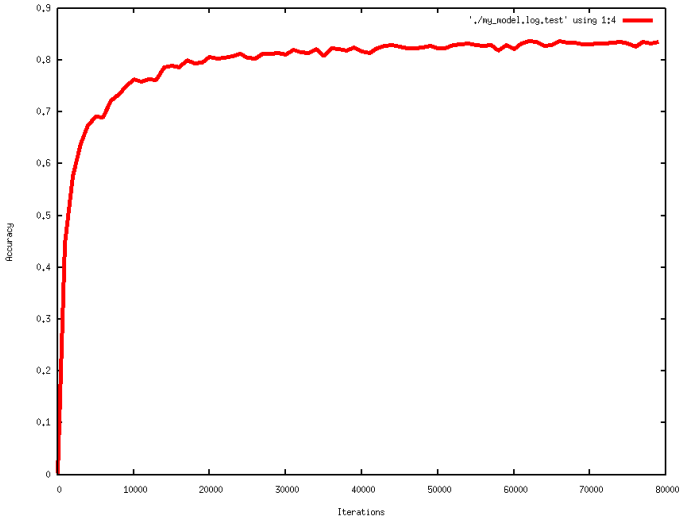


Figura 2.36: Curva de precisão da rede ZooplanktoNet[33]

Capítulo 3

Métodos de Pré-Processamento de Imagem

3.1 Introdução

Dado o ruído presente em imagens subaquáticas foi necessário desenvolver um sistema que o corrigisse de forma a poder efectuar-se detecções nessas mesmas imagens. Este capítulo explica o que a função de melhoramento de imagem faz.

3.2 Arquitectura do sistema

Figura 3.1: Diagrama da Função de melhoramento de imagem

3.3 Implementação

Foi implementado um algoritmo de normalização numa imagem modelo de forma a testar a resposta de uma imagem com alguma luz, algo que não era esperado obter nas imagens de estudo. A imagem usada foi apenas um teste, visto que o grupo de imagens alvo continham seres de pequena dimensão, nada como a imagem apresentada (figura 3.2).

O teste revelou que a imagem ficava com mais luz e os objectos que ficavam em destaque eram os peixes. Apesar da imagem ficar, no seu todo, mais clara, os peixes ficavam mais

nítidos e detectáveis.

(a) (b)

Figura 3.2: (a) Imagem Original (b) Imagem Normalizada

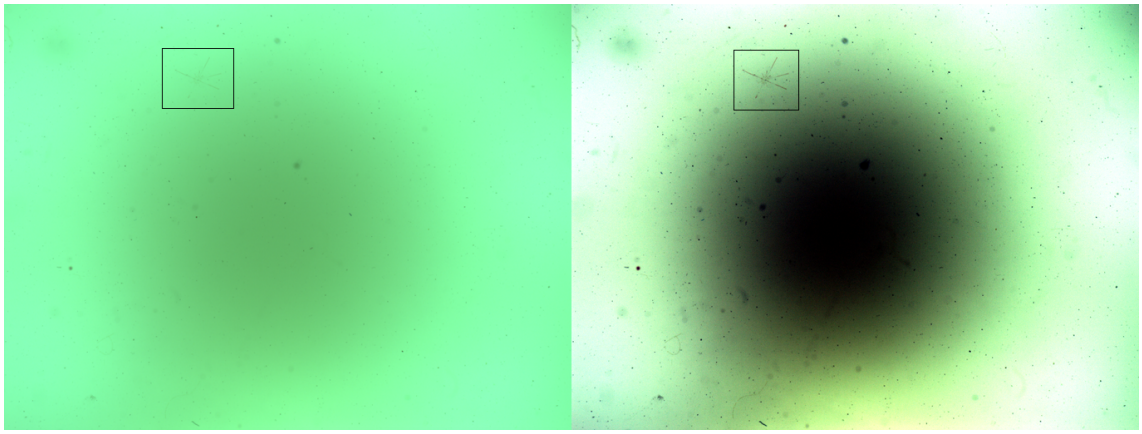
Para um segundo teste foram usadas imagens de uma missão para o projecto *Marin-Eye* que continham plâncton quase indetectável a olho nu (figura 3.3 lado esquerdo). O algoritmo acima referido foi utilizado apenas com uma mudança de *threshold* às imagens recebidas (figura 3.2 lado direito). Os resultados foram bastante positivos, logo à partida, apesar das imagens recebidas não estarem perfeitas devido a um erro na concepção do sistema de iluminação que fez com que o centro de todas as fotografias tiradas não fosse iluminado de forma conveniente e, por isso, todas as imagens depois de normalizadas terem um centro negro, mesmo assim, é possível observar 70% da imagem e verificar que os seres presentes nelas são visíveis.

Na figura 3.2 é visível na parte central do topo da imagem um ser praticamente invisível na imagem original.

3.3.1 *Filter Factor*

Após tentativas de valores do factor do filtro de normalização, chegou-se ao valor ideal 1. O factor é multiplicado pela quantidade de uma cor no histograma e somado a ele próprio, por exemplo, para o verde $hist\mathbf{g}[\mathrm{i}] = hist\mathbf{g}[\mathrm{i}] + hist\mathbf{g}[\mathrm{i}-1]$, até $i = 255$, para BGR (*Blue Green Red*).

Como se pode observar na figura 3.4 (de notar que os problemas e sucessos apresentados nestes dois *frames* estendem-se aos restantes capturados na missão), o factor a 0.5 e todos os valores anteriores tendem a dar mais destaque à zona escura central, mas a delinear bem os objectos para a sua detecção. Este valor traz 2 problemas. Um dos problemas relaciona-se com o *hardware*, que provoca o círculo central, mas resolver esse problema de iluminação na parte central não mudaria o segundo problema. Neste exemplo de apenas



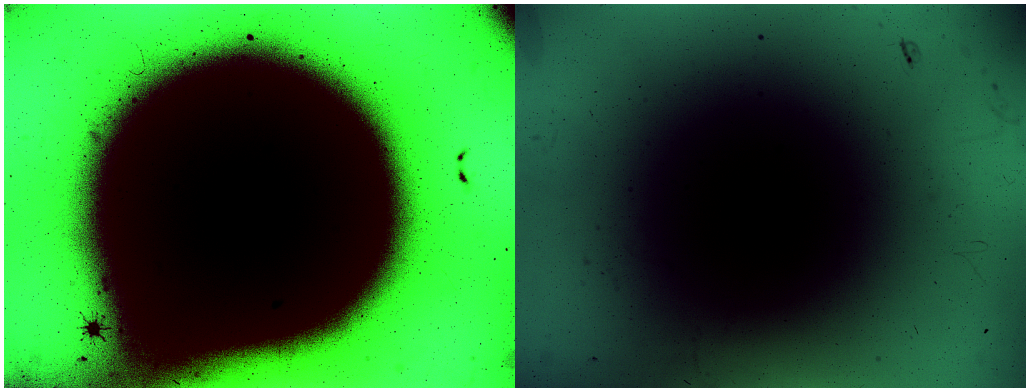
(a) (b)

Figura 3.3: Imagens obtidas numa missão do projecto *MarinEye*. (a) Imagem original (b) Imagem passada pelo algoritmo de normalização

dois *frames*, este valor tende a ser apenas interessante para o *frame* 2, porque no *frame* 4, de características ligeiramente diferentes, o filtro torna a imagem muito mais escura e, mais à frente, impossível de ser usada para detecção de objectos devido à falha de detecção do *canny edge detection*.

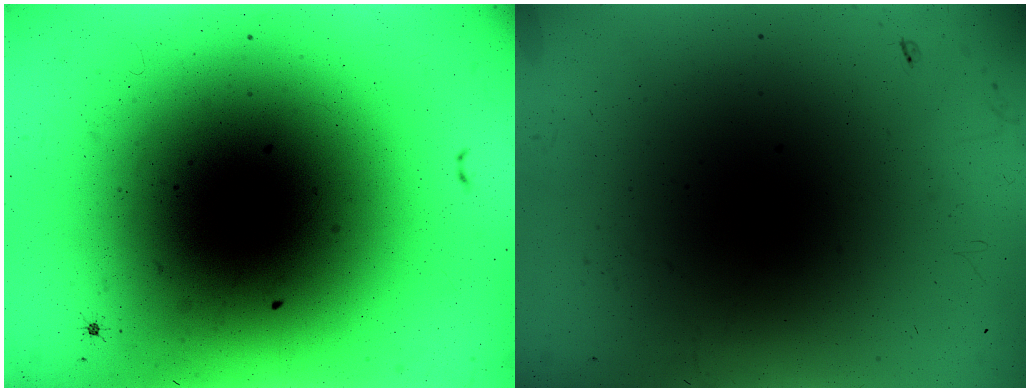
A figura 3.5, mais uma vez ilustra um valor perto do ideal para o *frame* 2, mas, como no valor anterior, torna aproximadamente metade das imagens recolhidas escuras e, posteriormente, indetectáveis nos algoritmos apresentados na secção de detecção de objectos.

Depois de vários testes chegou-se ao valor ideal, 1. O brilho não é excessivo, o contraste é suficiente para as detecções funcionarem e todos os objectos/indivíduos presentes nas imagens são detectáveis, requisito necessário para serem categorizados na fase seguinte do projecto *Marineye*, mas, mais que isso, são visíveis, algo que torna muito mais simples a avaliação do sucesso de uma missão à medida que ela decorre. Os resultados com este valor são apresentados na figura 3.6.



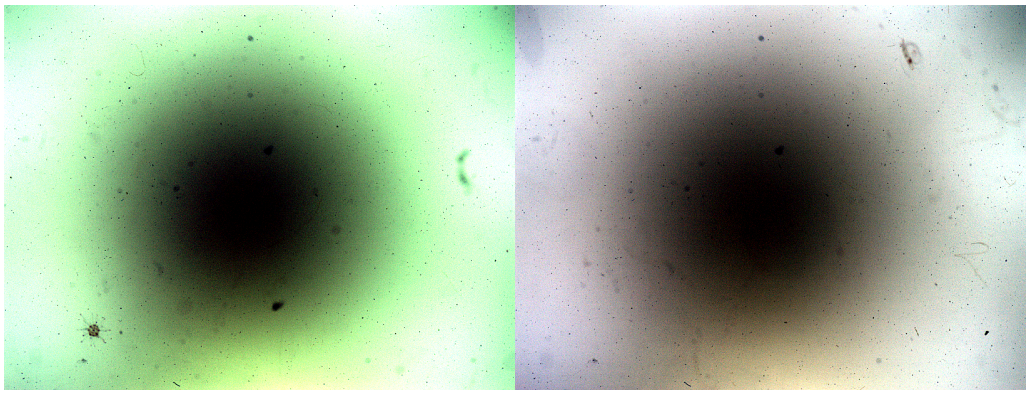
(a) (b)

Figura 3.4: (a) *Frame* 2 (b) *Frame* 4 com *Filter Factor* $= 0.5$



(a) (b)

Figura 3.5: (a) *Frame 2* (b) *Frame* 4 com *Filter Factor* $= 0.9$



(a) (b)

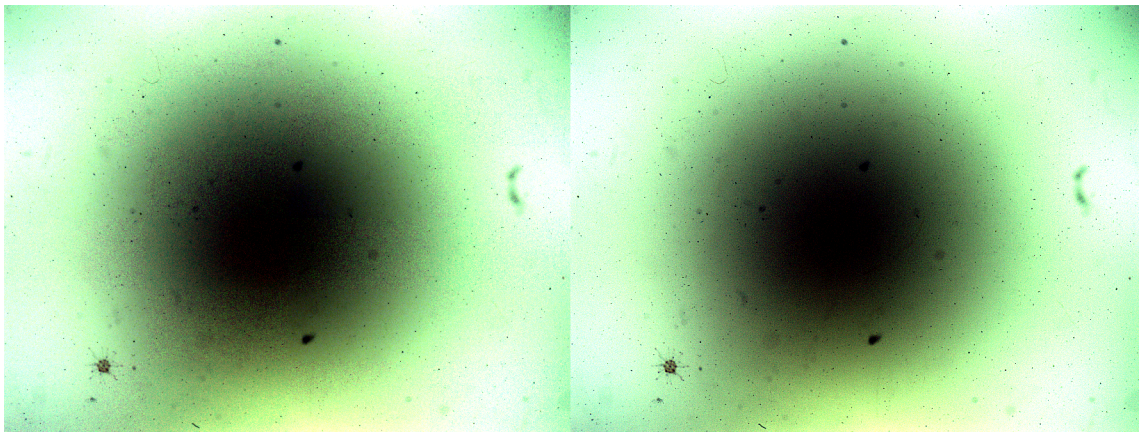
Figura 3.6: (a) *Frame 2* (b) *Frame* 4 com *Filter Factor* = 1

Valores acima de 1 fazem a imagem voltar ao seu estado original de aparência de nevoeiro e escuridão onde a detecção é impossível.

