

Simulação da navegação de um AUV baseada na deteção de Ghost Nets com recurso a Deep Learning

CARLOS MIGUEL MENDONÇA E SILVA
outubro de 2024

POLITÉCNICO DO PORTO
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Simulação da navegação de um AUV
baseada na deteção de *Ghost Nets*
com recurso a *Deep Learning*

Carlos Miguel Mendonça e Silva

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Sistemas Autónomos

ISEP INSTITUTO SUPERIOR
DE ENGENHARIA DO PORTO

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto

Outubro, 2024

*Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de
Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em
Engenharia Electrotécnica e de Computadores, Área de Especialização em
Sistemas Autónomos.*

Candidato: Carlos Miguel Mendonça e Silva, N.º 1161127,
1161127@isep.ipp.pt

Orientação Científica: Eduardo Pereira da Silva, eps@isep.ipp.pt

Coorientação Científica: Alfredo Oliveira Martins, aom@isep.ipp.pt

ISEP INSTITUTO SUPERIOR
DE ENGENHARIA DO PORTO

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Outubro, 2024

Agradecimentos

Esta dissertação marca o final de uma etapa que contou com o apoio de várias pessoas, sem as quais não teria sido possível alcançar.

Em primeiro lugar, gostaria de agradecer ao Professor Alfredo Martins e ao Professor Eduardo Silva pela oportunidade em desenvolver a tese na área da robótica submarina, algo que sempre despoletou o meu interesse.

Aos meus colegas de curso, Luís e Pedro, com quem passei tantas horas no LSA a resolver problemas, criar projetos e a aprender. Teria sido difícil passar do primeiro ano sem eles.

Aos meus amigos de longa data, Apura, Vlad, Paupério, Gabriel, Tiago e Jorge pelos momentos espetaculares e por todas as aventuras partilhadas ao longo destes anos.

Quero também agradecer ao Toni, à Mafas, ao Tiago, à Érica e ao André por me mostrarem sempre que posso dar um pouco mais do que tenho.

Ao Marco, ao Luís, à Carolina, à Lisa e ao Policarpo por me fazerem arriscar e seguir o sonho de voar cada vez mais alto.

Ao Lemos, ao Rúben e à Kika pela amizade espetacular.

Tenho especialmente que agradecer à minha família que sempre me motivou e nunca me deixou desistir. Ao Paulo, à Marta, à Ana. Ao meu pai, ao Tomi, à Lúce e à minha mãe.

À Sofia.

Resumo

Os últimos 70 anos têm vindo a registar um aumento de detritos plásticos encontrados no oceano. Estes englobam uma quantidade considerável de lixo derivado de atividades piscatórias, sendo estas maioritariamente artes de pesca perdidas ou deliberadamente abandonadas, denominadas de *Ghost Nets*. Dada a necessidade da preservação do ambiente, um *Autonomous Underwater Vehicle* (AUV) poderá servir como plataforma para a deteção e recuperação destas redes.

Assim, a presente dissertação insere-se no contexto do projeto NetTag. Esta propõe o desenvolvimento de uma solução simulada para a navegação do AUV *Inspection and Recovery Intelligent System* (IRIS) baseada na deteção visual de *Ghost Nets* com recurso a *Deep Learning*, a fim de conduzir o veículo ao encontro das mesmas para a sua posterior recuperação.

Desta forma, foram desenvolvidos os diferentes componentes do sistema, tendo por base o Stonefish para o ambiente de simulação. Este é lançado e gerido por um orquestrador que controla o modelo do IRIS com base nas inferências executadas pela sua câmara frontal. Estes valores traduzem-se, posteriormente, em comandos destinados aos *thrusters*. Por sua vez, as inferências fundamentam-se no modelo YOLOv5 nano treinado com um *dataset* criado para esta dissertação.

Após a validação isolada de cada componente, procedeu-se ao teste da simulação completa que incorporou todos os componentes desenvolvidos. Foram obtidos resultados satisfatórios, tendo-se verificado a aproximação do submarino à rede de forma autónoma.

Palavras-Chave: *Ghost Net*, *Ghost Fishing*, Simulação, Stonefish, AUV, IRIS, *Deep Learning*, YOLO, ROS.

Abstract

The last 70 years have been registering an increase of plastic debris found in the ocean. These encompass a considerable amount of waste derived from fishing activities, being mostly lost fishing nets or deliberately abandoned, called *Ghost Nets*. Given the necessity to preserve the environment, an AUV may serve as a platform for the detection and recovery of these nets.

Therefore, this dissertation should be seen in the context of the NetTag project. It proposes the development of a simulated solution for the navigation of the AUV IRIS based on the visual detection of *Ghost Nets* using *Deep Learning*, in order to lead the vehicle to meet them for later recovery.

In this way, the different components of the system have been developed, having Stonefish as the basis of the simulation environment. This is launched and managed by an orchestrator that controls the IRIS model based on the executed inferences through its frontal camera. These values are later translated into commands destined to the thrusters. The inferences are fundamented on the YOLOv5 nano model, trained with a custom dataset created for this dissertation.

After the isolated validation of each component, the full simulation that incorporated all the developed components was tested. Satisfactory results were achieved, by having verified the autonomous approach carried out by the submarine to the net.

Keywords: *Ghost Net*, *Ghost Fishing*, Simulação, Stonefish, AUV, IRIS, *Deep Learning*, YOLO, ROS.

Índice

Lista de Figuras	ix
Lista de Tabelas	xiii
Lista de Acrónimos	xv
1 Introdução	1
1.1 Âmbito da Dissertação	2
1.2 Objetivos	3
1.2.1 Resultados esperados	3
1.3 Plano de Trabalho	4
1.4 Estrutura	4
2 Formulação do Problema	5
2.1 <i>Ghost Fishing</i>	5
2.2 Caracterização das artes de pesca na costa portuguesa	7
2.3 Discussão	10
3 Estado da Arte	11
3.1 <i>Deep Learning</i>	11
3.1.1 <i>Convolutional Neural Networks</i> (CNNs)	12
3.1.2 <i>MobileNet</i>	13
3.1.3 <i>You Only Look Once</i> (YOLO)	14
3.2 Métodos de <i>Deep Learning</i> aplicados em AUVs	16
3.2.1 <i>Manipulation Purpose Underwater Agent Vehicle for Ghost Net Recovery Mission</i>	16
3.2.2 <i>Deep Generative Network and Regression Network for Fishing Nets Detection in Real-time</i>	18
3.2.3 <i>Real-time Detection Method of Marine Small Object with Underwater Robot Vision</i>	19
3.2.4 <i>Perception and Recognition of Underwater Cable Based on Deep Learning</i>	20
3.3 Outros métodos de deteção aplicados em AUVs	22

3.3.1	<i>What's mine is yours: Pretrained CNNs for limited training sonar ATR</i>	22
3.3.2	<i>An adaptive method of damage detection for fishing nets based on image processing technology</i>	23
3.3.3	<i>Research on Recognition Method of Optical Detection Image of Underwater Robot for Submarine Cable</i>	24
3.4	Discussão do Estado da Arte	24
4	Projeto	27
4.1	Domínio do problema	27
4.1.1	<i>Ghost Fishing</i>	27
4.1.2	Análise do IRIS	28
4.2	Requisitos do Sistema	29
4.2.1	Requisitos para o ambiente de simulação	29
4.2.2	Requisitos para a detecção de <i>Ghost Nets</i>	30
4.2.3	Requisitos para a manobra de aproximação à <i>Ghost Net</i>	30
4.3	Domínio da Solução	30
4.3.1	Ambiente de simulação	30
4.3.2	Deteção visual de <i>Ghost Nets</i>	32
	Características do <i>dataset</i>	32
	Escolha dos algoritmos de <i>Deep Learning</i>	32
4.3.3	Navegação e Controlo por Deteção Visual	33
4.3.4	Arquitetura do <i>Software</i>	34
5	Implementação e Resultados	37
5.1	Configuração da simulação	37
5.2	<i>Dataset</i>	39
5.3	Considerações prévias ao treino dos modelos	42
5.4	Treino do YOLOv5	43
5.4.1	YOLOv5 nano com <i>batch size</i> 64	43
5.4.2	YOLOv5 nano com <i>batch size</i> 32	46
5.4.3	YOLOv5 nano com <i>batch size</i> 128	49
5.5	Treino do YOLOv8	52
5.5.1	YOLOv8 nano com <i>batch size</i> 64	52
5.5.2	YOLOv8 nano com <i>batch size</i> 32	55
5.5.3	YOLOv8 nano com <i>batch size</i> 128	58
5.6	Inferência nas imagens de teste	61
5.7	Inferência no ambiente simulado	63
5.8	Controlo visual da IRIS	64
5.8.1	IRIS Vision	65
5.8.2	IRIS Control	66

5.9	Teste da simulação do IRIS	68
6	Conclusões	71
6.1	Trabalho Futuro	72
	Referências	73

Lista de Figuras

1.1	IRIS AUV	3
2.1	Fotografias de uma tartaruga envolta numa <i>Ghost Net</i> e de um caranguejo na mesma situação, num recife de coral	6
2.2	Armadilha submersa nas ilhas Órcades, Escócia	6
2.3	Rede emaranhada em navio naufragado no Mar do Norte	7
2.4	Representação da arte de Cerco	7
2.5	Redes de Arrasto	8
2.6	Rede de Emalhar e Tresmalho e rede Envolvente Arrastante (Arte Xávega)	9
2.7	Representação esquemática de armadilhas	9
2.8	Palangre	10
3.1	Arquitetura de alto nível de uma R-CNN	12
3.2	Arquitetura de alto nível de uma Faster R-CNN	13
3.3	<i>Depthwise Seperable Convolution</i>	13
3.4	Arquitetura da MobileNetSSDv2	14
3.5	<i>Performance</i> do YOLOv3	15
3.6	<i>Performance</i> do YOLOv5	15
3.7	Latência do YOLOv8	15
3.8	<i>Agent</i> AUV	17
3.9	Deteção da <i>Ghost Net</i> em tempo real com YOLO v3-Tiny	17
3.10	Determinação da captura da rede	17
3.11	Resultado do treino com o <i>dataset</i> aumentado com DCGAN	18
3.12	Resultados experimentais de deteção e seguimento de organismos marinhos	19
3.13	Diagrama ilustrativo do algoritmo de <i>Underwater Image Enhancement</i>	21
3.14	Imagem original e processada	21
3.15	Gráfico das métricas dos modelos de Sonar	23
3.16	Deteção de pontos de inflexão numa rede	23
3.17	Imagem pré-processada à esquerda; imagem com segmentação ao centro; imagem com filtros de <i>edge detection</i> à direita	24
4.1	Modelo do IRIS	28

4.2	Modelo do AUV Girona 500 no Stonefish	31
4.3	Diagrama de blocos da arquitetura	35
5.1	IRIS no Stonefish	38
5.2	Modelos de artes de pesca criadas em Blender	38
5.3	Imagens do <i>dataset</i>	39
5.4	Imagens representativas das classes <i>fish</i> e <i>crab</i>	40
5.5	Imagens acrescentadas por <i>augmentation</i>	41
5.6	<i>Augmentation</i> por <i>Tiling</i>	41
5.7	Primeiros resultados do YOLOv5 nano	43
5.8	Matriz de Confusão do YOLOv5 nano	44
5.9	Inferência do YOLOv5 nano no <i>dataset</i> de validação	45
5.10	Resultado de um treino do YOLOv5 nano com um <i>batch size</i> de 32	46
5.11	Matriz de confusão do YOLOv5 nano com um <i>batch size</i> de 32	47
5.12	Inferência do YOLOv5 nano com um <i>batch size</i> de 32 no <i>dataset</i> de validação	48
5.13	Resultado de um treino do YOLOv5 nano com um <i>batch size</i> de 128	49
5.14	Matriz de confusão do YOLOv5 nano com um <i>batch size</i> de 128	50
5.15	Inferência do YOLOv5 nano com um <i>batch size</i> de 128 no <i>dataset</i> de validação	51
5.16	Resultados do YOLOv8 nano com um <i>batch size</i> de 64	52
5.17	Matriz de confusão do YOLOv8 nano com um <i>batch size</i> de 64	53
5.18	Inferência do YOLOv8 nano com um <i>batch size</i> de 64 no <i>dataset</i> de validação	54
5.19	Resultado do treino do YOLOv8 nano com <i>batch size</i> 32	55
5.20	Matriz de confusão do YOLOv8 nano com <i>batch size</i> 32	56
5.21	Inferência do YOLOv8 nano com <i>batch size</i> 32 no <i>dataset</i> de validação	57
5.22	Resultado do treino do YOLOv8 nano com <i>batch size</i> 128	58
5.23	Matriz de confusão do YOLOv8 nano com <i>batch size</i> 128	59
5.24	Inferência do YOLOv8 nano com <i>batch size</i> 128 no <i>dataset</i> de validação	60
5.25	Inferência dos modelos em imagens do <i>test set</i> com boa visibilidade. YOLOv5 à esquerda e YOLOv8 à direita	61
5.26	Inferência dos modelos em imagens do <i>test set</i> com fraca visibilidade. YOLOv5 à esquerda e YOLOv8 à direita	62
5.27	Armadilhas no Stonefish em diferentes condições de visibilidade	63
5.28	Rede de Emalhar no Stonefish em diferentes condições de visibilidade	64
5.29	Diagrama de fluxo do Ghost Net Inference	65
5.30	Diagrama de fluxo do Setpoints Visual Navigation	66
5.31	Estratégia para o controlo visual do veículo	67