3.3.2 Qualidade da Imagem Final

Numa primeira fase, para a análise da imagem de entrada e detecção de objectos, a imagem original era mantida numa variável e um clone, copiado para outra matriz, era reduzido utilizando a função *resize* do *OpenCV* para que o processamento fosse mais rápido, melhorado e enviado para a detecção. Se a detecção retornasse *true*, a imagem original, mantida na variável originalmente referida, era enviada para a função de melhoramento e o resultado era guardado com o tamanho original, 4912 x 3684.

A rede neuronal utilizada no projecto *Marineye* interpreta imagens um pouco acima de 980 x 736, por isso, foi feita a experiência de melhorar a imagem, caso houvesse o mesmo retorno de verdadeiro referido acima, depois de a reduzir para 982 x 736 (tamanho original dividido por 5, valor inteiro usado para manter as proporções originais da imagem) e o resultado foi superior ao método anterior, como pode ser observado na figura 3.7. A Imagem da esquerda tem algum ruído, ao passo que na da direita esse ruído é bastante menos significativo e a imagem apresenta-se com o tamanho ideal para ser processada por uma rede neuronal logo à saída do programa, o que tornou esta solução positiva em duas vertentes.



(a) (b)

Figura 3.7: (a) Imagem sem *Resize* prévio (b) Imagem com *Resize* prévio

Capítulo 4

Detecção de Objectos

4.1 Introdução

Após melhorar a qualidade da imagem é necessário proceder à detecção de objectos. Este capítulo explica o que a função de detecção de objectos faz.

4.2 Arquitectura do sistema

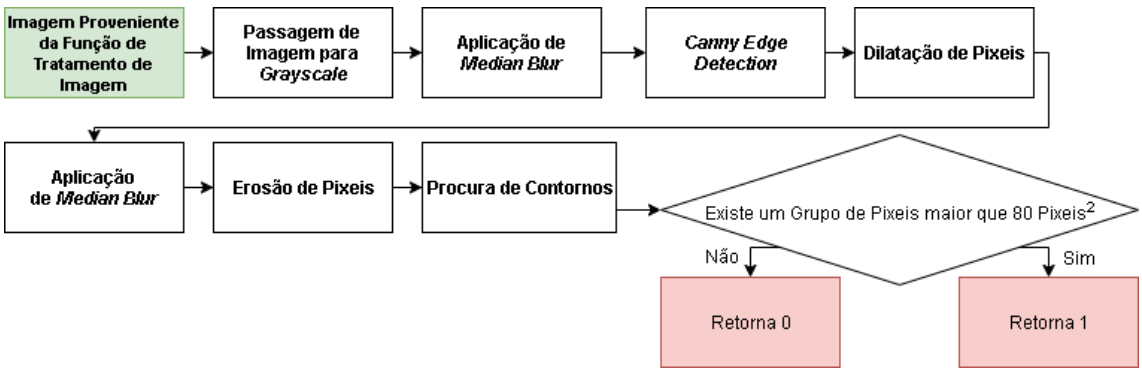
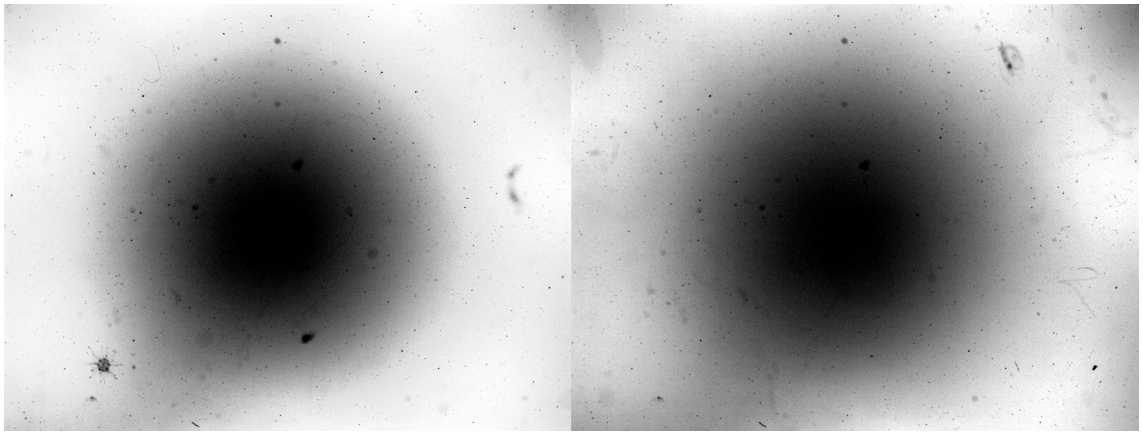


Figura 4.1: Diagrama da Função de detecção de objectos

4.3 *Grayscale*

O primeiro passo do processo de detecção foi passar o *frame* a ser processado para *grayscale* com a função *cvtColor();* do *OpenCV*. Esta função foi utilizada com três argumentos: imagem a transformar, imagem transformada e *COLOR_BGR2gray*, este último o parâmetro que define a conversão para *grayscale*.



(a) (b)

Figura 4.2: (a) *Frame* 2 (b) *Frame* 4 em *grayscale*

4.4 *Smothing* de Imagens

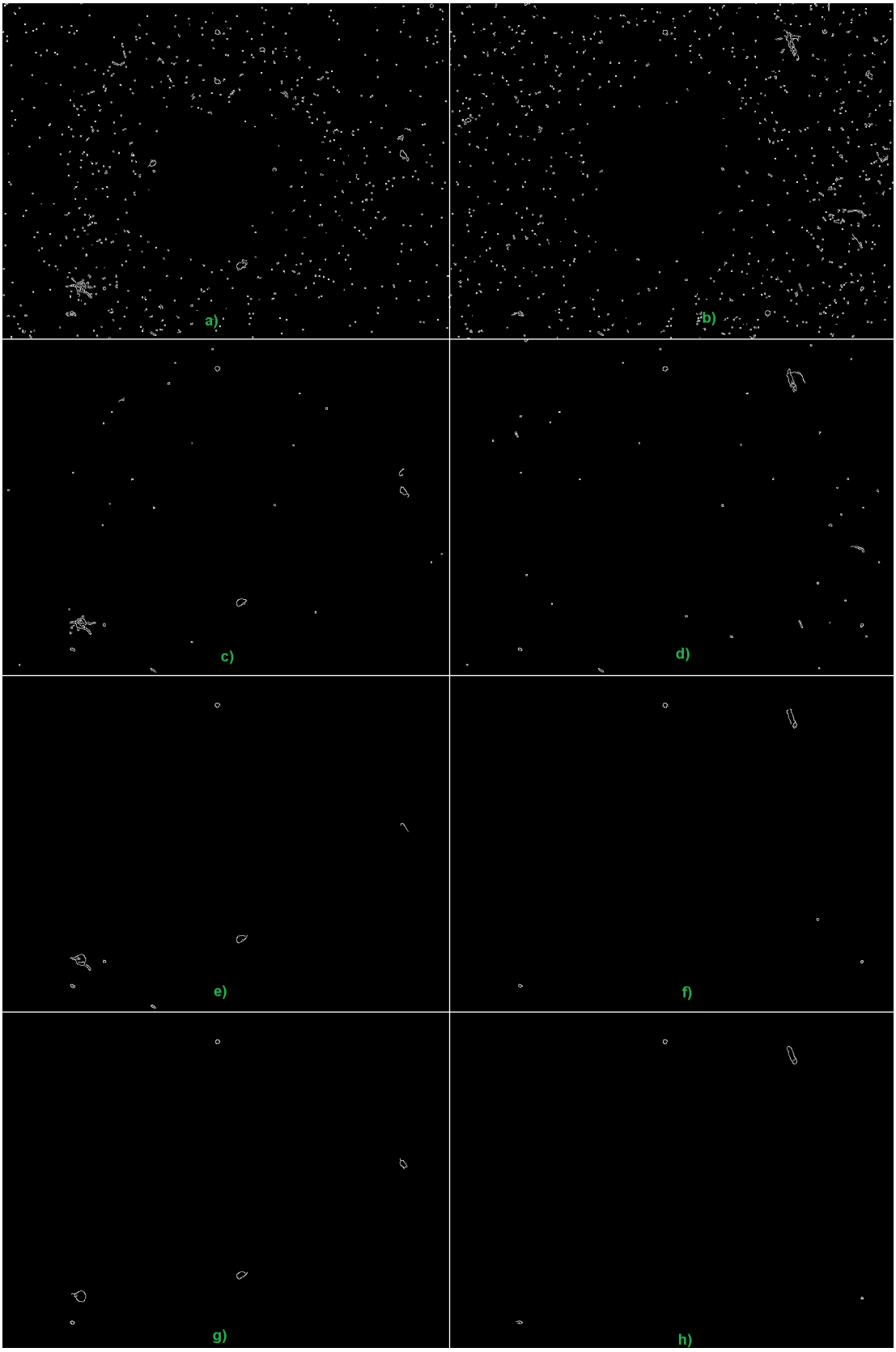
O *smothing* é um passo importante para *edge detection* porque reduz ruído e "alisa" os pixeis da imagem deixando-os menos rectos, o que faz com que as *edges*, que não são mais do que rectas compostas por pixeis, sejam alisadas de forma a descartar possíveis linhas rectas que apenas o são quando analisadas ao nível do pixel[34]. Aplicar esta técnica a uma imagem dá uma sensação de visão mais humana à função seguinte de *edge detection*, no sentido em que, após passar pelo processo, a função passa a ter menos sensibilidade para detectar objectos como detritos, nevoeiros ou outro tipo de partícula de pequena dimensão.

A função utilizada foi o *medianBlur* do *OpenCV* que tem três parâmetros: imagem introduzida, imagem tratada e *kernel size*, o tamanho linear de abertura, que tem de ser um número ímpar. Esta função alisa a imagem usando um filtro de médias(*median filter*) com abertura *kernel size* por *kernel size*. O impacto desta técnica de tratamento de imagem é ilustrada na figura 4.3. O valor escolhido na calibração final do programa foi 5, com o resultado ilustrado em e) e f) pela dimensão dos indivíduos, algo que se perde quando se aumenta o valor para 7. Apesar de ainda serem detectáveis, os resultados não são tão viáveis.

4.5 *Canny Edge Detection*

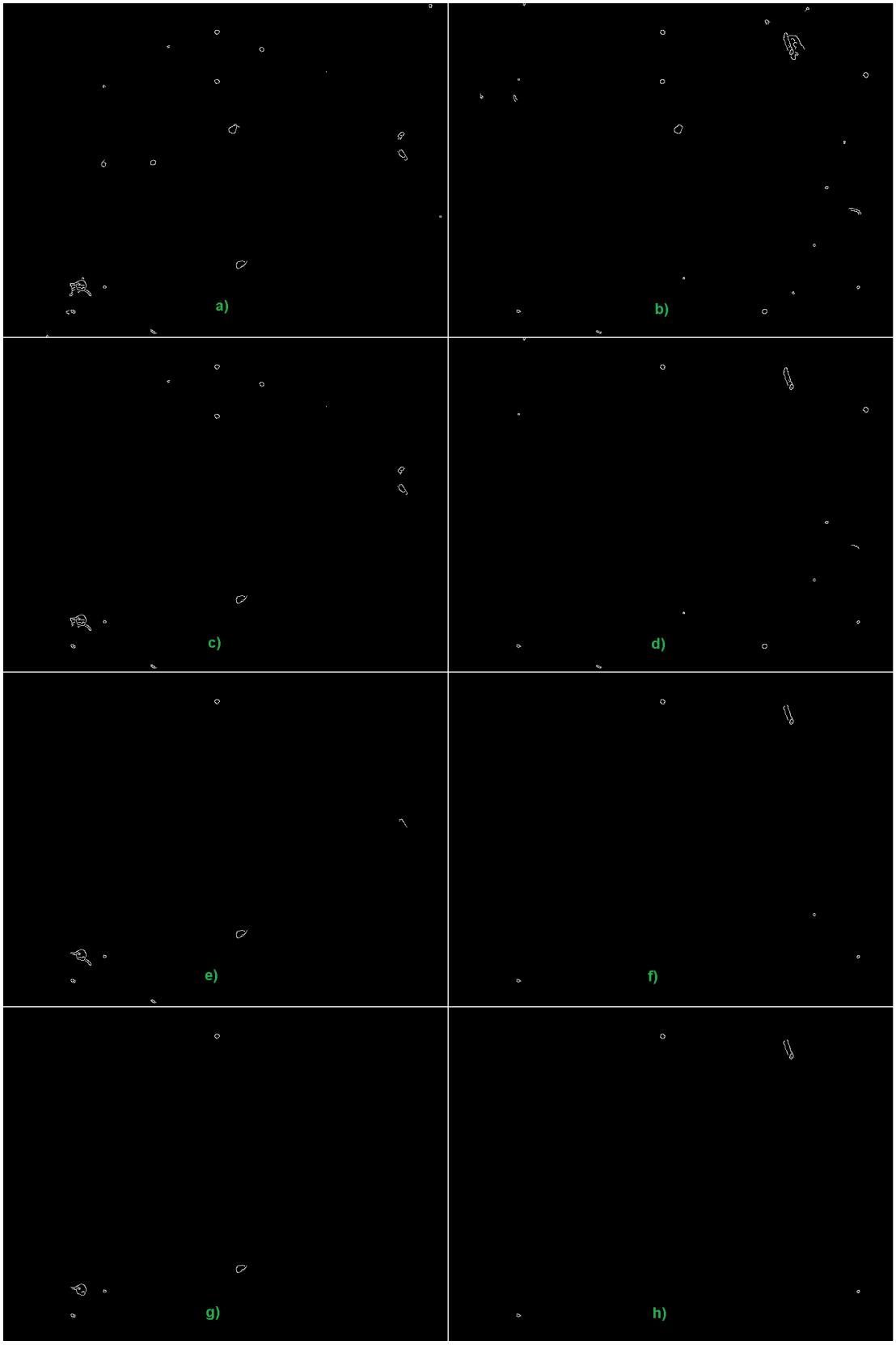
A função utilizada para *canny edge detection* foi o *Canny();* do *OpenCV*. Esta função tem quatro parâmetros por esta ordem: imagem a detectar em *grayscale* e 8 *bits*, imagem com as detecções, primeiro *threshold* para os processos de histerese e o segundo *threshold*

para os processos de histerese [5][35]. O valor de *threshold* ideal resultante de tentativas de calibração foi 57. O resultado de outros valores considerados para as detecções pode ser visto na figura 4.4.



(A) (B)

Figura 4.3: (A) *Frame* 2 (B) *Frame* 4. a) e b) sem *Smothing*, c) e d) *Kernel Size* 3, e) e f) *Kernel Size* 5, g) e h) *Kernel Size* 7. *Threshold=57*



(A) (B)

Figura 4.4: *Canny Edge Detection* nos (A) *Frame* 2 (B) *frame* 4 com diferentes valores de *Threshold*. a) e b) 30, c) e d) 40, e) e f) 57, g) e h) 65. *kernel size* no *smoothing* = 5

4.6 Transformações Morfológicas

Transformações morfológicas são operações baseadas na forma de uma imagem. São operações aplicadas em imagens binárias e utilizam três parâmetros: imagem alvo, imagem transformada e a forma da transformação, o elemento que decide a natureza da operação[36].

As transformações usadas foram dilatação e erosão. A primeira função, *dilate();* do *OpenCV*, dilata os pixeis numa determinada forma com um determinado tamanho específicado no terceiro parâmetro, que no caso foi *getStructuringElement(MORPH _RECT,Size(11,11))*, dilatação dos pixeis em rectângulo 11 x 11. O efeito da transformação nos *frames* 2 e 4 pode ser visto na figura 4.5. A segunda função erode os pixeis mais pequenos que o especificado no terceiro parâmentro da função *erode();*, que no caso foi *getStructuringElement(MORPH _RECT,Size(13,13))*, que elimina elementos menores que 13 x 13. O resultado desta operação é visível na figura 4.6.

Este método faz com que pixeis separados ou em pequenos grupos, normalmente detritos, sejam aumentados ficando com a dimensão de 11 x 11, mas grupos de pixeis maiores, como detecções, grupos de interesse ou apenas objectos grandes, fiquem com os seus pixeis combinados quando dilatados da maneira certa e criem grupos de pixeis maiores que 13 x 13. Passando estes dois exemplos pela função *erode*, os pequenos grupos de pixeis, que passaram a ter dimensões de 11 x 11 são eliminados, mas grupos de pixeis compostos pela combinação de pixeis vizinhos originem grupos de pixeis de 20 x 20, por exemplo, e se mantenham na imagem final, excluindo, assim, objectos sem interesse para a detecção.

4.7 Detecção de Contornos

Nos resultados obtidos no passo anterior é necessário destacar a presença das áreas na imagem, por essa razão, é utilizada a função *findContours();* do *OpenCV*, que encontra contornos em imagens binárias e guarda-as num *array*, de forma a fechar as formas resultantes do passo anterior.

4.8 Detecção de Áreas

Depois a área final estar definida é percorrido o vector e, se algum objecto tiver uma área superior a 80 pixeis quadrados, a função de detecção retorna 1 e a imagem tratada (estado da imagem no final do capítulo 3) é guardada, se não tiver retorna 0 e a imagem seguinte é avaliada. O resultado das detecções em dois *frames* está presente nas figuras 4.7 e 4.8.

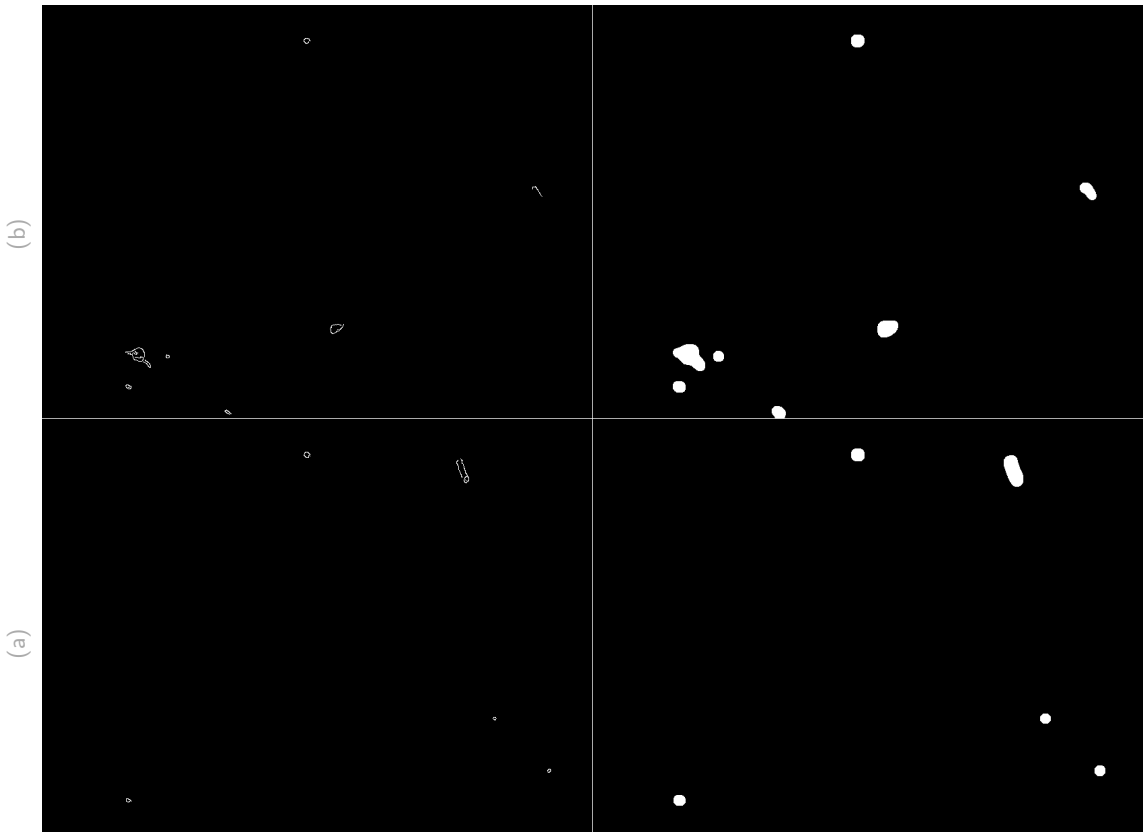


Figura 4.5: (a) *Frame* 2 antes e após aplicar dilatação. (b) *Frame* 4 antes e após aplicar dilatação