5.32	Simulação da navegação visual do IRIS. Encontra-se à esquerda o cenário de simulação e os valores publicados nos <i>thrusters</i> . À direita, visualiza-se a inferência que o veículo executa	68
5.33	Estabilização do IRIS na distância de segurança à rede	69
5.34	Simulação da navegação visual do IRIS com um valor de <i>Jerlov</i> de 18%	69
5.35	Simulação da navegação visual do IRIS com um valor de <i>Jerlov</i> de 38%	70

Lista de Tabelas

3.1	Comparação dos resultados obtidos entre os 3 modelos treinados com o <i>dataset</i> e o aumentado	18
3.2	Comparação entre Faster R-CNN ZF e VGG-16	19
3.3	Resultados obtidos nos treinos	20
3.4	Métricas obtidas com os modelos treinados com dados de Sonar . . .	22
4.1	Características dos YOLOv5 e YOLOv8	33
5.1	Hiperparâmetros - com <i>batch size</i> de 32 - utilizados num treino do YOLOv5 nano	46
5.2	Hiperparâmetros - com <i>batch size</i> de 128 - utilizados num treino do YOLOv5 nano	49
5.3	Hiperparâmetros - com <i>batch size</i> de 64 - utilizados num treino do YOLOv8 nano	52
5.4	Hiperparâmetros - com <i>batch size</i> de 32 - utilizados num treino do YOLOv8 nano	55
5.5	Hiperparâmetros - com <i>batch size</i> de 128 - utilizados num treino do YOLOv8 nano	58

Lista de Acrónimos

ATR	<i>Automatic Target Recognition</i>
AUV	<i>Autonomous Underwater Vehicle</i>
BOW	<i>Bag-of-Words</i>
CCC	<i>Chinese Control Conference</i>
CIIMAR	Centro Interdisciplinar de Investigação Marinha e Ambiental
CNN	<i>Convolutional Neural Network</i>
CPU	<i>Central Processing Unit</i>
CSPNet	<i>Cross Stage Partial Network</i>
CV	<i>Computer Vision</i>
DCGAN	<i>Deep Convolution Generative Adversarial Network</i>
DGRM	Direção-Geral de Recursos Naturais, Segurança e Serviços Marítimos
DL	<i>Deep Learning</i>
DSC	<i>Depthwise Seperable Convolution</i>
Fast R-CNN	<i>Fast Region-based Convolutional Neural Network</i>
Faster R-CNN	<i>Faster Region-based Convolutional Neural Network</i>
GF	<i>Ghost Fishing</i>
GN	<i>Ghost Net</i>
GPU	<i>Graphics Processing Unit</i>
HCD	<i>Harris Corner Detector</i>
IA	Inteligência Artificial
ICSSE	<i>International Conference on System Science and Engineering</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>

INESC TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
IoU	<i>Intersection over Union</i>
IRIS	<i>Inspection and Recovery Intelligent System</i>
IROS	<i>International Conference on Intelligent Robots and Systems</i>
ISEP	Instituto Superior de Engenharia do Porto
KFC	<i>Kernelized Correlation Filter</i>
LSA	Laboratório de Sistemas Autónomos
mAP	<i>mean Average Precision</i>
ML	<i>Machine Learning</i>
ONG	Organização Não-Governamental
ONNX	<i>Open Neural Network Exchange</i>
PI	<i>Proportional-Integral</i>
PID	<i>Proportional-Integral-Derivative</i>
R-CNN	<i>Region-based Convolutional Neural Network</i>
ROS	<i>Robot Operating System</i>
ROV	<i>Remotely Operated Vehicle</i>
RPN	<i>Region Proposal Network</i>
RSJ	<i>The Robotics Society of Japan</i>
SGD	<i>Stochastic Gradient Descent</i>
SRC	<i>Sparse Reconstruction-based Classifier</i>
SSD	<i>Single Shot Detector</i>
SVM	<i>Support Vector Machine</i>
VGG	<i>Visual Geometry Group</i>
YOLO	<i>You Only Look Once</i>

Capítulo 1

Introdução

Desde a década de 50 que se regista um aumento na quantidade de detritos plásticos encontrados no oceano. A adoção destes materiais como substitutos de outros menos duradouros, aliada à falta de uma gestão eficaz no término de plásticos em fim de vida, contribuem diretamente para o crescimento deste fenómeno [1]. Todos os anos, são estimadas que 4,8 a 12,7 milhões de toneladas de plástico sejam lançadas ao mar. Este intervalo na estimativa deve-se à ausência de estudos que quantifiquem, com precisão, o lixo marinho e a sua origem [2, 3].

Para além dos plásticos mais comuns, estão também presentes detritos oriundos de atividades piscatórias. Tratam-se de redes e armadilhas utilizadas na pesca, também conhecidas como artes de pesca, que foram perdidas no mar. Embora não sejam tão visíveis como outros tipos de lixo, não são menos prejudiciais ao ecossistema marinho. Prevê-se que as artes de pesca possam representar até 10% do lixo marinho a nível global [2, 4].

Um dos maiores problemas que estas redes representam trata-se do *Ghost Fishing* (GF). Este termo é associado a artes de pesca perdidas ou deliberadamente abandonadas que, de forma passiva, continuam a pescar e a provocar a morte de diversos organismos marinhos. São denominadas de *Ghost Nets* (GNs) [5].

Existe, portanto, a necessidade da monitorização dos ambientes marinhos para a sua consequente preservação. Para esse propósito, são utilizados *Remotely Operated Vehicles* (ROVs) e *Autonomous Underwater Vehicles* (AUVs). Estes veículos analisam fatores como a abundância e diversidade biológica, a poluição, o tráfego marítimo, indicadores de aquecimento global, entre outros [6].

A deteção de artes de pesca perdidas com recurso a *Computer Vision* (CV) pode ser adicionada ao leque de ações destes veículos. São diversos os métodos para alcançar esta funcionalidade, como é o caso de algoritmos de *Object Detection* baseados em *Deep Learning* (DL). No entanto, deve ser considerada a limitação computacional dos veículos, visto que pode implicar resultados insatisfatórios [7].

A obtenção de um sistema que descreva um controlo estável do veículo pode ser difícil de alcançar, já que a implementação direta no AUV requer um ciclo de desenvolvimento moroso. É necessário um conjunto de iterações que contem com o *deployment* do *software* para o veículo e a sua posterior validação num tanque de água, antes de ser lançado ao oceano.

Para além disso, e numa nota mais comedida, devem ser considerados os riscos relacionados à missão em mar. Um veículo que esteja dotado de navegação autónoma com base na deteção visual de GNs está exposto às condicionantes do oceano. Fala-se de correntes, de fraca visibilidade e de terreno irregular. O AUV deve ser robusto a estes fatores a fim de garantir o sucesso da missão. Assim, destaca-se o potencial da simulação, ao fornecer um ambiente controlado para o desenvolvimento do sistema.

Considerando estas características, a dissertação procura desenvolver uma solução, em ambiente simulado, que permita a navegação autónoma de um AUV com base na deteção visual de artes de pesca.

1.1 Âmbito da Dissertação

Desenvolvida no âmbito do mestrado de Engenharia Eletrotécnica e de Computadores, ramo de Sistemas Autónomos do Instituto Superior de Engenharia do Porto (ISEP), esta dissertação insere-se no contexto do projeto *Tagging Fishing Gears and Enhancing on Board Best-Practices to Promote Waste Free Fisheries* (NetTag). Formado por um consórcio de parceiros internacionais, o projeto procura apresentar uma solução para a mitigação do lixo gerado pela atividade piscatória. Para tal, é composto por uma vertente de sensibilização e outra tecnológica [8].

O Laboratório de Sistemas Autónomos (LSA) do ISEP é um centro de robótica do Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC) que participa neste projeto. O laboratório é responsável pelo desenvolvimento do *Inspection and Recovery Intelligent System* (IRIS), representado na Figura 1.1. Sendo este uma parte integrante da vertente tecnológica do projeto, trata-se de um AUV cuja missão consiste na localização e recuperação de artes de pesca perdidas no mar [9].

Por sua vez, o veículo é munido de sensores que o habilitam a navegar de forma acústica e visual até à respetiva arte de pesca. Deste modo, a presente dissertação assume o IRIS como a plataforma a ser integrada na simulação. No entanto, a

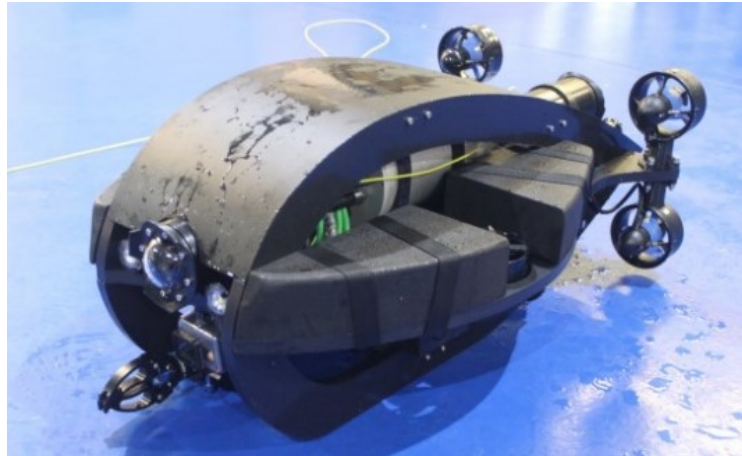


Figura 1.1: IRIS AUV [9]

navegação acústica não é aqui considerada, sendo apenas retratada a validação da detecção visual de *Ghost Nets* e consequente aproximação às mesmas.

1.2 Objetivos

Tendo por base a contextualização fornecida, o presente trabalho procura o desenvolvimento de uma simulação na qual o IRIS seja capaz de navegar de forma autónoma por intermédio da detecção de GNs. Por sua vez, o simulador permite uma iteração célere, executando testes e atualizações ao *software on-demand*. É um fator que acelera o processo de validação do eventual algoritmo a ser implementado no veículo real. Assim, o desenvolvimento desta dissertação incide nos seguintes objetivos:

- Integração do IRIS num simulador;
- Análise e desenvolvimento de modelos de *Deep Learning* para a detecção de *Ghost Nets*;
- Comparação e validação da inferência dos modelos no simulador;
- Desenvolvimento do controlador visual do AUV;
- Validação da navegação visual do veículo.

1.2.1 Resultados esperados

É expectável que esta dissertação forneça, em ambiente simulado, uma solução à navegação visual e autónoma do IRIS. A simulação deve contar com um cenário que se assemelhe ao encontrado no oceano, dispondo de artes de pesca afundadas. O veículo deverá ser capaz de detetar estas redes pela imagem captada pela câmara frontal e navegar ao encontro das mesmas até uma determinada distância de segurança.

1.3 Plano de Trabalho

É importante conhecer primeiramente o problema a ser tratado. Logo, deve ser realizado um estudo detalhado da problemática do *Ghost Fishing*. De seguida, é necessário compreender melhor os diferentes métodos de *Computer Vision* que serão relevantes para a deteção das redes, assim como aplicações já existentes em cenários semelhantes.