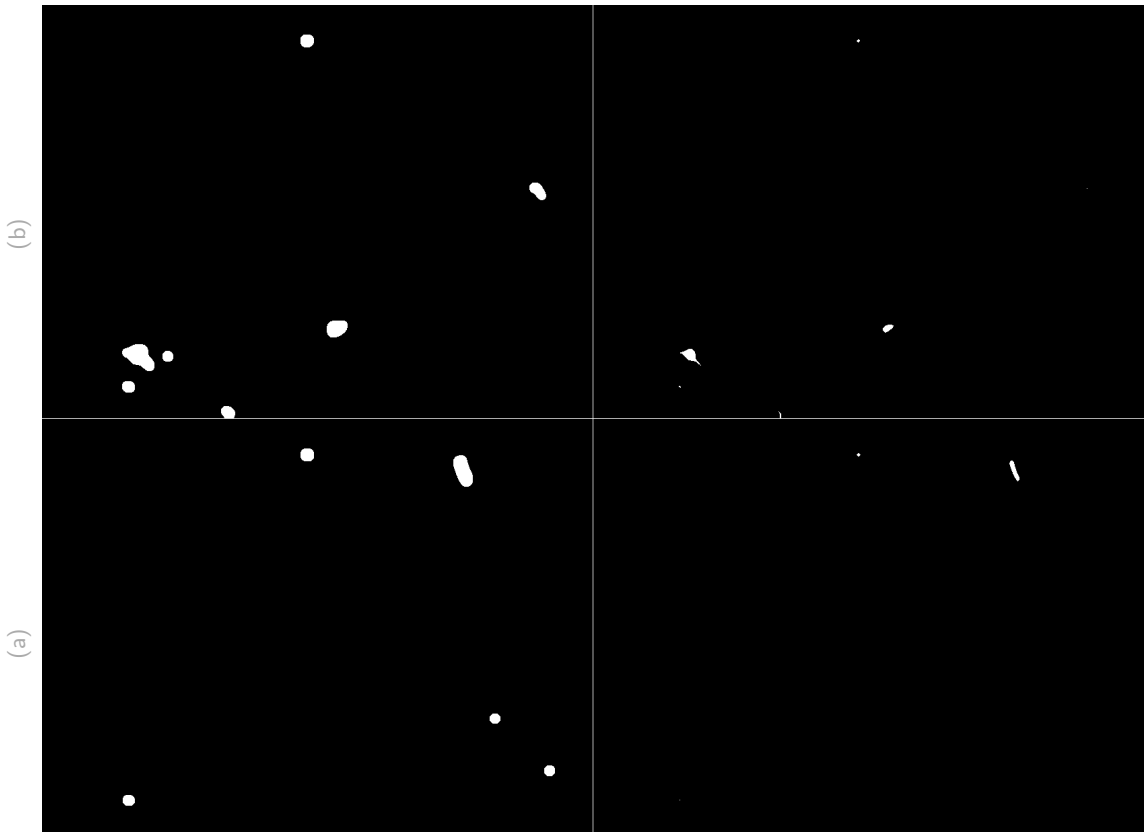
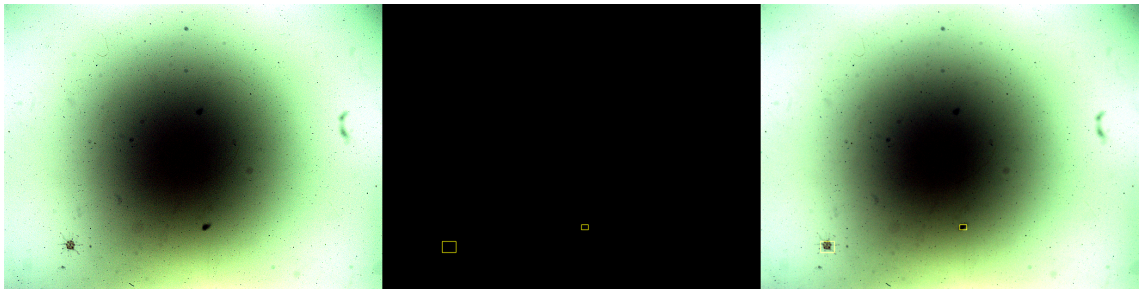
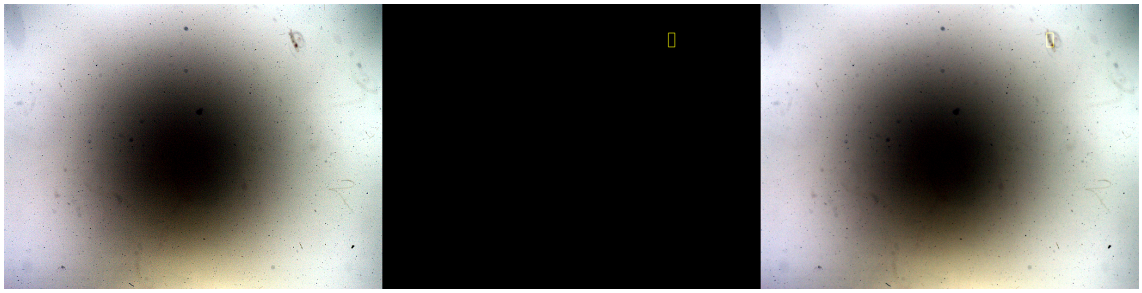


Figura 4.6: (a) *Frame* 2 com dilatação e após aplicar erosão. (b) *Frame* 4 com dilatação e após aplicar erosão



(a) (b) (c)

Figura 4.7: Detecções no *Frame* 2. (a) Imagem melhorada (b) Detecções feitas (c) *Overlay* das detecções



(a) (b) (c)

Figura 4.8: Detecções no *Frame* 4. (a) Imagem melhorada (b) Detecções feitas (c) *Overlay* das detecções

4.9 Resultados

Resumindo, o processo de detecção passa por utilizar a imagem melhorada, passa-la para *grayscale*, ulitizar *smoothing* para facilitar o passo seguinte, *Canny edge detection*, de seguida dilatação e erosão dos pixeis, de forma a eliminar detecções de pequena dimensão, detecção de contornos aplicada aos pixeis restantes e deteção de áreas com dimensões maiores que 80 pixeis quadrados. O resultado deste processo é apresentado nas figuras 4.9 a 4.13, em *frames* com características diferentes, de forma a demonstrar vários cenários em que as imagens são guardadas ou descartadas. De notar que a última imagem de cada figura (canto inferior direito) existe apenas para ilustrar a detecção. Quando a imagem é aceite não progride para a rede neuronal com a caixa que se apresenta nestas detecções, essa caixa é o resultado da detecção aplicado em cima da imagem original.

O *frame* 1 apresenta um objecto grande, fácil de detectar, mas que se pode revelar de pouca importância para o passo seguinte, a rede neuronal, já que se encontra desfocado e difícil de decifrar. O objecto é detectado e assinalado sem qualquer outra detecção.

O *frame* 2 apresenta um indivíduo em perfeitas condições, com um núcleo e patas visíveis, mas, também, alguns objectos de dimensão considerável que podem ser detectados pelo programa. Este *frame* tem informação perfeita para categorizar, por exemplo, com *LabelImg*. Apesar do objecto de importância ter sido detectado, foi também detectado um falso positivo.

O *frame* 3 apresenta um individuo ligeiramente mais pequeno que o *frame* anterior e um objecto aproximadamente da mesma dimensão deste pequeno. Este *frame* foi um dos principais utilizados na calibração do sistema devido ao tamanho do alvo de importância. Como era esperado, a detecção retorna dois corpos, o desejado e o de igual dimensão.

O *frame* 4 apresenta um objecto bem definido, com uma dimensão considerável, mas com alguma transparência, algo que o tornou interessante como imagem de de calibração. O indivíduo aparece assinalado sem qualquer outra forma ter sido detectada.

O *frame 5* é um *frame* sem detecções, apenas detritos, características que tornaram este *frame* importante para definir os parâmetros de dilatação e erosão que tornariam uma imagem vazia em zero detecções.

Os resultados foram bastante positivos, sendo que 88,02% das imagens com detecções têm informação válida para pós-processamento, 9,53% das imagens são falsos positivos e 2,45% são inconclusivos (dados baseados em 451 imagens resultantes de uma amostra de 1334 imagens). Os falsos negativos, detecções falhadas, acontecem em 2,29% das vezes

(dados baseados em 3 imagens que continham objectos de difícil detecção numa amostra de 131 *frames*).

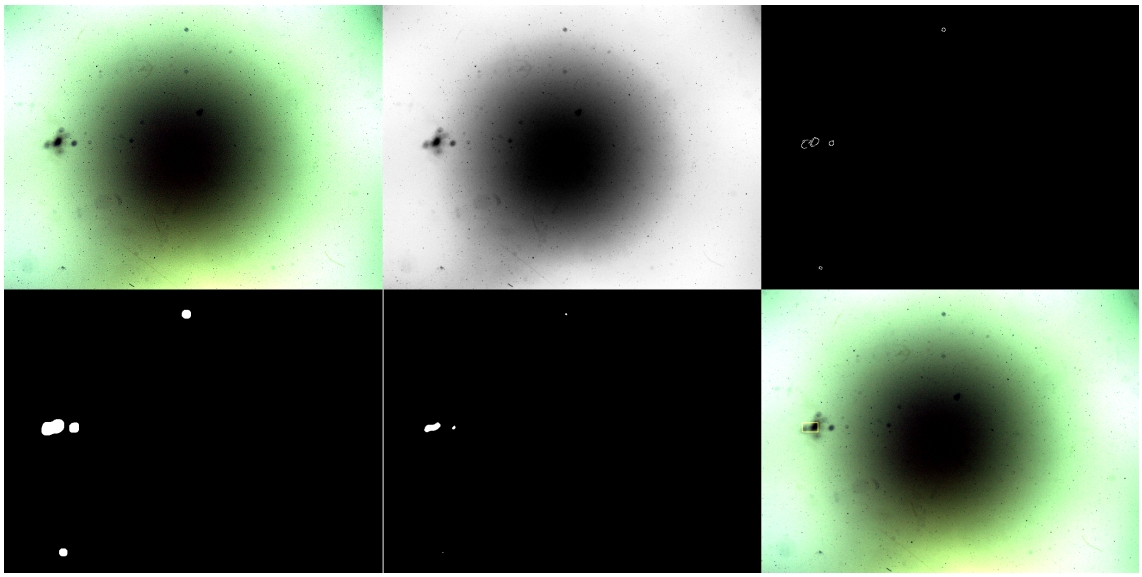


Figura 4.9: Processo total de detecção aplicado ao *Frame* 1. Imagem melhorada, Imagem em *grayscale*, *canny edge detection*, dilatação, erosão e detecção

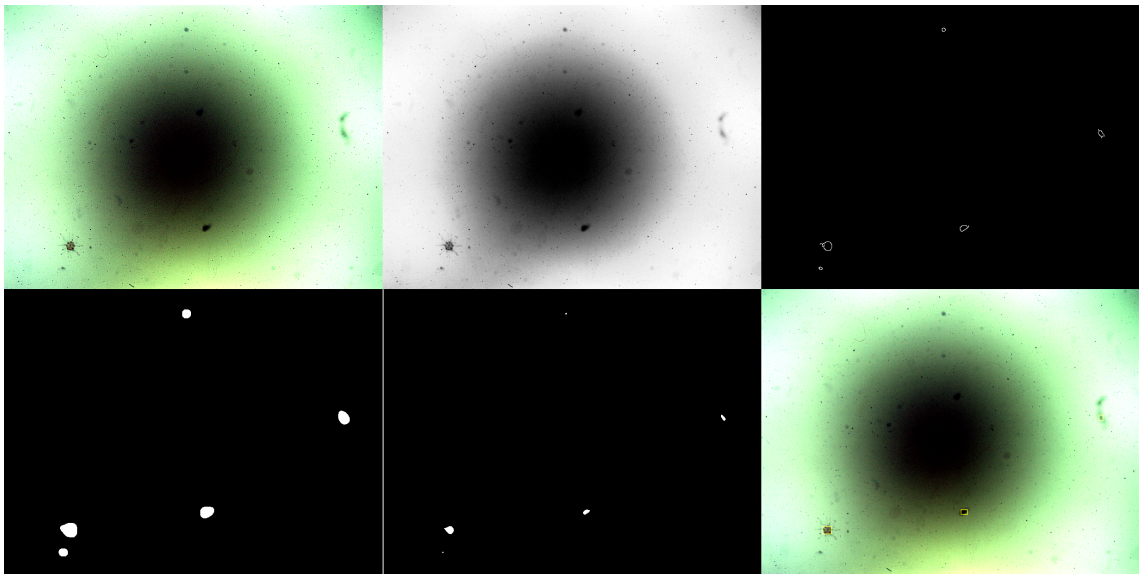


Figura 4.10: Processo total de detecção aplicado ao *Frame* 2. Imagem melhorada, Imagem em *grayscale*, *canny edge detection*, dilatação, erosão e detecção