Dado por concluído este estudo, deverá fazer-se um levantamento dos requisitos do sistema para a posterior elaboração do domínio da solução. Posto isto, seguir-se-á a fase de implementação, onde será configurado o simulador, desenvolvido o modelo de deteção e ainda o controlador do veículo.

Por fim, deve ocorrer a integração de todos os componentes desenvolvidos numa simulação, com o intuito de validar a navegação visual do veículo.

1.4 Estrutura

Esta dissertação está organizada em 6 capítulos.

O Capítulo 1 introduz o tema desta dissertação, apresentando o seu âmbito e objetivos.

Segue-se o Capítulo 2 com a formulação do problema. O presente aborda a temática do *Ghost Fishing*, explicando em pormenor do que se trata e as implicações que representa à vida marinha. Esta explicação é complementada com uma caracterização dos diferentes tipos de artes de pesca encontrados ao longo da costa portuguesa.

Já o Capítulo 3 realiza um estudo a técnicas de *Deep Learning* e as possíveis aplicações que as mesmas possam apresentar em AUVs. O capítulo faz ainda uma breve análise a técnicas alternativas, terminando com uma discussão do estudado.

Por sua vez, o Capítulo 4 elabora o domínio do problema do ponto de vista técnico e faz também o levantamento dos requisitos propostos. Posto isto, procede com o domínio da solução, discriminando as diferentes etapas a realizar.

O Capítulo 5 detalha a implementação realizada, contando com o *setup* do simulador, levantamento do *dataset*, treino e validação dos modelos, nós de controlo e posterior integração de todos os componentes.

Por fim, o Capítulo 6 apresenta as conclusões retiradas sobre o projeto elaborado, assim como indica o trabalho futuro a ser realizado.

Capítulo 2

Formulação do Problema

A problemática das artes de pesca perdidas no oceano deve ser devidamente compreendida, sendo estas consideravelmente prejudiciais ao mesmo. Assim, este capítulo efetua um breve estudo sobre o tópico do *Ghost Fishing* e as suas consequências, seguindo-se uma análise das diferentes redes de pesca e das suas características.

2.1 *Ghost Fishing*

Uma vez perdidas no mar, as artes de pesca demonstram uma tendência para continuar a pescar por um período de tempo indeterminado. Este fenómeno é denominado de *Ghost Fishing* [4]. Para além de danificarem o fundo marinho, estas redes são responsáveis por emaranhar e encurralar diversos animais, eventualmente levando-os à morte. Neste contexto, a Figura 2.1 ilustra uma tartaruga presa numa *Ghost Net*, à esquerda, juntamente com um caranguejo num recife de coral envolto numa rede, à direita. Na maioria dos casos, as redes em questão tratam-se de artes de fundo, sendo depositadas no solo marinho para a captura das diferentes espécies. Uma vez perdidas, são bastantes difíceis de localizar dado o arraste sofrido por correntes marítimas [9].

Neste sentido, é destacada a diferença entre artes de pesca perdidas e abandonadas. As primeiras devem-se a um leque de fatores que estão fora da jurisdição dos pescadores. É o caso de condições meteorológicas adversas, problemas técnicos e outras condicionantes que conduzem à perda do equipamento. Por outro lado, o abandono é um ato deliberado que se regista em tempos de submersão acima do

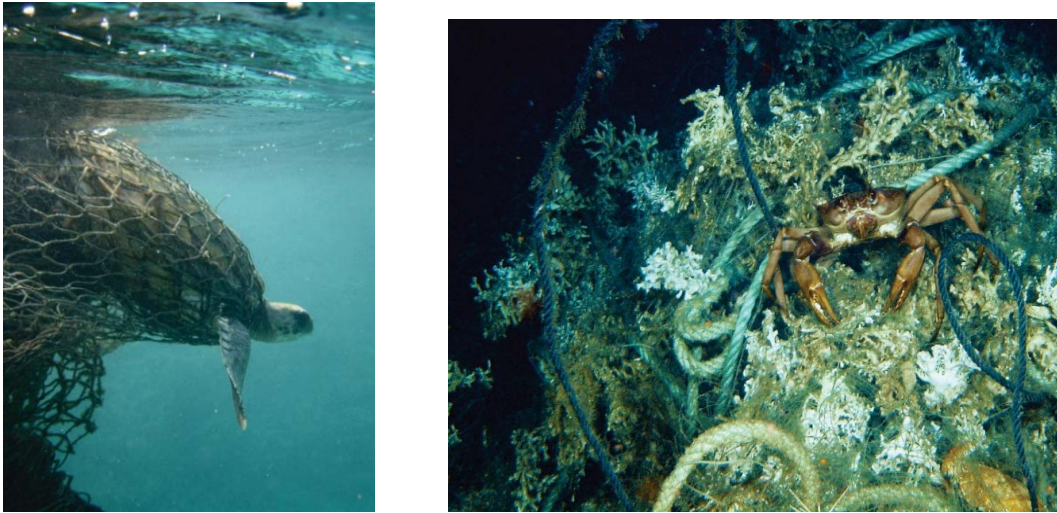


Figura 2.1: Fotografias de uma tartaruga envolta numa *Ghost Net* e de um caranguejo na mesma situação, num recife de coral [1]

legal. Pode ser motivado pelo despejo definitivo do equipamento, por incapacidade de recuperação do mesmo ou para pesca excessiva [10].

Tendo em consideração os diferentes fatores do oceano aos quais as artes perdidas estão expostas, é de notar a alteração na sua aparência. Dependendo do estado de degradação, aliado a uma fraca visibilidade, as *Ghost Nets* podem inclusive serem confundidas com o fundo marinho. As Figuras 2.2 e 2.3 mostram dois exemplos deste cenário.

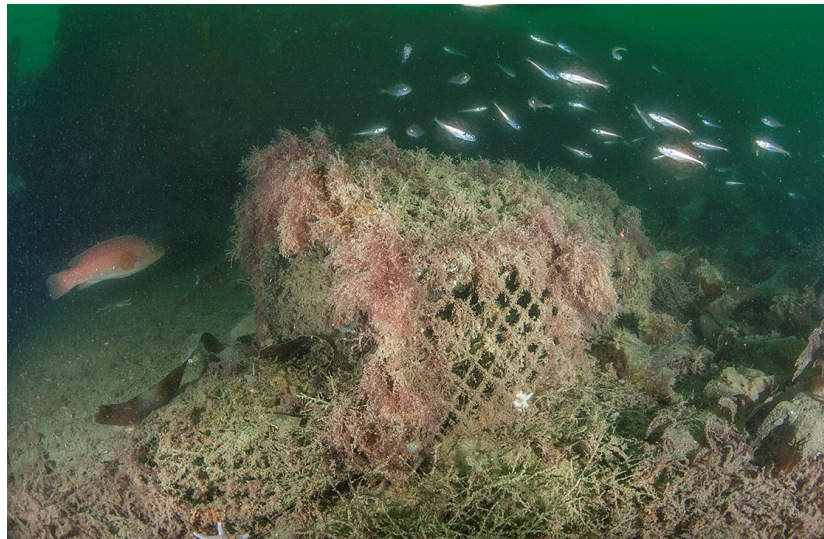


Figura 2.2: Armadilha submersa nas ilhas Órcades, Escócia [11]

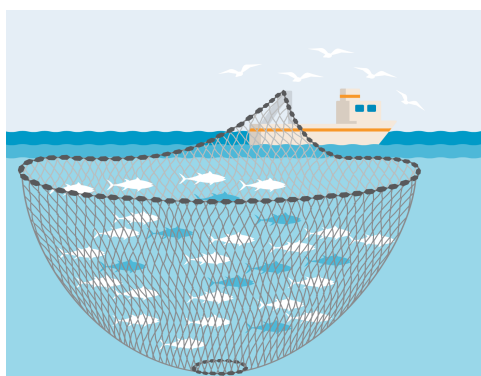


Figura 2.3: Rede emaranhada em navio naufragado no Mar do Norte [11]

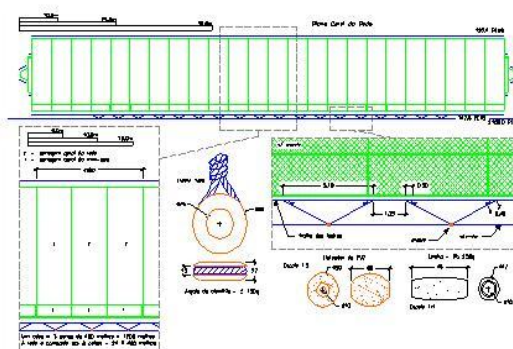
2.2 Caracterização das artes de pesca na costa portuguesa

A maioria do lixo presente nos diferentes meios aquáticos é composto por plástico. Por norma, é dividido em macroplásticos, microplásticos e ainda nanoplásticos, partículas com diâmetro inferior a 5 mm [2]. As redes de pesca inserem-se nos macroplásticos, sendo as mais predominantes nesta categoria. Em Portugal, os principais tipos de pesca estão definidos por sete portarias da Direção-Geral de Recursos Naturais, Segurança e Serviços Marítimos (DGRM) que definem as características das respetivas artes, classificando-se em Cerco, Arrasto, Redes de Emalhar e de Tresmalho, Anzol, Armadilhas, Envolventes Arrastantes e ainda a Apanha [13].

Analisando as artes de maior dimensão, o Cerco (Figura 2.4) remete para qualquer método que utilize uma parede de rede longa, tendo em vista a captura de



(a) Ilustração do Cerco



(b) Esquema da rede

Figura 2.4: Representação de arte de Cerco [12]

espécies pelágicas. Lançada ao mar a partir de embarcações, esta rede pretende envolver o peixe e impedir a sua fuga ao usar uma retenida na parte inferior, fechando a bolsa quando a rede está cheia [12].

O Arrasto (Figura 2.5) engloba estruturas rebocadas, sendo que se divide em Arrasto de Portas, Arrasto de Vara e Ganchorra [14]. Construídas para pesca de fundo, dispõem de uma bolsa formada por uma malha, onde as duas primeiras são de média ou grande dimensão, pelo que são utilizadas a partir de embarcações. Por sua vez, a Ganchorra apresenta-se como uma arte de pesca mais pequena, podendo ser rebocada a pé.

Alguns estudos afirmam que o *Ghost Fishing* deve-se, em parte, às redes de Emalhar, de Tresmalho e às Armadilhas [10]. Retratado na Figura 2.6, o método de Emalhar e de Tresmalho recorre a redes retangulares que permanecem no plano vertical. São mantidas no fundo marinho por lastros ou à superfície por bóias [15]. Ainda na mesma figura, a Envolvente Arrastante, também conhecida por Arte Xá-vega, é composta por uma bolsa de malha com asas laterais, permitindo o seu arrasto e o cerco do peixe. Pode ser largada a partir de uma embarcação ou a partir de terra [16].

As Armadilhas distinguem-se substancialmente das restantes artes, quer na forma,

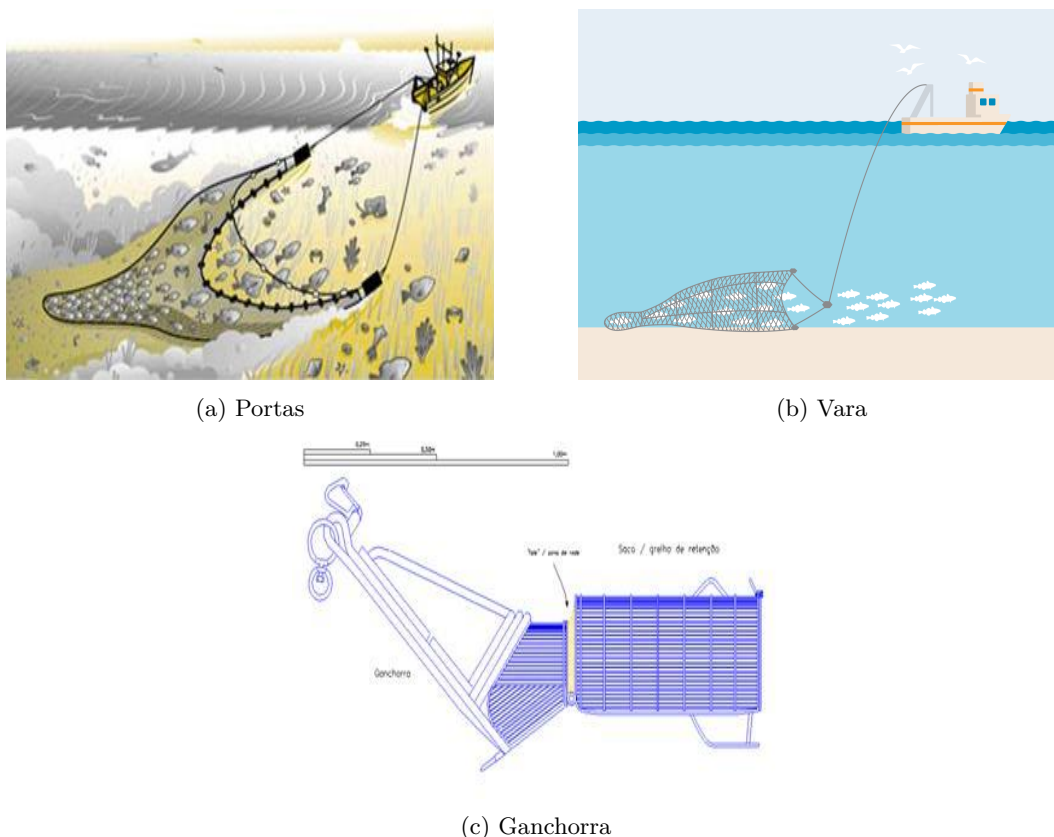
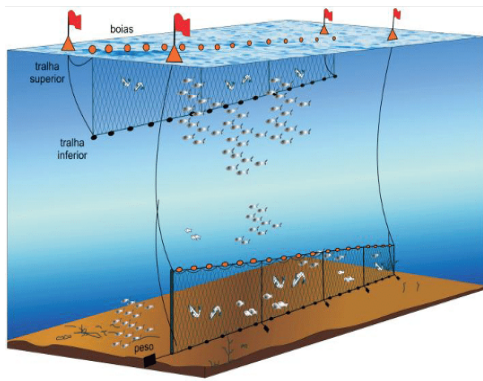


Figura 2.5: Redes de Arrasto [14]



(a) Rede de Emalhar e Tresmalho [17]



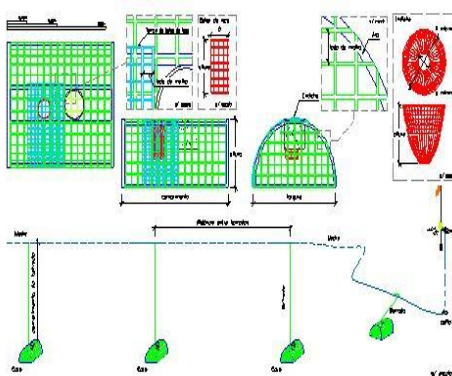
(b) Pescadores em terra com Arte Xávega [16]

Figura 2.6: Rede de Emalhar e Tresmalho e rede Envolvente Arrastante (Arte Xávega)

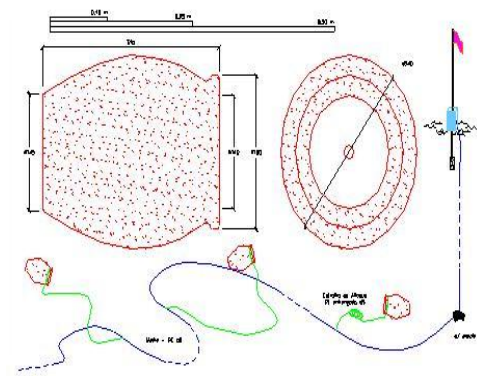
quer na função. São utilizadas para pesca de fundo, permanecendo propositadamente submersas por longos períodos de tempo. Observando a Figura 2.7, encontra-se, à esquerda, uma Armadilha de Gaiola. Constituída por uma estrutura rígida, a mesma é envolta numa rede. Verifica-se que uma única corda entreliga as diferentes armadilhas à superfície, servindo de sinalização para a sua recuperação. As Armadilhas de Armação seguem a mesma estrutura, embora maiores. À direita desta figura, encontra-se uma Armadilha de Abrigo. Apresentando-se como um pote de barro, esta não dispõe de qualquer tipo de malha, estando igualmente ligada à superfície por uma corda. Estes abrigos são concebidos para a captura de polvo.

Para além da Armadilha de Abrigo, o Anzol é uma arte de pesca que não apresenta uma malha. Dispõe de uma ou mais linhas, dependendo do número de anzóis, recorrendo a bóias e lastros. Por fim, a Figura 2.8 ilustra um Palangre, utilizado na pesca de fundo. O Palangre é composto por uma linha de grande comprimento da qual derivam os estralhos, linhas mais pequenas onde se encontram vários anzóis.

Das sete portarias, a Apanha é a única que não utiliza redes ou armadilhas.



(a) Armadilha de Gaiola



(b) Armadilha de Abrigo

Figura 2.7: Representação esquemática de armadilhas [13]

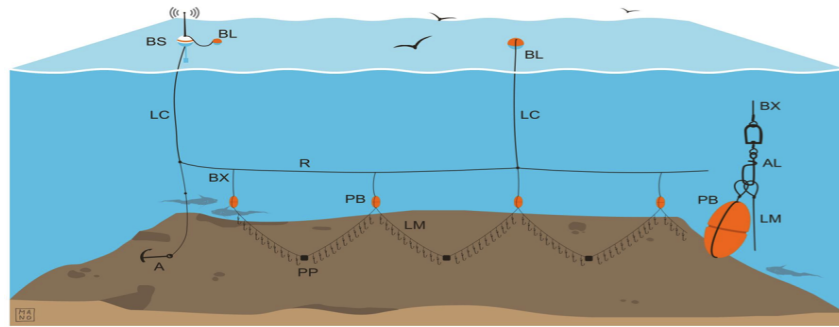


Figura 2.8: Palangre [18]

2.3 Discussão

Com base neste breve estudo, verifica-se que, de um modo geral, as artes de fundo aparentam ser as maiores contribuintes para o fenómeno do *Ghost Fishing*, como é o caso das redes de Emalhar, de Tresmalho e as Armadilhas. Não obstante, a perda de uma rede de Cerco pode implicar consequências graves, sendo capaz de encurrular um grande número de espécies marinhas e levá-las à morte.

Tendo em consideração as estruturas destas artes, todas apresentam um padrão relativamente semelhante à exceção do anzol. Uma *Ghost Net* é facilmente distinguida por este motivo. Contudo, como analisado, a sua deteção é agravada por um conjunto de variáveis.

Quando submersas, as redes sofrem uma degradação causada por fatores inerentes ao oceano, alterando o seu aspeto. Em certas ocasiões, a sua localização torna-se bastante complicada devido à sua camuflagem com o meio. Estas redes podem estar inseridas em locais como naufrágios ou recifes de coral, onde o terreno não é regular. A acumulação de pequenos organismos marinhos nas suas superfícies contribui igualmente para a camuflagem das GNs. Para além do mais, devem ser tidos em consideração aspetos como a sustentação de partículas na água, visto que diminuem bastante a visibilidade.

Assim, deve ser encontrada uma solução que permita a deteção visual de *Ghost Nets* nestas condições.

Capítulo 3

Estado da Arte

A análise anterior despoletou a necessidade de uma investigação sobre as técnicas existentes de deteção de objetos. Deste modo, o presente capítulo explora a aplicação do *Deep Learning* em *Computer Vision*, descrevendo sucintamente alguns modelos para o efeito. De seguida, é apresentado um conjunto de aplicações encontradas em ROVs e AUVs para a deteção de objetos submersos, complementando ainda com uma revisão de técnicas alternativas que se mostram como possíveis soluções ao problema.

3.1 *Deep Learning*

Numa fase introdutória, é importante compreender que o *Machine Learning* (ML) é um tipo de Inteligência Artificial (IA). Sob este contexto, o ML engloba os algoritmos que, com base nos dados e parâmetros de entrada fornecidos, são capazes de aprender automaticamente a informação de interesse durante o seu treino. O resultado deste processo consiste num modelo treinado, tendo em vista a posterior inferência em novos dados para análise e classificação dos mesmos [19].

Por sua vez, o *Deep Learning* requer um volume de dados consideravelmente maior. Funciona por intermédio de redes neuronais compostas por unidades computacionais, os *neurons*, e no que diz respeito aos problemas de *Computer Vision*, o *Deep Learning* engloba casos como *Classification*, *Semantic Segmentation* e *Object Detection* [20].

3.1.1 Convolutional Neural Networks (CNNs)

Uma *Convolutional Neural Network* (CNN) é uma rede neuronal cuja arquitetura se enquadra em *Deep Learning*. Ao contrário das redes neurais tradicionais, as CNNs aplicam convoluções que, fundamentalmente, consistem na combinação de duas funções para gerar uma terceira [21]. Perante o processamento de uma imagem, uma CNN extrai as *features* relevantes da mesma como é o caso de *edges* e texturas que são depois submetidas às diferentes camadas do modelo.

Por sua vez, uma *Region-based Convolutional Neural Network* (R-CNN) começa por dividir a imagem em diversas regiões de interesse, denominadas de *region proposals*. Estas mostram uma elevada probabilidade de conter os objetos de interesse, pelo que o algoritmo extrai cerca de 2000 *features* a partir das mesmas. As regiões de interesse são posteriormente convertidas para o tamanho de entrada da CNN, um processo denominado de *warping*. O processamento das *features* permitirá a classificação das *region proposals*, ditando se imagem contém, de facto, os objetos de interesse ou não, como demonstrado na Figura 3.1 [22].

Tendo por base as redes já descritas, em 2016 surgiu a *Faster Region-based Convolutional Neural Network* (Faster R-CNN). Pese embora não seja relativamente recente, a alta precisão que a caracteriza torna-a bastante conhecida e recorrente em diferentes projetos. A arquitetura, exposta na Figura 3.2, é composta por dois módulos, nomeadamente uma *Region Proposal Network* (RPN) e uma *Fast Region-based Convolutional Neural Network* (Fast R-CNN) [23].

A RPN trata-se de uma pequena rede neuronal que identifica e fornece as diferentes regiões de interesse da imagem à Fast R-CNN. Já a Fast R-CNN consiste numa *deep network* que executa *Object Detection* nas regiões assinaladas. Em suma, é esta relação de dependência que torna o modelo mais rápido e preciso em relação aos seus antecessores.

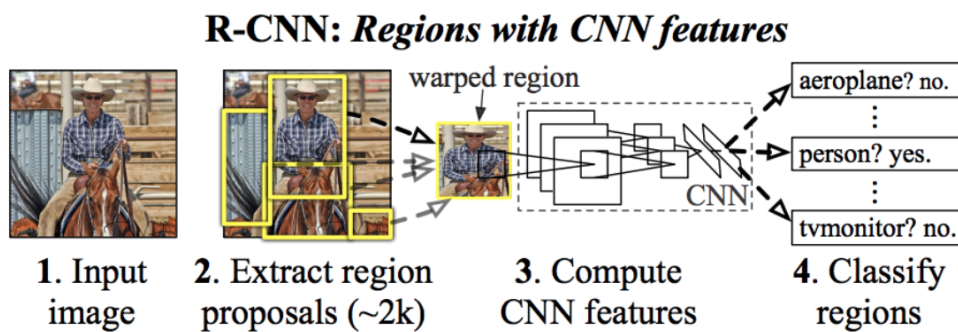


Figura 3.1: Arquitetura de alto nível de uma R-CNN [22]

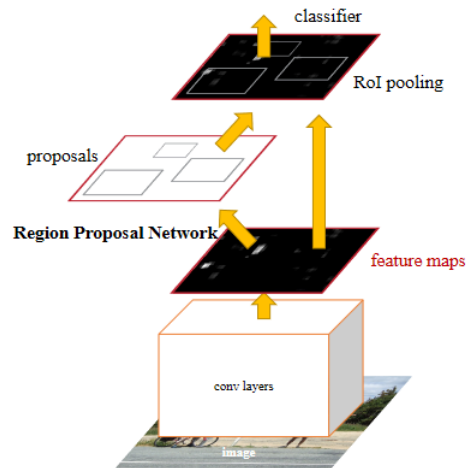


Figura 3.2: Arquitetura de alto nível de uma Faster R-CNN [23]

3.1.2 MobileNet

Lançada em 2017 pela Google, a *MobileNet* apresenta uma vasta gama de aplicações em *Object Detection*, *Semantic Segmentation* e *Image Classification*. Foi desenvolvida com o propósito de ser integrada em sistemas móveis e embecidos que apresentem uma capacidade computacional limitada. Para o efeito, a arquitetura da mesma recorre a *Depthwise Seperable Convolutions* (DSCs) [24].