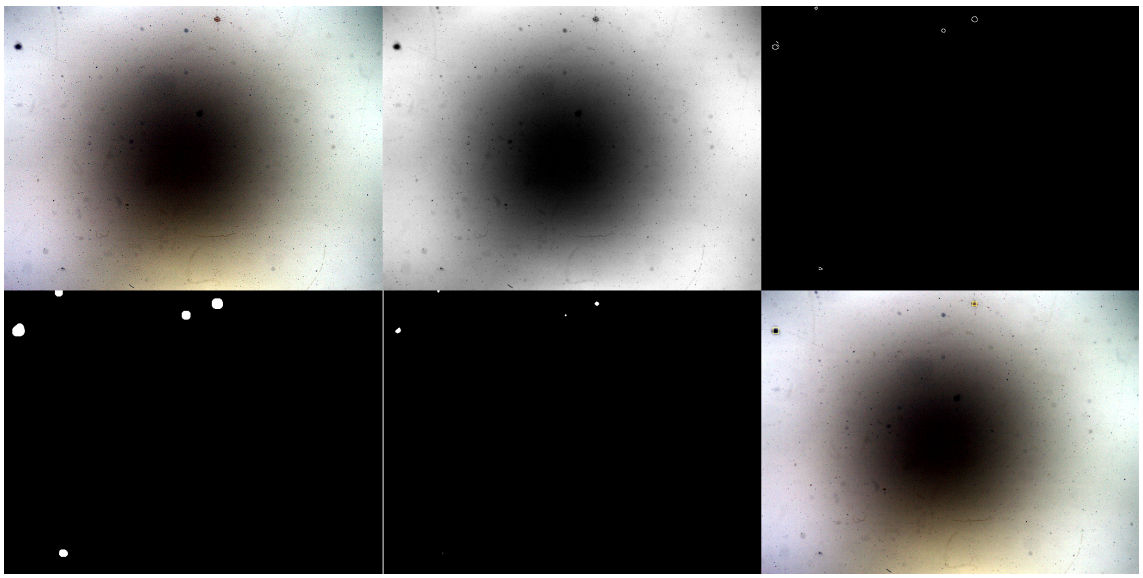


Figura 4.11: Processo total de detecção aplicado ao *Frame* 3. Imagem melhorada, Imagem em *grayscale*, *canny edge detection*, dilatação, erosão e detecção

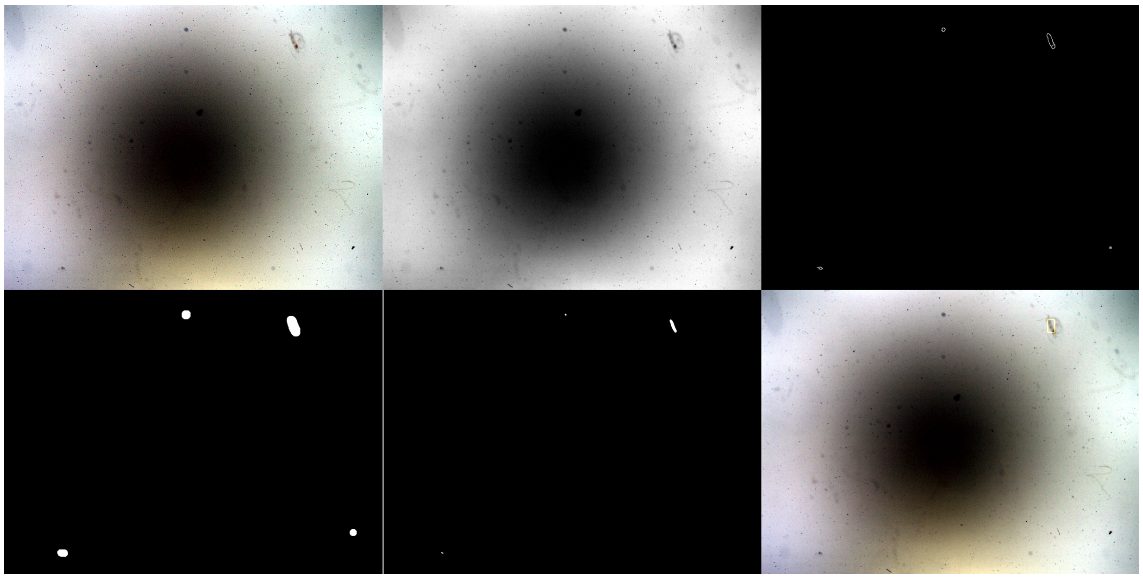


Figura 4.12: Processo total de detecção aplicado ao *Frame* 4. Imagem melhorada, Imagem em *grayscale*, *canny edge detection*, dilatação, erosão e detecção

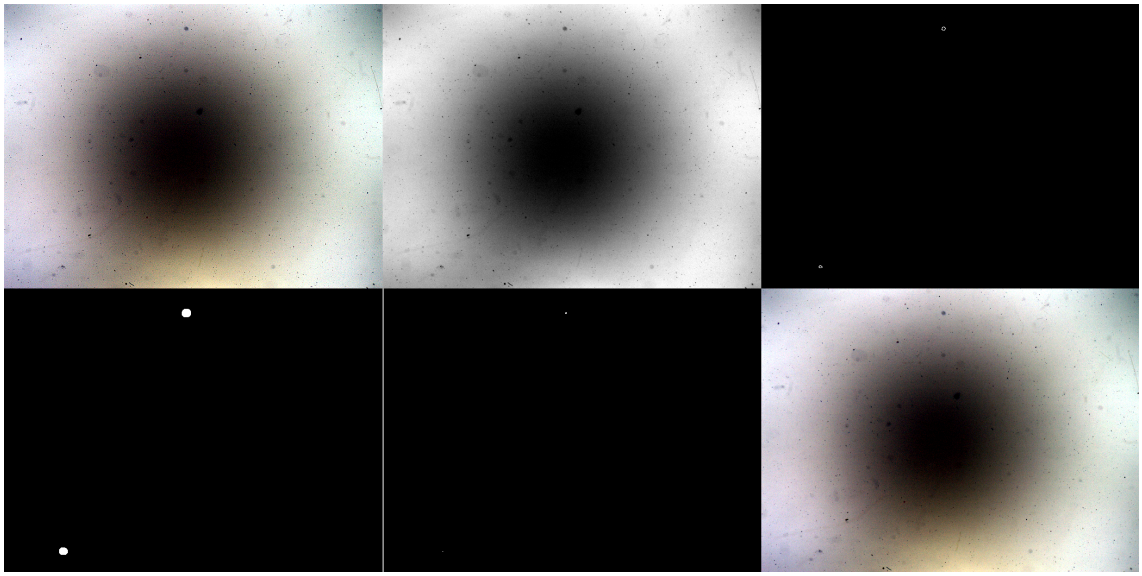


Figura 4.13: Processo total de detecção aplicado ao *Frame* 5. Imagem melhorada, Imagem em *grayscale*, *canny edge detection*, dilatação, erosão e detecção

Esta Página
Foi Intencionalmente
Deixada Em Branco

Capítulo 5

Implementação em tempo real

5.1 Introdução

Até ao momento a solução implementada corria em C++ com a função *imread();* utilizada para ler imagens guardadas numa pasta e a velociadade do processo era pouco mais de 2.2 *frames* por segundo, o que fez com que fosse necessário um último passo para que o programa corresse em tempo real e pudesse integrar missões do projecto *Marineye* de forma eficiente e eficaz.

5.2 Arquitectura do sistema

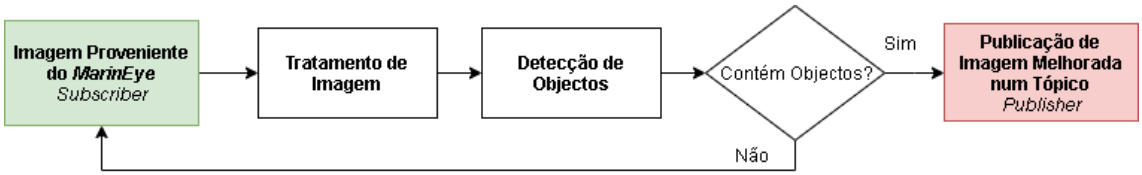


Figura 5.1: Diagrama da função de *callback*

5.3 Implementação

O modelo de trabalho do projecto *MarinEye*, até ao momento consistia em publicar as imagens tiradas em missões e guarda-las em ficheiros *.bag* (um tipo ficheiro que grava mensagens Ros[37]) e ler esses ficheiros após voltar ao laboratório de forma a guardar o resultado numa pasta para a informação ser utilizada.

Para utilizar ROS no programa foi necessário subscrever o tópico publicado pelo *MarinEye*, */camera/image _raw* e utilizar a informação da mensagem de forma a substituir a

função *imread();* do *OpenCV* utilizada até ao momento. Somente este passo diminuiu o tempo de processamento de cada *frame* em 0.3 segundos. O programa teve duas versões testadas para que fosse possível escolher a mais indicada para as missões: uma versão que guarda as imagens numa pasta, outra que as publica num tópico ROS. Nesta segunda versão a grande mudança foi a substituição da função *imwrite();* do *OpenCV* pela publicação dessa imagem no tópico imagem_final.

Foi, também, implementado um interface gráfico de forma a informar o utilizador da percentagem de *frames* gravados até ao momento, número de *frames* gravados e *frames* por segundo a que o programa está a processar imagens, como é possível ver na figura 5.2.

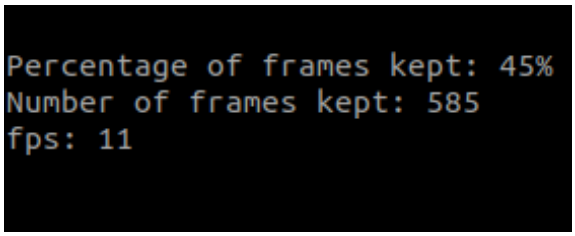


Figura 5.2: Interface gráfica do programa (primeira versão - Subscreve um tópico, guarda resultados directamente na memória)

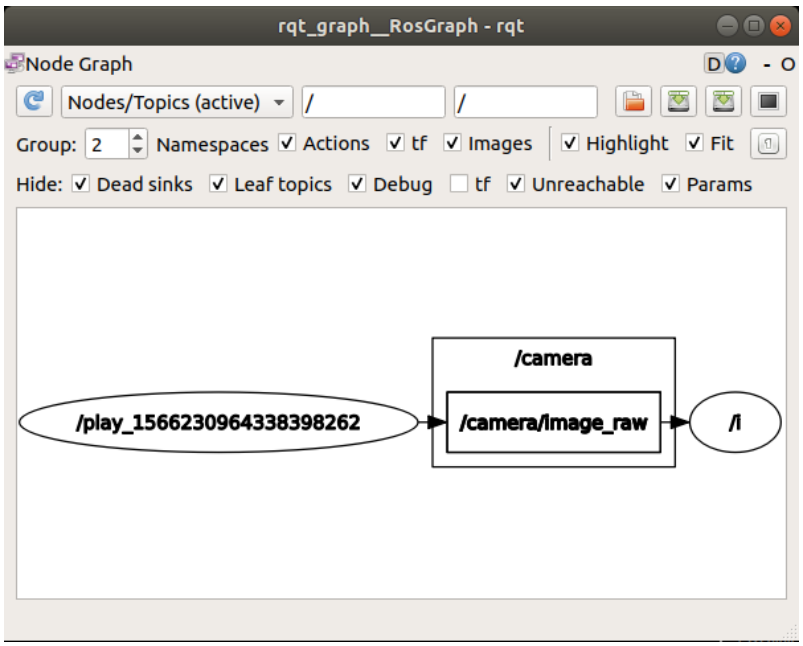


Figura 5.3: *Output* do *rqt_graph* (primeira versão - subscreve um tópico, guarda resultados directamente na memória)

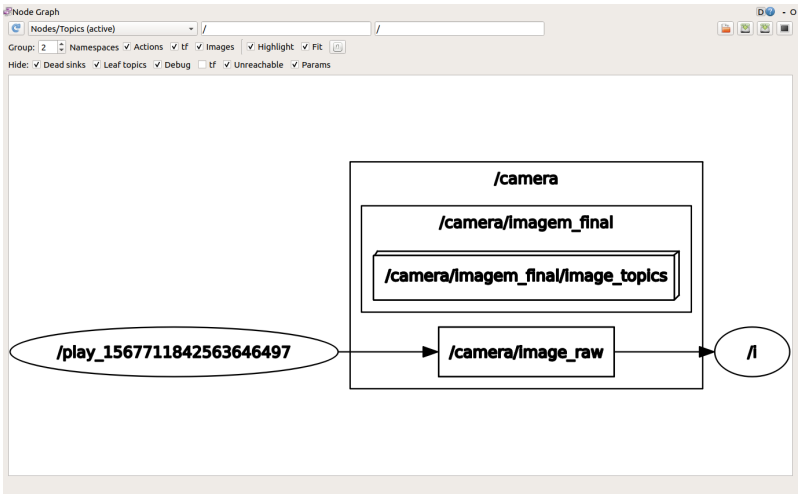


Figura 5.4: *Output* do *rqt_graph* (segunda versão - subscreve um tópico, publica os resultados no tópico imagem_final)

5.4 Resultados

A primeira versão mostrou imediatametne resultados positivos já que a velocidade de processamento aumentou substancialmente, passando dos 2.2 fps para um processamento a 11 fps que desce para cerca de 8 quando uma imagem está a ser guardada numa pasta. Tendo em conta que o sistema do *MarinEye* captura imagens a 5 fps e a rede neuronal avalia imagens a 7 fps, a velocidade atingida após incluir as funcionalidades do ROS é suficiente para que o programa corra em tempo real.

A segunda versão teve ainda melhores resultados, já que foi removido o passo que baixava a velocidade de processamento para 8 fps, e o programa passou a correr a 12 *frames* por segundo, chegando, por vezes, aos 13. A publicação da imagem no tópico pode ser vista na figura 5.4, através do comando *rqt_graph* e a diferença entre as duas versões pode ser observada comparando as figuras 5.3 e 5.4 e a diferença de performance pode ser vista nas figuras 5.5 e 5.6. Foram ainda desenvolvidos dois modos de funcionamento: um modo manual onde o utilizador pode alterar dois parâmetros e um modo automático, que se adapta ao cenário em que está inserido (ver anexo B).

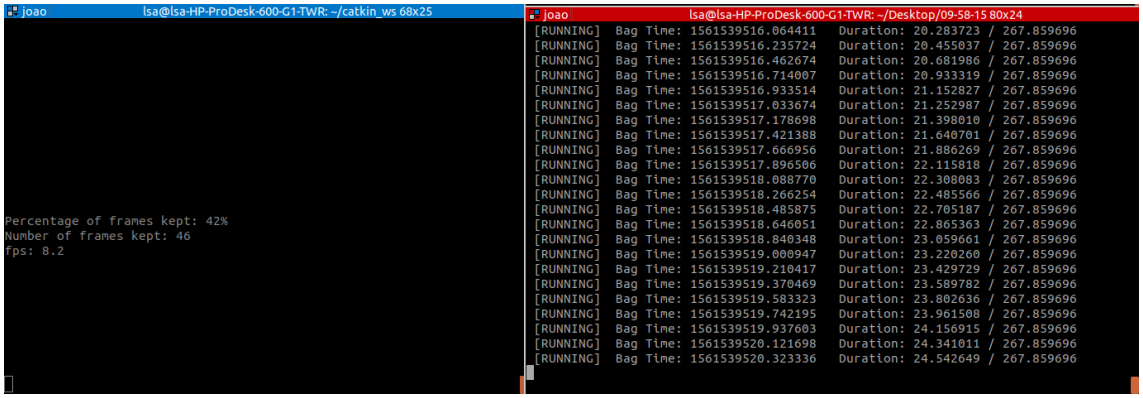


Figura 5.5: Lado esquerdo - *Output* ROS da primeira versão quando um *frame* é guardado, lado direito - enquanto ficheiro *.bag* é Lido

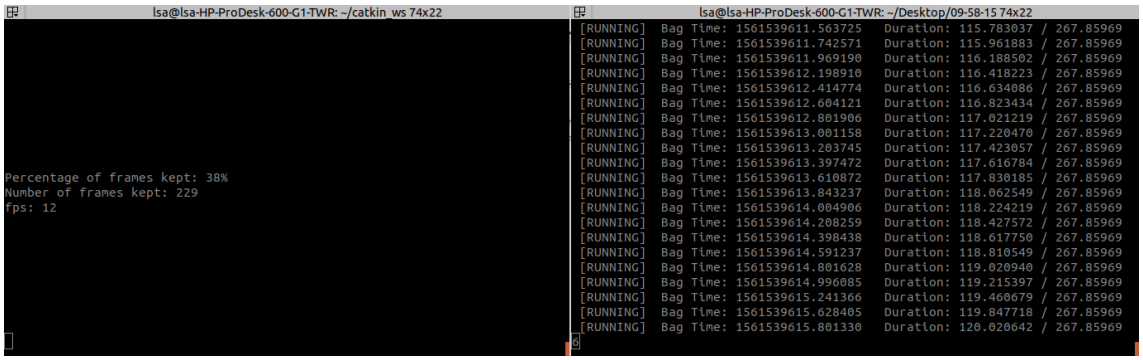


Figura 5.6: Lado esquerdo - *Output* ROS da segunda versão, lado direito - enquanto ficheiro *.bag* é Lido

Capítulo 6

Testes no terreno

6.1 Introdução

O problema de luz central apresentado até aos testes referidos nos capítulos anteriores foi corrigido e o sistema foi levado para o terreno para comprovar a eficácia do programa desenvolvido.

6.2 Tanque LSA (24/09/2019)

O primeiro teste em tempo real foi realizado no tanque do LSA de forma a validar o estado da iluminação e a capacidade do sistema desenvolvido de ignorar lixo e detectar apenas objectos com potêncial interesse. O sistema de captação de imagens(figura 6.1 e 6.2) capturou as mesmas a 5 *frames* por segundo e o programa desenvolvido, em tempo real, processou imagens a 12 *frames* por segundo, concluído-se, assim, que o sistema consegue acompanhar as missões do projecto *MarinEye* e tem processamento suficiente para ser capaz de acompanhar melhorias feitas ao sistema desde que se cinjam aos 12 *frames* por segundo, mais do dobro da taxa de aquisição actual. Foram necessários alguns ajustes de iluminação, mas os testes foram um sucesso no sentido de testar o sistema em tempo real. Os resultados deste teste estão presentes na figura 6.3.

6.3 Tanque LSA (25/09/2019)

Realizados os ajustes necessários à iluminação(que se concluíram ser exteriores ao sistema de captação de imagens), foi realizado um novo teste de modo a revalidar a eficácia já provada do sistema desenvolvido e ainda a qualidade final das imagens capturadas. Devido ao facto de não haver objectos de interesse no tanque do LSA, este teste foi útil no sentido de calibrar o programa de modo a descartar lixo que pudesse ser detectado como falsos positivos. Os ajustes e resultados deste teste podem ser vistos na figura 6.4.

Figura 6.1: Sistema de captação de imagens

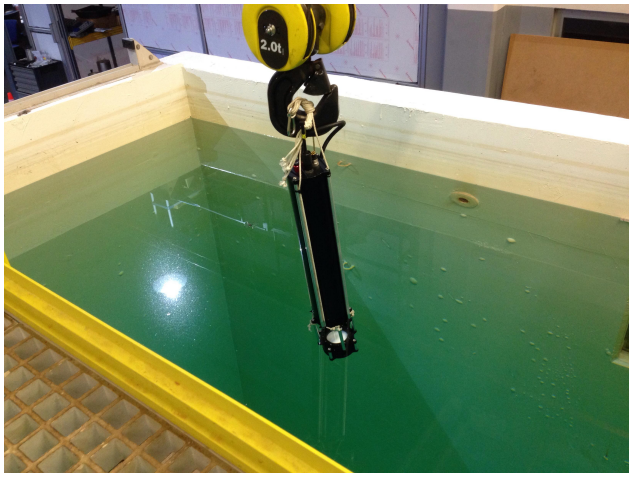


Figura 6.2: Captação de imagens no tanque do LSA

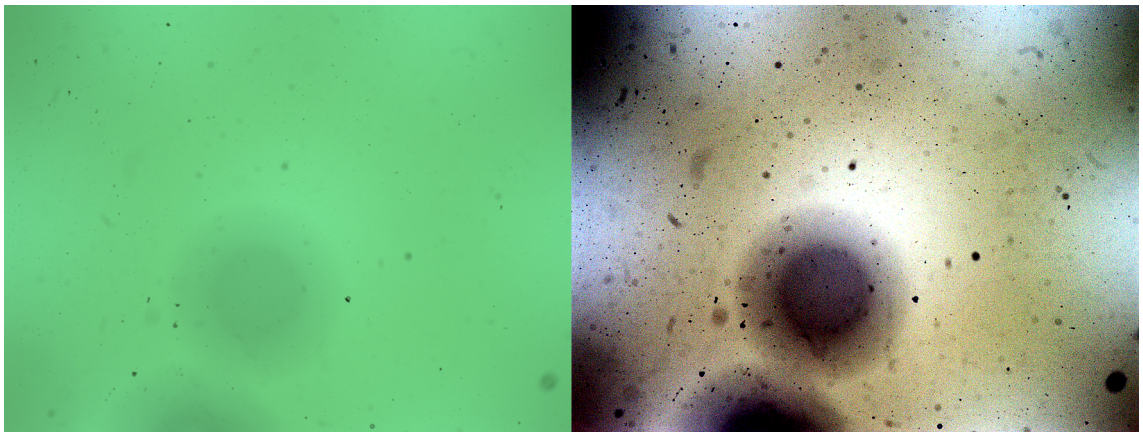


Figura 6.3: Captação feita no tanque do LSA e imagem melhorada

6.4 Valbom (26/09/2019)

Validada a eficácia do programa num ambiente semi-controlado (tanque do LSA), o sistema de captação e o sistema desenvolvido foram levados para uma plataforma flutuante na margem norte do rio Douro(figuras 6.5 e 6.6). Tal como verificado nos testes realizados no laboratório, com a mesma taxa de aquisição pelo sistema de captação, o sistema desenvolvido manteve-se a 12 *frames* por segundo. Ao contrário do ambiente do tanque, o ambiente de rio apresentou uma água muito mais turva e com difícil exclusão de falsos positivos. Devido a estas dificuldades, os parâmetros até ao momento fixos tiveram de ser alterados e passou a haver uma restrição muito maior na aceitação de uma imagem pelo

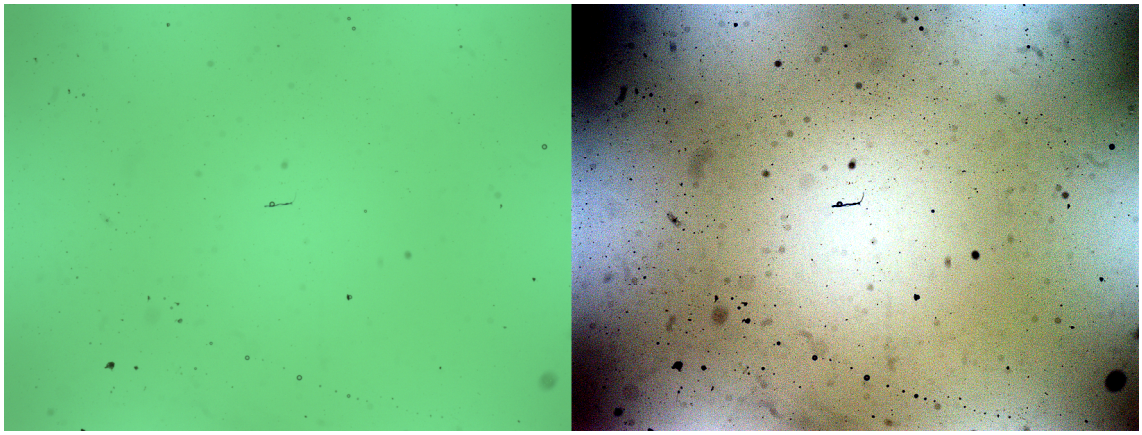


Figura 6.4: Captação feita no tanque do LSA com luz corrigida e imagem melhorada

sistema, nomeadamente no tamanho do que é considerado um objecto de interesse.