Para gerar um conjunto de *outputs*, uma convolução dita normal combina os filtros (ou *kernels*) com os *inputs* do modelo num único passo. No entanto, a aproximação levada a cabo pelas DSCs divide este processo em duas camadas sequenciais: a *Depthwise Convolution* e a *Pointwise Convolution*.

A *Depthwise Convolution* é responsável por aplicar um filtro a cada canal de entrada do modelo. Por sua vez, a *Pointwise Convolution*, uma convolução 1x1,

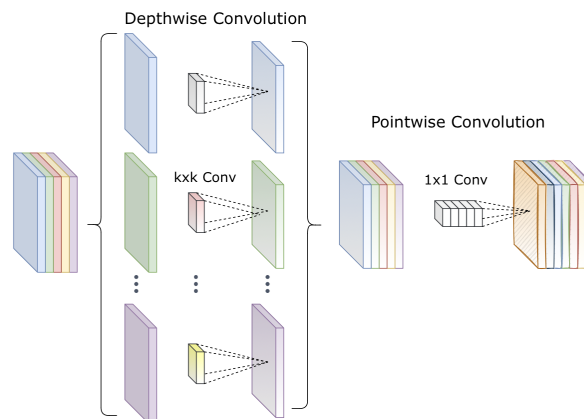


Figura 3.3: *Depthwise Seperable Convolution* [25]

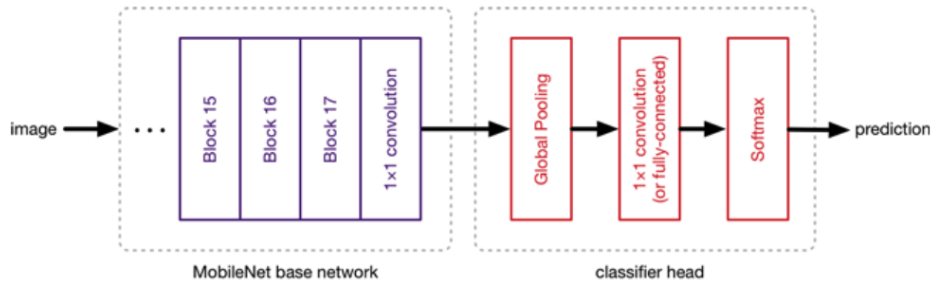


Figura 3.4: Arquitetura da MobileNetSSDv2 [27]

recebe os valores resultantes da camada anterior e agrega-os numa combinação linear para posterior geração de novas *features*, como representado na Figura 3.3 [26].

Desde então, foram lançadas novas versões desta rede. Com foco na detecção de objetos, a MobileNetV2 foi combinada com um *Single Shot Detector* (SSD), formando a MobileNetSSDv2. Este modelo, dispendo de 267 *layers* e 15 milhões de parâmetros, infere em tempo-real em dispositivos computacionalmente limitados, como é o caso de *smartphones* [27]. Aquando o processamento de uma imagem, a MobileNetV2 é responsável pela extração das suas *features*, enquanto que o SSD utiliza as mesmas para a classificação dos objetos de interesse (Figura 3.4).

3.1.3 *You Only Look Once* (YOLO)

Apresentado pela primeira vez em 2016, o YOLO representa um grupo de modelos de *Computer Vision* desenvolvidos por diferentes autores. O modelo original foi escrito em Darknet, uma *framework open source* baseada em C e CUDA que suporta computação em *Central Processing Unit* (CPU) e *Graphics Processing Unit* (GPU) [28].

À semelhança do MobileNetSSDv2, a arquitetura do YOLO é *Single Stage Detector*, uma vez que lida com a identificação de objetos e a sua classificação num único componente. No entanto, é geralmente mais rápido e preciso que o MobileNet.

Lançado em 2018, o YOLOv3 utiliza a Darknet-53 como *backbone*. Para além do mais, o modelo apresenta uma *objectness score* para cada *bounding box*, indicando a probabilidade de um objeto estar presente na *box*, independentemente da sua classe. Ainda de forma a aumentar o desempenho na detecção de objetos mais pequenos, esta versão executa previsões em 3 escalas distintas. A Figura 3.5 mostra um gráfico relativo ao rácio inferência/precisão de alguns detetores de objetos quando testados com o COCO *dataset*, onde o YOLOv3 se apresenta como o mais rápido e mais preciso [29].

Desenvolvido pela Ultralytics em 2020, o YOLOv5 é a extensão do YOLOv3 baseado não em Darknet mas em PyTorch, sendo conhecido pela sua velocidade e grande precisão na detecção de objetos. O modelo recorre a uma *Cross Stage Partial*

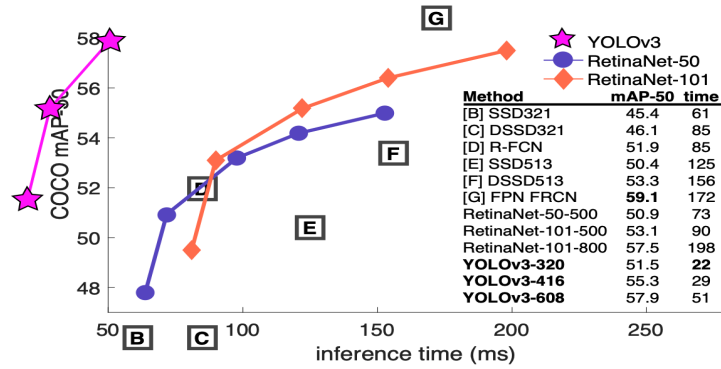


Figura 3.5: Performance do YOLOv3 [29]

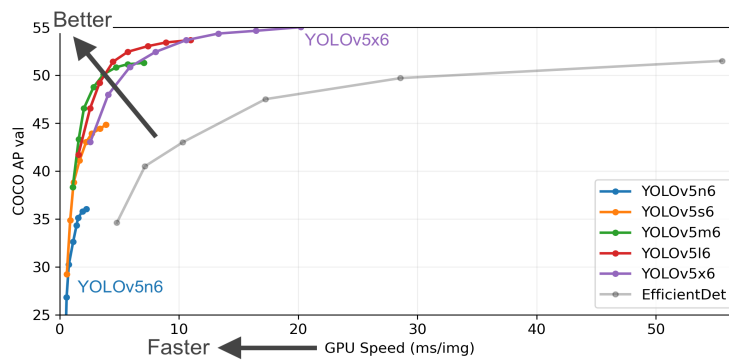


Figura 3.6: Performance do YOLOv5 [30]

Network (CSPNet) como *backbone* para formular as *features* das imagens [30]. A Figura 3.6 expõe a *performance* das diferentes variantes do YOLOv5.

A mais recente iteração da Ultralytics traduziu-se no YOLOv8. A arquitetura desta versão é *anchor free*, pelo que o algoritmo executa a previsão diretamente no centro do objeto, ao invés dos modelos anteriores que se baseavam no *offset* da *anchor box*. O gráfico da Figura 3.7 representa uma comparação da latência do YOLOv8 com a de modelos anteriores [31].

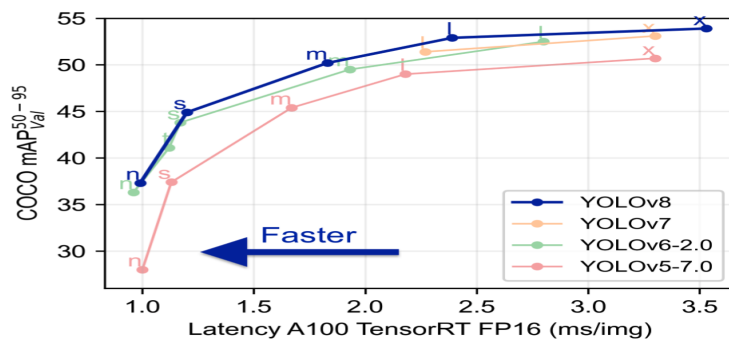


Figura 3.7: Latência do YOLOv8 [31]

3.2 Métodos de *Deep Learning* aplicados em AUVs

Aliado ao crescente leque de possíveis aplicações de um AUV, o *Deep Learning* tem sido empregue em vários projetos de investigação de índole marítima. Pese embora a deteção de *Ghost Nets* ainda não ser comum, existem diversos projetos de deteção de objetos submersos ou de organismos marinhos.

Para além do mais, é de notar que estas aplicações são igualmente submetidas às limitações impostas pelo oceano. É o caso de fraca visibilidade, gerada devido a más condições de luminosidade ou da elevada sustentação de partículas encontradas na água.

3.2.1 *Manipulation Purpose Underwater Agent Vehicle for Ghost Net Recovery Mission*

Apresentado em 2019 no IEEE/RSJ *International Conference on Intelligent Robots and Systems* (IROS), o projeto procura fornecer uma solução viável à localização e recuperação de *Ghost Nets* [32].

A operação de recuperação implica a utilização de um braço robótico no AUV. A atuação do braço altera o centro de massa do veículo, pelo que seria necessário um controlador de estabilidade adequado. Como alternativa, os autores focaram o seu trabalho no desenvolvimento de um sistema de três AUVs, onde o *Main* é responsável pela monitorização dos restantes *Agents*. Estando um veículo representado na Figura 3.8, os *Agents* procedem à recuperação da rede e recebem a sua orientação no espaço a partir do *Main*. Dispõem ainda de uma NVIDIA Jetson-TX2 como módulo principal de computação, caracterizada por um processamento de imagem de alta *performance*.

Para a deteção das *Ghost Nets*, foi utilizado o YOLO v3-Tiny. O *retraining* do mesmo contou com um *dataset* de 800 imagens, das quais 500 foram retiradas do Google e as restantes 300 consistiam em redes de pesca fotografadas pelos autores dentro e fora de um tanque de água. A Figura 3.9 retrata um teste realizado após o treino do modelo, onde se verifica a deteção da GN dentro e fora de água.

Por sua vez, a Figura 3.10 mostra o teste de captura da rede. Uma vez que as pontas da garra são laranja, o veículo assume a captura se a garra estiver fechada e se não for possível visualizar as pontas. Na eventual falta de uma destas condições, o AUV determina a falha do procedimento.

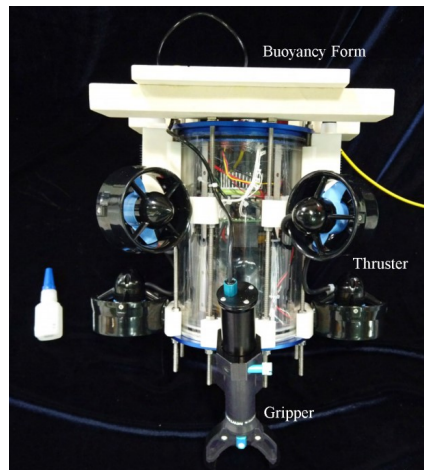


Figura 3.8: *Agent* AUV [32]



Figura 3.9: Detecção da *Ghost Net* em tempo real com YOLO v3-Tiny [32]

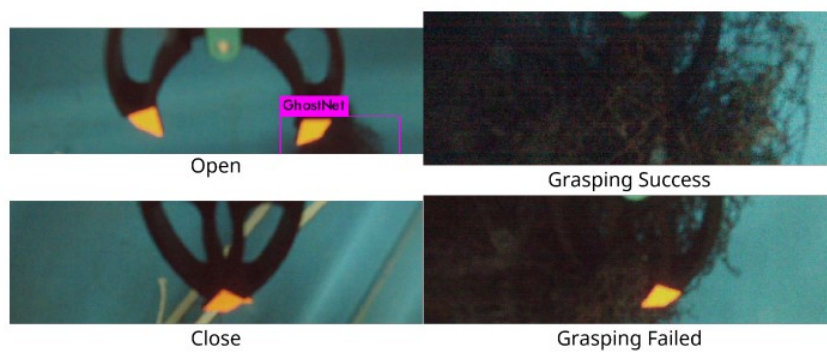


Figura 3.10: Determinação da captura da rede [32]

3.2.2 Deep Generative Network and Regression Network for Fishing Nets Detection in Real-time

Para além do risco que as *Ghost Nets* representam à vida marinha, tem-se registado um aumento no número de danos causados a embarcações. As artes de pesca ficam presas nas pás dos motores, o que incapacita o navio e gera prejuízos avultados. Por este motivo, os autores do projeto, apresentado no *Chinese Control Conference* (CCC) 2018, procuram uma solução para a deteção das redes submersas [33].