Um dos pontos a apontar neste cenário é o facto da ondulação afectar a captação. Aproximadamente $\frac{1}{6}$ das capturas feitas apresentam-se desfocadas, tremidas ou com objectos deformados devido ao movimento do cilindro causado pela agitação da água. Apesar de ser muito pouca, cada vez que um barco passava, mesmo que a vinte metros do local onde a captura estava a ser efectuada, as ondas que chegavam minutos depois faziam com que o cilindro ficasse instável por momentos. Este problema seria resolvido se os testes fossem feitos no mar para lá da ondulação. A missão foi um sucesso em termos de detecção de indivíduos de interesse, tal como mostram as figuras 6.7 e 6.8, e o sistema continuou a responder muito positivamente ao manter a taxa de 12 *frames* por segundo anteriormente atingidos com uma duração de missão de mais de duas horas.

Figura 6.5: Sistema de captação de imagens a ser colocado na água

Figura 6.6: Sistema de captação em acção

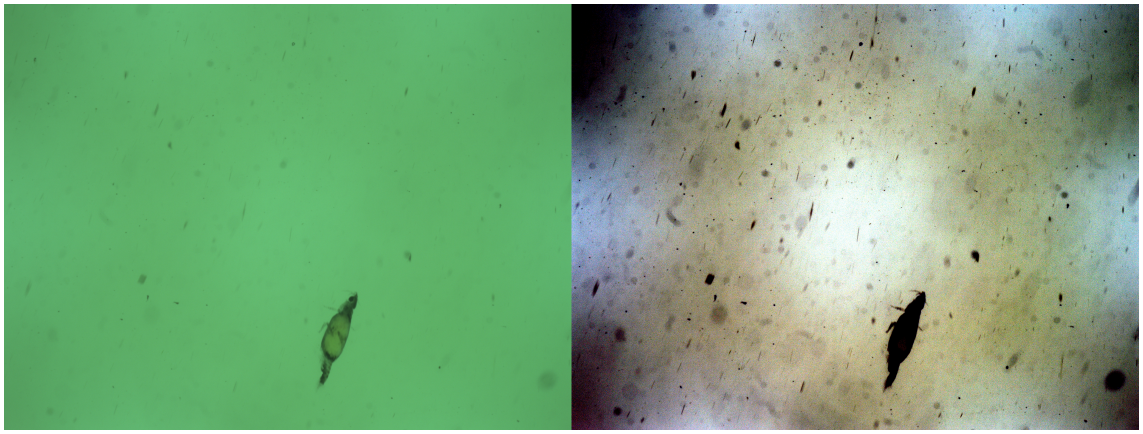


Figura 6.7: Detecção feita no rio Douro durante a missão a Valbom

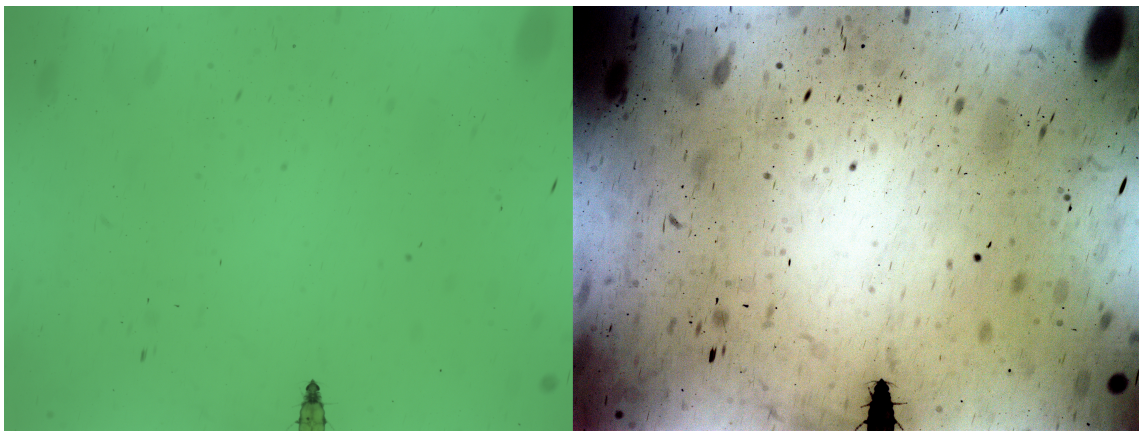


Figura 6.8: Detecção feita no rio Douro durante a missão a Valbom

Esta Página
Foi Intencionalmente
Deixada Em Branco

Capítulo 7

Conclusão e Trabalho Futuro

7.1 Conclusão

O objectivo desta tese foi criar um sistema que aumentasse a quantidade de informação viável recolhida pela câmara de plâncton do *MarinEye* em missões de captura de imagens, descartando informação irrelevante. O produto final foi resultante de experimentação com processamento de imagem que resultou numa cadeia de processos que tornou possível obter o resultado desejado.

Foi desenvolvido com sucesso um processo de melhoramento de imagens subaquáticas que facilitou a observação de imagens subaquáticas, fez com que fosse possível identificar mais facilmente os objectos de estudo e fez com que algoritmos de segmentação como *canny edge detection* identificassem mais facilmente pontos de interesse contribuído, assim, para facilitar o processo de isolar objectos em imagens.

Foi implementada uma sucessão de algoritmos que foi bem sucedida na detecção de objectos que fez com que fosse possível descartar imagens não relevantes, um dos grandes objectivos desta tese.

O sistema desenvolvido foi implementado em ROS e passou a processar imagens em tempo real. Este passo fez com que o programa quintuplicasse a sua velocidade de processamento, mantendo a eficácia antes demonstrada. Com os testes subsequentes e a missão realizados ficou demonstrado que no terreno a eficácia do sistema que se mantém capaz de processar imagens a 12 *frames* por segundo.

Foi desenvolvida uma ferramenta autónoma de calibração de modo a tornar o processo de aquisição e selecção de imagens com potenciais objectos de interesse completamente autónoma.

7.2 Trabalho Futuro

De futuro, algo desejado para o sistema desenvolvido é a implementação de uma ferramenta de calibração que se baseie em estatísticas de missões passadas e estatísticas da missão actual de forma a tornar o sistema desenvolvido completamente autónomo. Com esta ferramenta seria possível guardar conjuntos de características de missões com sucesso e descartar conjuntos de missões com pouca informação recolhida de modo a ter um *preset* automaticamente definido no início de cada nova missão.

Anexo A

ROS

O ROS (*Robot Operating System*) é uma biblioteca de *frameworks* e *software* criada para uniformizar a maneira de comunicar em rotbótica [37]. É *open-source*, baseado em mensagens (*message-based*) e em ferramentas (*tool-based*). O ROS utiliza um sistema de bibliotecas desenhadas para serem usadas independentemente que comunicam por sistemas de mensagens guardadas em nós (*nodes*). A informação contida nos nós pode ser subscrita, caso se queira aceder à informação contida no nó, ou publicada, caso se queira guardar informação para ser subscrita por outros nós de Ros, resultando numa teia de nós que pode ser consultada através do comando *rqt_graph*, como ilustra a figura A.1.

As mensagens estão organizadas por tópicos que podem ser acedidos com o comando *rostopic* que dá ao utilizador acesso à informação a ser processada no momento, por exemplo, a informação que está a ser lida por um programa que subscreveu aquele nó em específico (ex: *rostopic echo /camera/image _raw*), a que frequência aparecem novas mensagens e os nomes dos tópicos acessíveis no momento(*rostopic list*), como se pode ver na figura A.2.

As mensagem são passadas por comunicação *peer to peer* e não têm de ser escritas em nenhuma linguagem de programação específica, já que os nós podem ser escrito em C, C++, *python*, LISP, *Octave* ou qualquer linguagem desde que haja um *wraper* para ROS escrito por alguém para essa linguagem. O ROS corre apenas em *linux* e a filosofia do ROS é inspirada na do *Unix*: Ferramentas desenhadas para comunicar entre elas num ambiente aberto a qualquer programador.

As bibliotecas e os nós de ROS estão organizados em pacotes que contêm bibliotecas, nós e definições das mensagens que o pacote pode usar. Cada pacote tem de ter funcionalidades suficientes para ser útil, mas não demasiadas, pois o tamanho dos pacotes ter de se manter dentro de um tamanho razoável. Existem também *stacks*, cujo melhor exemplo é o *ros_core*, que contém a infraestrutura básica do ROS e tem de ser chamado antes de qualquer processo feito em ROS para que o programa desejado corra[38].

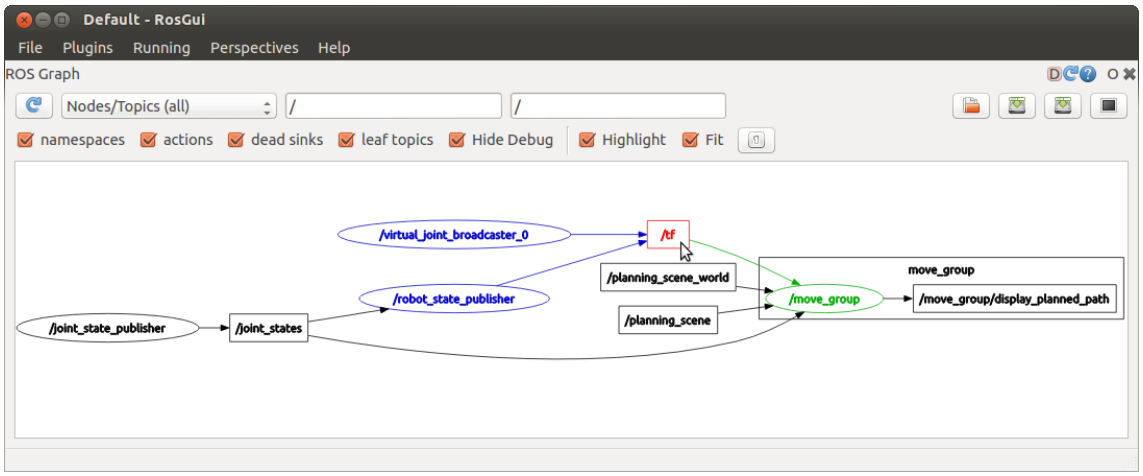


Figura A.1: Exemplo do Comando *rqt_graph* [37]

```
lsa@lsa-HP-ProDesk-600-G1-TWR:~/Desktop/09-58-15$ rostopic list
/camera/image_raw
/clock
/rosout
/rosout_agg
lsa@lsa-HP-ProDesk-600-G1-TWR:~/Desktop/09-58-15$
```

Figura A.2: Exemplo do Comando *rostopic list*

Anexo B

Modos de funcionamento

B.1 Ferramenta desenvolvida para alteração de parâmetros

Devido às dificuldades que os cenário dos pontos 6.2 a 6.4 apresentaram, foi necessário desenvolver uma ferramenta gráfica para que fosse possível ser alterar dois parâmetros chave de detecção no início de cada missão de forma a munir o sistema de alguma flexibilidade perante diferentes cenários. Os parâmentros foram os *threshold* utilizado no *canny edge detection* e o *kernel size* do *median blur* utilizado no mesmo procedimento. Desta forma o utilizador pode calibrar o programa de forma a eliminar detritos que resultariam facilmente em falsos positivos. A ferramenta apresenta a figura original em *grayscale* e o resultado de um *canny edge detection*(figura 6.9). Após fechar a janela, os valores de *threshold* e *kernel size* são guardados e o programa inicia a detecção e publicação das detecções num tópico *ROS*.

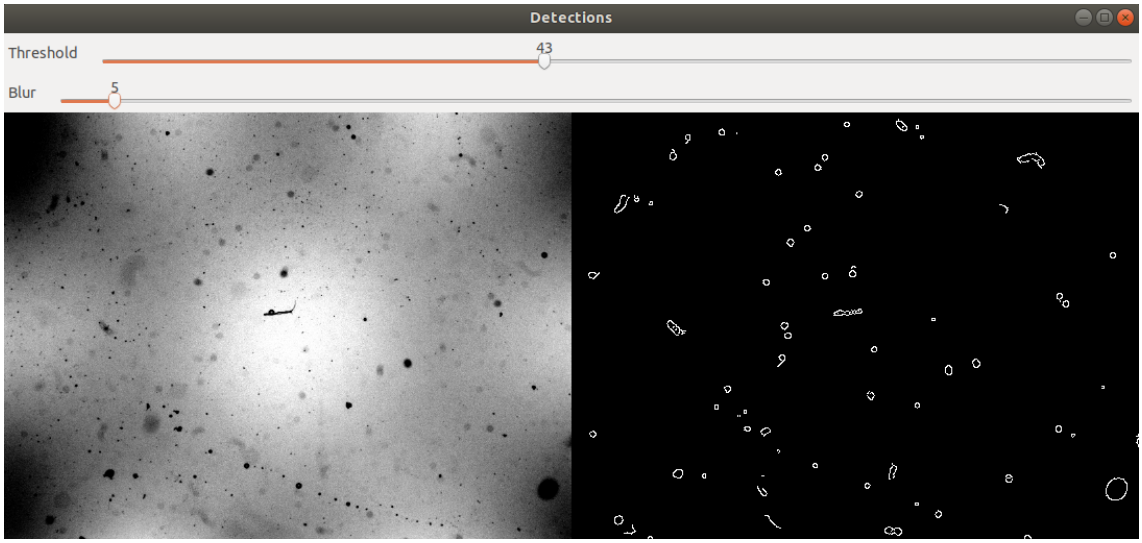


Figura B.1: Ferramenta de calibração

B.2 Adaptação autónoma

Para que o sistema tivesse um modo automático foi implementado um algoritmo de adaptação. Este algoritmo procura manter as detecções numa banda de aceitação. Se as detecções forem entre cem e noventa por cento das imagens capturadas, o algoritmo reduz os parâmetros de interesse de forma diminuir os objectos de interesse apresentados (por experiência adquirida nos testes realizados, se cem a noventa por cento dos *frames* forem detecções, o sistema está a detectar algo comum às imagens (provavelmente, lixo) e não objectos de interesse), se as detecções forem muito perto de zero os parâmetros são alterados para considerar mais objectos como objectos de interesse. Esta técnica calibra o sistema após alguns minutos, mas os testes realizados mostram que a percentagem obtida depois de calibrado o sistema é entre vinte e cinquenta por cento. Com os resultados obtidos com a biblioteca original utilizada para desenvolver o programa (imagens que continham um círculo no centro) a percentagem de objectos foi entre trinta e sete e quarenta e dois por cento.

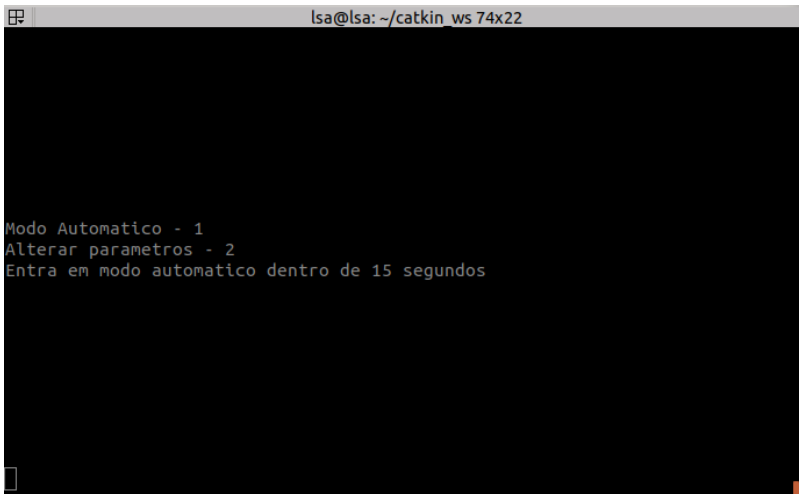


Figura B.2: Menu inicial do programa desenvolvido

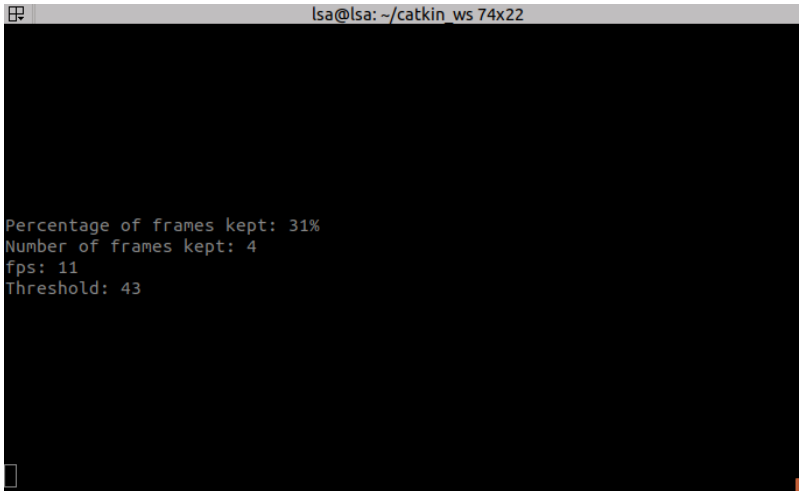


Figura B.3: Interface gráfica com adaptação autónoma

Esta Página
Foi Intencionalmente
Deixada Em Branco

Referências

[1] *Technology Readiness Level*. https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html. Accessed: 09/10/2019.

[2] *MarineEye - Watching the Ocean*. http://marineye.ciimar.up.pt/. Accessed: 19/08/2019.

[3] Rafael C. González e Richard Eugene Woods. *Digital Image Processing*. 2007.

[4] Chongyi Li Saeed Anwar e Fatih Porikli. "Deep Underwater Image Enhancement". Em: *Cornell University - Computer Vision and Pattern Recognition* (2018).

[5] *OpenCV: Canny Edge Detection*. https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html. Accessed: 16/08/2019.

[6] Ghassan Mahmoud husien Amer e Dr. Ahmed Mohamed Abushaala. "Edge Detection Methods". Em: *IEEE* (2015).

[7] Zhao Xu Xu Baojie e Wu Guoxin. "Canny edge detection based on Open CV". Em: *IEEE* (2017).

[8] David G. Lowe. "Object Recognition from Local Scale-Invariant Features". Em: *International Conference on Computer Vision* (1999).

[9] *Introduction to SURF (Speeded-Up Robust Features)*. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html. Accessed: 16/08/2019.

[10] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Em: *International Journal of Computer Vision* (2004).

[11] *Introduction to SIFT (Scale-Invariant Feature Transform)*. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html. Accessed: 09/10/2019.

[12] Tinne Tuytelaars Herbert Bay Andreas Ess e Luc Van Gool. "Speeded-Up Robust Features (SURF)". Em: *Elsevier* (2008).

[13] Barry Brumitt Kentaro Toyama John Krumm e Brian Meyers. "Wallflower: Principles and Practice of Background Maintenance". Em: *IEEE: Seventh IEEE International Conference on Computer Vision* (1999).

[14] *OpenCV: How to Use Background Subtraction Methods*. `https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html`. Accessed: 16/08/2019.

[15] Carsten Rother e Andrew Blake Vladimir Kolmogorov. “”GrabCut- Interactive Foreground Extraction using Iterated Graph Cuts”. Em: *Microsoft Research Cambridge, UK* (2004).

[16] Junseong Bang Jinsu Lee e Seong-Il Yang. “Object Detection With Sliding Window in Images Including Multiple Similar Objects”. Em: *IEEE* (2017).

[17] Sander Dieleman. “Classifying plankton with deep neural networks”. Em: *National Data Science Bowl* (2015).

[18] *labelImg*. `https://github.com/tzutalin/labelImg`. Accessed: 16/08/2019.

[19] *The PASCAL Visual Object Classes Homepage*. `http://host.robots.ox.ac.uk/pascal/VOC/`. Accessed: 13/11/2019.

[20] *ImageNet*. `http://image-net.org/`. Accessed: 16/08/2019.

[21] Tien-Ju Yang Vivienne Sze Yu-Hsin Chen e Joel S. Emer. “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”. Em: *Proceedings of the IEEE* (2017).

[22] Ross Girshick Joseph Redmon Santosh Divvala e Ali Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. Em: ().

[23] Fatih Ertam e Galip Aydin. “Data Classification with Deep Learning using Tensorflow”. Em: *IEEE, 2nd International Conference on Computer Science and Engineering* (2017).

[24] Jianmin Chen Martín Abadi Paul Barham e Zhifeng Chen. “TensorFlow: A system for large-scale machine learning”. Em: *12th USENIX Symposium on Operating Systems Design and Implementation* (2016).

[25] Ahmet ÇÕnar Emine Cengil e Erdal Özbay. “Image Classification with Caffe Deep Learning Framework”. Em: *IEEE, 2nd International Conference on Computer Science and Engineering* (2017).

[26] Dario Lodi Rizzini Fabio Oleari Fabjan Kallasi e Jacopo Aleotti. “An Underwater Stereo Vision System: from Design to Deployment and Dataset Acquisition”. Em: *OCEANS 2015 - Genova* (2015).

[27] Claris Jose Alex Raj e Supriya M. H. “Hardware realization of canny edge detection algorithm for underwater image segmentation using field programmable gate arrays”. Em: *Journal of Engineering Science and Technology* (2017).

[28] Wenwei Xu e Shari Matzner. “Underwater Fish Detection using Deep Learning for Water Power Applications”. Em: *IEEE* (2018).

[29] Yang Bu Zhe Chen Zhen Zhang e Fengzhao Dai. "Underwater Object Segmentation Based on Optical Features". Em: *Sensors 2018* (2018).

[30] Christophe De Vleeschouwer Codruta O. Ancuti Cosmin Ancuti e Philippe Bekaert. "Color Balance and Fusion for Underwater Image Enhancement". Em: *IEEE Transactions on Image Processing* (2018).

[31] Eduardo Silva Alfredo Martins André Dias. "MarinEye – A tool for marine monitoring". Em: *OCEANS 2016 - Shanghai* (2016).

[32] A. Dias e A. Martins Joao Resende Pedro Barbosa. "Autonomous High-Resolution Image Acquisition System for Zooplankton". Em: *Oceans 2019 - Seattle* (2019).

[33] A. Martins Pedro Geraldes Joel Barbosa. "In situ real-time Zooplankton Detection and Classification". Em: *Oceans 2019 - Marseille* (2019).

[34] *Smoothing Images*. `https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html`. Accessed: 20/08/2019.

[35] *Canny Edge Detector*. `https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`. Accessed: 20/08/2019.

[36] *Morphological Transformations*. `https://docs.opencv.org/3.3.0/d9/d61/tutorial_py_morphological_ops.html`. Accessed: 22/08/2019.

[37] *ROS.org*. `http://wiki.ros.org/`. Accessed: 16/08/2019.

[38] Steve Cousins. "Welcome to ROS Topics". Em: *IEEE Robotics & Automation Magazine* (2010).