A refração da luz na água impossibilita a deteção de GNs por métodos de visão computacionais tradicionais. A utilização de um detetor de *edges* pode mostrar baixa precisão perante uma rede de pesca danificada. Por outro lado, o artigo defende que a utilização de *Deep Learning* resulta num *overfit* do modelo como consequência do pequeno *dataset* utilizado. O resultado obtido no treino com redes neuronais é tanto melhor quanto maior for o *dataset*, sendo que à data deste projeto, não havia nenhum *dataset* público específico para artes de pesca.

Por conseguinte, a solução passa pelo treino de 3 modelos distintos: Faster R-CNN, YOLO e SSD. Foi gerado um *dataset* de 2000 imagens obtidas por um sistema composto por uma câmara e um *laser*. O *laser* é insensível à temperatura e salinidade da água, pelo que elimina o ruído na fusão com a imagem obtida pela câmara. Ao *dataset* original, foram adicionadas 1000 *frames* novas, geradas a partir de uma *Deep Convolution Generative Adversarial Network* (DCGAN).

O treino dos modelos com o *dataset* aumentado mostrou melhores *mean Average Precisions* (mAPs), como demonstrado na Tabela 3.1. A Figura 3.11 ilustra o resultado do treino com DCGAN.

Tabela 3.1: Comparação dos resultados obtidos entre os 3 modelos treinados com o *dataset* e o aumentado [33]

Modelo (VGG16)	<i>Dataset</i> original	<i>Dataset</i> aumentado
Faster R-CNN	57,8%	63%
Tiny YOLO	54,7%	58%
SSD	62%	67%

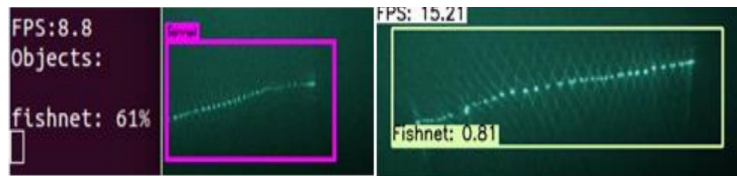


Figura 3.11: Resultado do treino com o *dataset* aumentado com DCGAN [33]

3.2.3 Real-time Detection Method of Marine Small Object with Underwater Robot Vision

A Aquacultura é também uma área na qual se verifica a utilização de *Deep Learning* em AUVs. A utilização destes veículos permite a localização e apanha de pequenas espécies marinhas, podendo substituir os mergulhadores. Apresentado no OCEANS 2018, o estudo incide no desenvolvimento de um AUV que recorre a técnicas de *Deep Learning* para este propósito [34].

Atendendo à fraca visibilidade muitas vezes encontrada no mar, os autores consideram que as técnicas tradicionais de deteção de objetos baseadas em *features* não produzem os melhores resultados, pelo que optaram por uma aproximação focada em *Deep Learning* com CNNs.

Para o efeito, foi criado um *dataset* de 28 mil imagens com 3 categorias, nomeadamente, ouriços-do-mar, pepinos-do-mar e escalopes. O mesmo foi utilizado para treinar dois modelos: um Faster R-CNN ZF e um Faster R-CNN VGG-16. Por sua vez, o treino foi executado num sistema com um Intel Core i7-6700K @ 4.00 GHz e uma NVIDIA GTX1080 Ti, sendo a mAP superior no VGG-16, como consta na Tabela 3.2.

Para além da deteção das diferentes espécies em cada *frame*, o objetivo do projeto passava por detetar o deslocamento destes organismos em tempo-real, pelo que foi implementado um *Kernelized Correlation Filter* (KFC) (ver Figura 3.12).

Tabela 3.2: Comparação entre Faster R-CNN ZF e VGG-16

Network	mAP
Faster R-CNN ZF	71,3%
Faster R-CNN VGG-16	79,6%

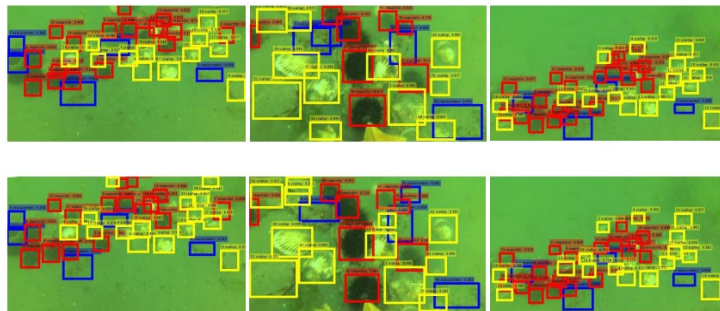


Figura 3.12: Resultados experimentais de deteção e seguimento de organismos marinhos [34]

3.2.4 *Perception and Recognition of Underwater Cable Based on Deep Learning*

Outra área de interesse para a utilização de AUVs consiste na detecção de cabos submarinos. O maior desafio para esta aplicação prende-se, novamente, às condições de visibilidade no mar, influenciando fortemente a qualidade das imagens obtidas. Os ambientes marinhos são caracterizados pela sustentação de partículas na água, seja areia, sedimentos ou mesmo plâncton. Para além destes fatores, uma fotografia subaquática apresenta baixo contraste, ruído elevado e ainda distorção de cor devido à atenuação da luz na água. Os autores deste artigo, apresentado na *International Conference on System Science and Engineering (ICSSE) 2020*, procuram então um método para melhorar a qualidade de um *dataset* e consequente treino de uma rede neuronal ao recorrer a um algoritmo de *Underwater Image Enhancement* [35].

O algoritmo tem em vista facilitar o processo de *labeling* do *dataset*, usando um conceito de fusão de pesos (*weight fusion*). O primeiro passo consiste na correção de cor, pelo que a partir da *frame* original, são geradas duas novas imagens nas quais são aplicados algoritmos de *white balance* e *auto level*. Posteriormente, é efetuada a fusão das duas imagens com base num determinado peso, resultando na Imagem C. A última fase trata-se da aplicação de um filtro bilateral por forma a eliminar o ruído presente na mesma. A Figura 3.13 descreve o fluxo do algoritmo.

O *dataset* de treino é composto por 876 imagens e o de teste por 219. O algoritmo de *enhancement* foi aplicado a ambos, sendo posteriormente anotados manualmente. Foram realizados treinos com o *dataset* original e com o melhorado em duas redes distintas, uma Faster R-CNN VGG16 e uma SSD VGG16. Os resultados obtidos nos treinos com o *dataset* melhorado mostram-se superiores àqueles do original, como se comprova na Tabela 3.3. A Figura 3.14 mostra uma comparação entre frames processadas e não-processadas pelo algoritmo.

Tabela 3.3: Resultados obtidos nos treinos

	Faster R-CNN		SSD	
	Original	Enhanced	Original	Enhanced
mAP	97,53%	98,75%	96,24%	98,20%
Test time	31,512 s	29,632 s	6,512 s	9,493 s

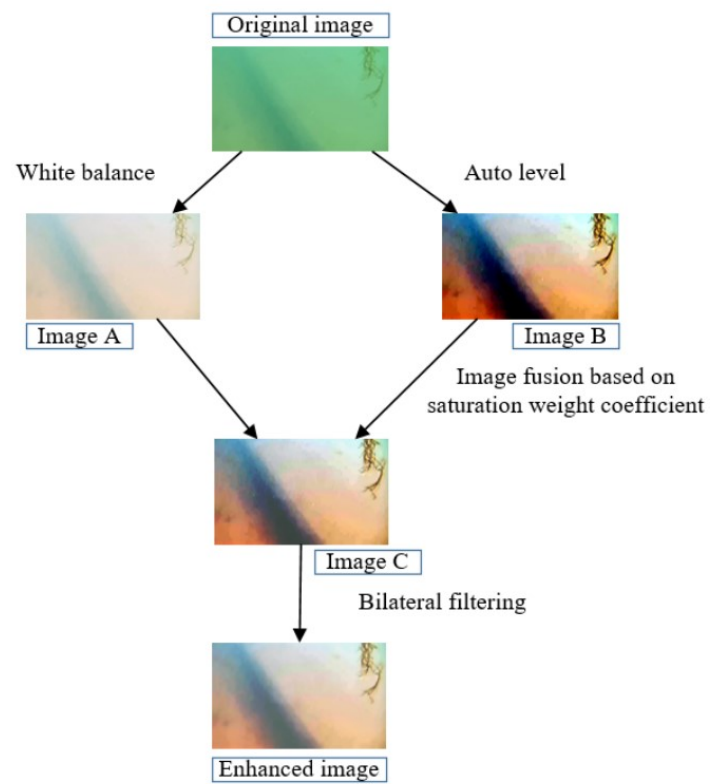


Figura 3.13: Diagrama ilustrativo do algoritmo de *Underwater Image Enhancement* [35]

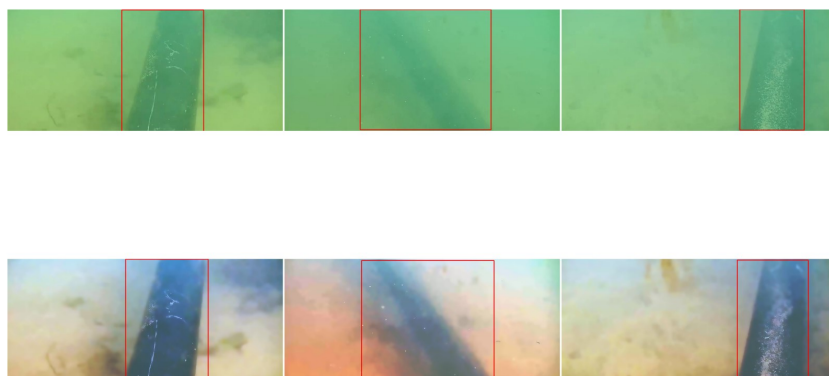


Figura 3.14: Imagem original e processada [35]

3.3 Outros métodos de detecção aplicados em AUVs

Pese embora o alcance considerável que uma câmara ótica possa dispor em águas limpas, o mesmo não se pode dizer para ambientes turvos. A título exemplificativo, podem ser considerados locais como portos e estuários. Nestes cenários, o Sonar apresenta uma clara vantagem, alguns inclusive capazes de gerar imagens de alta resolução a grandes distâncias, mesmo em visibilidade reduzida [36].

Porém, também se pode dar o caso da complexidade de um problema não justificar a implementação de algoritmos que impliquem elevados custos computacionais. É igualmente importante relembrar que o sucesso do *Deep Learning* é altamente dependente da quantidade de dados disponíveis. Por este motivo, esta secção aborda estudos de detecção visual subaquática baseados em métodos tradicionais de *Computer Vision*.

3.3.1 *What's mine is yours: Pretrained CNNs for limited training sonar ATR*

Um fator crucial na guerra submarina passa pela detecção de minas. Embora o papel dos Sonares seja fundamental neste tópico, não existem *datasets* públicos com dados suficientes que permitam o desenvolvimento de algoritmos eficazes de *Automatic Target Recognition* (ATR).

Deste modo, os autores acreditam que o *Transfer Learning* consiste na solução ideal para este problema, procurando comprovar os excelentes resultados que as CNNs possam apresentar quando combinadas com *Support Vector Machines* (SVMs) [37].

Deste modo, foram inicialmente retreinados 4 modelos diferentes, nomeadamente: um *Alexnet* combinado com SVM, um *Bag-of-Words* (BOW), um *Sparse Reconstruction-based Classifier* (SRC) e uma CNN reconstruída a partir das 3 primeiras camadas de uma *Visual Geometry Group* (VGG). Os resultados encontram-se descritos na Tabela 3.4 e ilustrados na Figura 3.15. A CNN SVM descreve claramente as melhores métricas, sendo uma diferença considerável quando comparada à CNN padrão.

Tabela 3.4: Métricas obtidas com os modelos treinados com dados de Sonar [37]

	CNN SVM	BOW	SRC	CNN
Precisão	98,8%	88,9%	78,3%	74,8%
<i>Recall</i>	98,8%	88,2%	77,5%	70,5%

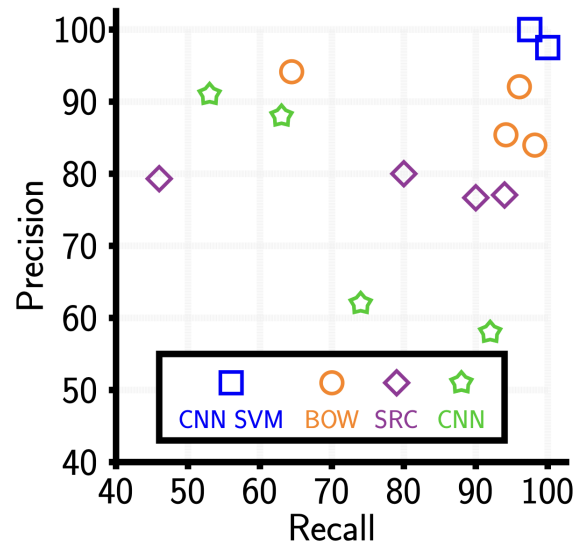


Figura 3.15: Gráfico das métricas dos modelos de Sonar [37]

3.3.2 *An adaptive method of damage detection for fishing nets based on image processing technology*

Uma vez que os danos sofridos nas armadilhas de aquacultura representam um avultado risco financeiro, existe a necessidade de executar inspeções regulares às mesmas. Assim, este *paper* propõe um método de inspeção visual que recorre a técnicas tradicionais de *Computer Vision*, sendo capaz de avaliar a dimensão do dano presente na arte de pesca e a sua localização na mesma [38].

De modo a ser o mais eficiente possível, o algoritmo baseia-se nos nós da rede, caracterizados pelas interseções dos fios. Cada interseção é identificada por 4 pontos de inflexão, detetados por um *Harris Corner Detector*. Na ausência de dano, os pontos de inflexão apresentam um padrão típico de uma rede. No entanto, se esses mesmos pontos identificados mostrarem uma forma semelhante a um círculo, poderá estar-se perante uma falha na rede (Figura 3.16).

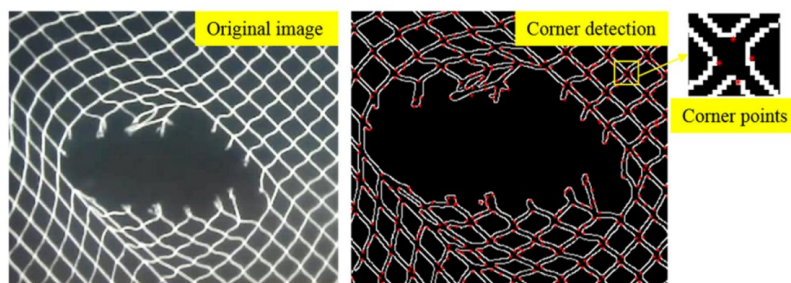


Figura 3.16: Detecção de pontos de inflexão numa rede [38]

3.3.3 Research on Recognition Method of Optical Detection Image of Underwater Robot for Submarine Cable

Foram fotografados tubos com 10 cm de diâmetro numa piscina para simular cabos submarinos. Estando a imagem original transformada para uma escala de tons de cinza, segue-se a sua segmentação, fazendo a distinção entre o cabo e o fundo da imagem. O processamento morfológico consiste no passo seguinte, cujo objetivo passa pela eliminação do ruído na fotografia, resultando numa imagem binária. A este nível, são aplicados algoritmos de *Edge Detection* para a delimitação do contorno do cabo [39]. A Figura 3.17 representa algumas das operações mencionadas.

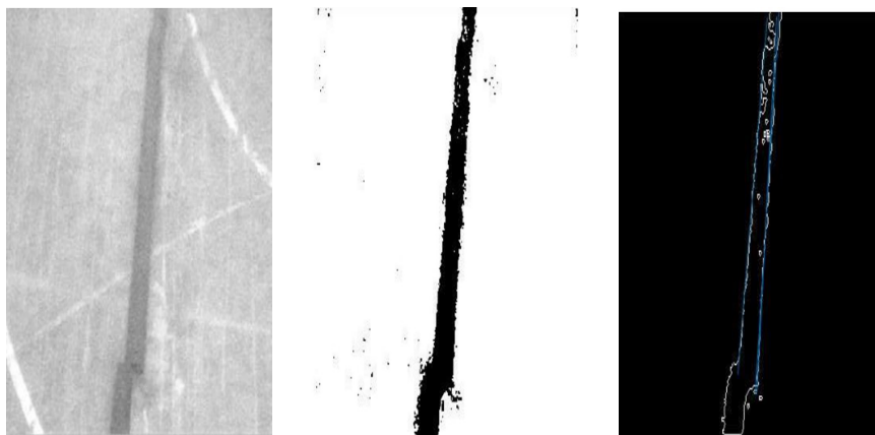


Figura 3.17: Imagem pré-processada à esquerda; imagem com segmentação ao centro; imagem com filtros de *edge detection* à direita [39]

3.4 Discussão do Estado da Arte

A análise efetuada ao Estado da Arte apresenta técnicas e algoritmos de visão para a deteção de objetos submersos, tendo sido igualmente registadas algumas aplicações dos mesmos em diferentes plataformas robóticas.

Endereçando os diferentes tipos de modelos de *Deep Learning* explorados, verifica-se que as CNNs são a base dos mesmos. Talvez a mais capaz, a Faster R-CNN é de igual modo a mais computacionalmente dispendiosa, sendo este um fator a considerar na integração num AUV. A MobileNetSSDv2 mostra-se como uma alternativa, tendo sido desenhada especialmente para dispositivos móveis. No entanto, os resultados obtidos poderão ser inferiores, pelo que é necessário considerar este *trade-off*. Por sua vez, as arquiteturas YOLO são mais recentes e aparentam bons resultados quando consideradas as suas velocidades de inferência.

Pese embora que apenas um dos veículos deste estudo tenha sido desenvolvido para a deteção de *Ghost Net*, é comprovada a possibilidade de detetar objetos submersos a partir de AUVs com recurso a *Deep Learning*. Os *papers* analisados

mostram ainda inferências em imagens captadas em águas turvas e de visibilidade reduzida, validando este tipo de cenários.

Apesar da enorme vantagem em ter um sistema baseado em Sonar que permite detecções a grandes alcances, é necessário um elevado conjunto de dados para a concretização de um modelo robusto. A obtenção deste tipo de dados é notoriamente mais complexa.

Por fim, os métodos tradicionais de *Computer Vision* descrevem bons resultados na detecção de artes de pesca, já que as mesmas são compostas por padrões visuais claramente distintos. No entanto, este tipo de modelo poderá deparar-se com problemas em ambientes turvos ou nos quais as redes estejam emaranhadas, sendo dificultada a classificação de *corners* ou *edges*.

Capítulo 4

Projeto

Após um estudo detalhado do *Ghost Fishing*, dos diferentes modelos de *Deep Learning* e das suas aplicações em veículos não-tripulados, procedeu-se a uma análise do problema e dos requisitos impostos para o projeto, tendo em vista o desenvolvimento de um sistema simulado que permita a navegação autónoma do IRIS com base na deteção de artes de pesca perdidas.

4.1 Domínio do problema

Como referido anteriormente, o projeto NetTag procura a mitigação do lixo gerado pelas atividades piscatórias por intermédio de uma componente de sensibilização e outra do espetro tecnológico. Por sua vez, a vertente tecnológica serve-se do IRIS, um AUV concebido para a deteção e recuperação de *Ghost Nets*.

4.1.1 *Ghost Fishing*

Ao revisitar o Capítulo 2, determina-se que a origem do problema se encontra no *Ghost Fishing*. São artes de pesca perdidas ou deliberadamente abandonadas que continuam a pescar e, inevitavelmente, a destruir o ecossistema marinho.

Estas redes apresentam diversas tipologias, cada qual com uma estrutura característica. Podem ser localizadas a diferentes profundidades e em diferentes relevos do solo marinho, o que pode elevar o grau de dificuldade para a sua deteção. Mais ainda, trata-se de uma operação que depende altamente da visibilidade encontrada no meio, muitas vezes agravada pelas condicionantes do oceano.

Além do mais, é comum encontrar animais encurralados nelas, como é o caso de peixes e caranguejos. No entanto, o estudo realizado registrou algumas aves e mamíferos.

4.1.2 Análise do IRIS

O IRIS (Figura 4.1) trata-se da plataforma a ser utilizada nesta dissertação. Desenvolvido para o NetTag, é um veículo pequeno, medindo apenas 80 *cm* de comprimento e pesando 25 *kg*. A sua unidade computacional remete para um sistema UPCore, tendo um Intel Atom como CPU e ainda 4 GB de memória. O seu controle apresenta 6 graus de liberdade graças aos seus 8 *thrusters*, sendo que:

- 4 *thrusters* traseiros controlam *Surge*, *Pitch* e *Yaw*;
- 2 *thrusters* laterais controlam *Sway*;
- 2 *thrusters* verticais controlam *Heave*.

Munido de duas câmaras, o veículo serve-se de uma câmara frontal para tarefas de inspeção e de monitorização, enquanto que emprega a câmara traseira apenas como um auxiliar à sua navegação.

O IRIS pode operar como ROV ou AUV. Quando em AUV, o submarino inicia a sua navegação com base em *pings* acústicos que o aproximam da *Ghost Net*. O veículo não é capaz de determinar a localização exata da rede visto que apenas dispõe de um *Modem* acústico que lhe fornece *range*. Assim, a partir de uma distância na qual se estima que a rede esteja visível, o veículo deve passar a navegação visual [9].

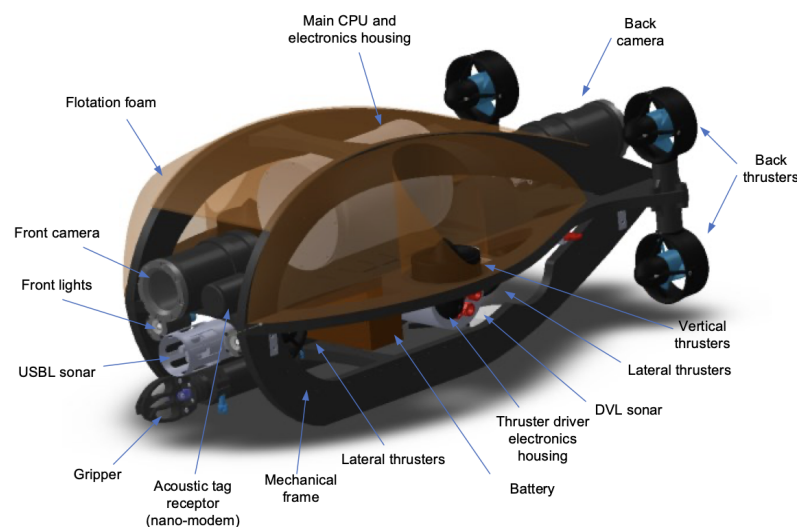


Figura 4.1: Modelo do IRIS [9]

4.2 Requisitos do Sistema

A integração direta de um algoritmo de navegação visual no IRIS acarreta um certo risco, uma vez que existe a probabilidade de se perder o veículo durante a missão. Como já analisado anteriormente, o meio no qual o AUV opera é caracterizado pelas condicionantes do oceano. Mais ainda, a própria aproximação a uma *Ghost Net* pode trazer complicações à navegação do submarino devido à tipologia da rede e do terreno.

Assim sendo, mostrou-se necessário o desenvolvimento de uma solução para a navegação visual em ambiente simulado. Esta opção permite uma validação completamente controlada e livre de riscos, tendo em vista uma futura implementação no veículo real. Portanto, o sistema deve cumprir os seguintes requisitos:

- O sistema deve apresentar um modelo do IRIS num ambiente simulado;
- O ambiente simulado do sistema deve incluir hidrodinâmica por forma a incorporar o melhor possível as características reais do oceano;
- O ambiente simulado do sistema deve incluir modelos tridimensionais de artes de pesca afundadas no solo marinho;
- O IRIS deve executar a deteção das *Ghost Nets* simuladas em diferentes condições de visibilidade;
- O IRIS deve executar uma manobra de aproximação a uma rede quando detetada;

4.2.1 Requisitos para o ambiente de simulação

A simulação do sistema deve considerar algumas características que são inerentes ao oceano, a fim de validar uma manobra de controlo que permita a posterior integração no veículo real.

Numa primeira instância, é importante escolher um simulador que seja capaz de lidar com hidrodinâmica. Isto permitirá expor o modelo do IRIS às condicionantes do meio, onde o mesmo terá de lidar com correntes marítimas, inércia nos movimentos, constante de flutuação, entre outros.

Para além do mais, o simulador deve ainda permitir diferentes condições de visibilidade. Estas são impostas pela hora do dia, sustentação de partículas na água e pelo fator de Jerlov, trantando-se o último de um sistema de classificação da absorção e dispersão da luz na água [40].

Por último, o meio simulado deve dispor de diferentes tipos de artes de pesca que servirão para a validação do algoritmo. Entre estes tipos, podem ser destacadas as armadilhas e as redes de emalhar.

4.2.2 Requisitos para a detecção de *Ghost Nets*

A detecção de *Ghost Nets* deve ser efetuada a partir da câmara frontal do IRIS em tempo-real. Para tal, as imagens recebidas devem ser processadas por um algoritmo de detecção de GNs que esteja implementado no veículo.

Uma vez que a inferência das imagens deve ser realizada no IRIS, torna-se necessária a consideração das limitações computacionais do veículo. Assim, os algoritmos a desenvolver devem ser relativamente leves, permitindo a integração no AUV e salvaguardando o melhor desempenho possível do mesmo.

Os algoritmos devem também ser capazes de detetar artes de pesca perdidas em condições de visibilidade adversas, tendo por base o intuito de simular a navegação visual em localizações tipicamente sujas, como é o caso de portos.

4.2.3 Requisitos para a manobra de aproximação à *Ghost Net*

Aquando da detecção de uma *Ghost Net*, o IRIS deve iniciar uma manobra de aproximação à mesma. De forma a simplificar o problema de controlo, são desconsiderados os cenários nos quais exista um obstáculo entre o veículo e a rede detetada.

O AUV deve executar a manobra de aproximação estando centrado com a *net*. A manobra é realizada até uma determinada distância de segurança, na qual o veículo deverá manter uma posição estacionária. Isto não só permite a posterior realização de uma inspeção visual à arte de pesca, como impede uma possível colisão com a rede ou o terreno.

4.3 Domínio da Solução

Dada por concluída a análise do problema e o respetivo estudo dos requisitos impostos, procedeu-se ao planeamento de uma solução para o sistema. Assim, esta secção apresenta respostas para a simulação a executar, para a detecção de GNs e para a manobra de controlo visual do IRIS.

4.3.1 Ambiente de simulação

Como descrito anteriormente, um simulador apresenta-se como uma ferramenta importante na validação de sistemas robóticos. Das possíveis opções, escolheu-se o Stonefish Simulator para servir esta dissertação, tendo ainda sido considerados o UWSim e o Gazebo.

O UWSim dispõe de um pacote de ROS embebido, o que permite implementar posteriormente o sistema simulado num AUV real, desde que este apresente uma arquitetura baseada no mesmo. No entanto, não é caracterizado por uma boa *performance* na simulação de hidrodinâmica. Por isso, é maioritariamente utilizado como uma ferramenta de visualização e de simulação de sensores subaquáticos.

No que concerne ao Gazebo, este é um dos simuladores mais conhecidos na robótica. Embora também apresente uma fácil integração com o ROS, o Gazebo não dispõe de um módulo nativo de hidrodinâmica, tendo que recorrer a *plugins* para o efeito. Além do mais, apresenta um *rendering* relativamente fraco na simulação subaquática.

Assim, segue-se a apresentação do Stonefish. Criado por uma equipa de investigadores da Universidade de Girona, trata-se de uma biblioteca de simulação desenvolvida em C++ direcionada a investigadores da área da robótica marinha, podendo ser igualmente utilizada na simulação de robótica terrestre ou aérea [41].

O Stonefish recorre a algoritmos implementados na biblioteca Bullet Physics para a computação da cinemática, dinâmica e colisão de objetos rígidos. Ao contrário dos outros simuladores, o mesmo incorpora ainda um módulo de hidrodinâmica, baseado na geometria dos objetos. Assim, o simulador fornece efeitos como a massa, a flutuabilidade e o atrito.

Relativamente ao *rendering*, o simulador emprega uma *pipeline* acoplada ao motor de física para esse efeito. O resultado traduz-se na alta qualidade obtida na simulação do céu e do oceano. O último dispõe também de efeitos visuais controlados pelos valores de *Jerlov* que afetam diretamente as condições de visibilidade na água, como se pode verificar na Figura 4.2.

O Stonefish habilita a integração da simulação com ROS ao integrar o *package* Stonefish ROS [42].

Estas características vão ao encontro dos requisitos propostos para o ambiente de simulação, resultando na escolha do Stonefish para este projeto.

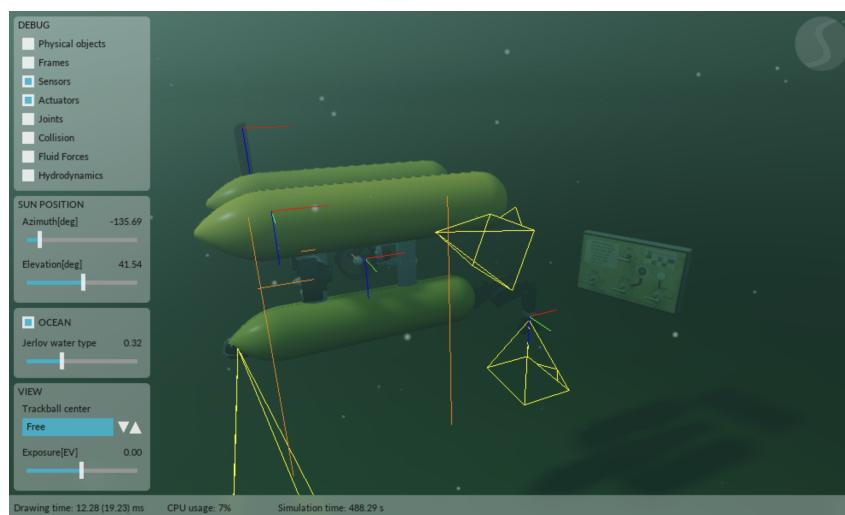


Figura 4.2: Modelo do AUV Girona 500 no Stonefish [41]

4.3.2 Detecção visual de *Ghost Nets*

Após a análise do problema e de algumas soluções existentes, optou-se pela implementação de algoritmos de *Deep Learning* para levar a cabo a deteção visual de *Ghost Nets*. Pese embora as redes de pesca apresentem um padrão característico, facilmente detetado por métodos tradicionais de *Computer Vision* (como é o caso do *Canny Edge Detector*), o *Deep Learning* mostrou ser um desafio mais interessante e capaz de produzir resultados superiores em condições de visibilidade adversas. Uma solução baseada em *Computer Vision* tradicional deparar-se-ia com alguns problemas perante, por exemplo, uma malha não visível, uma *Ghost Net* emaranhada ou mais afastada.

Existe a necessidade de um *dataset* relativamente grande para o treino dos modelos, assim como uma *framework* que permita o seu treino, monitorização e avaliação. Posteriormente, o modelo deve ser exportado e implementado no sistema da IRIS com o fim de inferir diretamente na imagem obtida pela câmara em tempo-real.

Características do *dataset*

Os artigos estudados sobre técnicas de *Deep Learning* foram baseados em *datasets* com imagens de alta qualidade, obtidas em águas com boa visibilidade ou até mesmo em terra. Este fator levanta um desafio para a inferência do modelo num ambiente turvo, com pouca visibilidade ou fracas condições luminosas.

Portanto, o *dataset* deve abranger diferentes cenários e não só dispor de imagens de alto contraste, nas quais as GNs sejam claramente visíveis, mas também de imagens obtidas em fracas condições de luminosidade, com sustentação de partículas e turbidez.

Escolha dos algoritmos de *Deep Learning*

Não é possível afirmar qual seria o melhor modelo de *Deep Learning* para a deteção das artes de pesca. São diversas as variáveis em cena, desde a própria qualidade do *dataset* como os hiperparâmetros do modelo. Portanto, a dissertação procura experimentar diferentes algoritmos e proceder ao seu ajuste, a fim de obter os melhores resultados possíveis.

Como ponto de partida, optou-se pelo treino de YOLOv5 e YOLOv8. A velocidade que estas arquiteturas apresentam, aliada à sua precisão, tornam-os interessantes para a implementação no IRIS. Outro ponto favorável à implementação remete para o custo computacional inferior quando comparados com, por exemplo, a uma Faster R-CNN.

Existem diversas variantes destes modelos, caracterizadas pelo seu tamanho, velocidades de inferência e mAP. Em suma, a mAP é uma métrica que retorna a percentagem da *Intersection over Union* (IoU) entre as *bounding boxes* verdadeiras

com as estimadas [43]. Logo, a mAP 50-95 remete para uma interseção das *boxes* entre 50% a 95%.

A Tabela 4.1 discrimina as versões *nano*, *small* e *medium* dos dois modelos. Sendo estas as versões mais pequenas dos modelos, são as que exigem menor poder computacional e, por isso, as mais interessantes para a futura implementação no IRIS. O YOLOv8 apresenta melhores mAPs em todas as suas versões, especialmente na *nano*. Contudo, é consideravelmente mais lento, algo que seria expectável para este tipo de precisão.

Tabela 4.1: Características dos YOLOv5 e YOLOv8 [31, 44]

Modelo	Inferência	Precisão (mAP 50-95)
YOLOv5 nano	45 <i>ms</i>	28,0%
YOLOv5 small	98 <i>ms</i>	37,4%
YOLOv5 medium	224 <i>ms</i>	45,4%
YOLOv8 nano	80,4 <i>ms</i>	37,3%
YOLOv8 small	128,4 <i>ms</i>	44,9%
YOLOv8 medium	234,7 <i>ms</i>	50,2%

4.3.3 Navegação e Controlo por Detecção Visual

A um determinado *range* da rede, a IRIS deve interromper a navegação por *tag* acústica e passar a navegar com base na informação visual recebida.

A manobra de controlo consiste no alinhamento do veículo com a rede, procurando mantê-la centrada na imagem. É também esperado que o AUV se aproxime da *Ghost Net* até uma determinada distância. Assim, segundo o sistema de coordenadas do veículo, são considerados os controlos em *Yaw*, *Heave* e *Surge*. Os controlos em *Yaw* e *Heave* são responsáveis por centrar a IRIS com a rede, enquanto que o controlador de *Surge* atua na aproximação. Deste modo, perante a deteção com sucesso de uma GN, foram consideradas duas estratégias para o controlo da IRIS.

A primeira procura centrar o veículo com a rede ao determinar os *center points* da imagem e da *bounding box*. Considerando o referencial da imagem, encontrar estes pontos envolve cálculos relativamente simples. O ponto central da *frame* encontra-se a meio do comprimento e da altura da imagem (ver Equação 4.1), enquanto que o ponto central da *bounding box* é dado pela divisão por 2 da soma da coordenada mínima com a coordenada máxima para cada eixo (Equação 4.2). O vetor resultante entre estes pontos (retratado na Equação 4.3) gera um *input* para os motores do veículo. Realça-se que a componente *x* da imagem traduz-se no *input* para o controlo lateral do veículo em *Yaw*. Por sua vez, a componente em *y* controla o *Heave*.

$$(x_{cf}, y_{cf}) = \left(\frac{img_{width}}{2}, \frac{img_{height}}{2} \right) \quad (4.1)$$

$$(x_{c_b}, y_{c_b}) = \left(\frac{x_{b_{min}} + x_{b_{max}}}{2}, \frac{y_{b_{min}} + y_{b_{max}}}{2} \right) \quad (4.2)$$

$$\overrightarrow{c_b c_f} = (x_{c_f} - x_{c_b}, y_{c_f} - y_{c_b}) \quad (4.3)$$

A segunda estratégia visa a aproximação do robot à *Ghost Net* até uma determinada distância de segurança. A ideia baseia-se na *bounding box*. Essencialmente, com a aproximação do veículo à rede, a sua dimensão aumenta. Assim, são calculados os perímetros da caixa de deteção e da imagem (sendo este último fixo, configurado pela resolução da câmara no *Stonefish*). O *input* para os motores é dado pelo erro entre três quintos do perímetro da imagem e o perímetro da *bounding box* (ver Equação 4.4), *input* esse que controla o *Surge*. Em suma, a IRIS deve fazer uma aproximação à rede até que o tamanho da *box* seja equivalente a três quintos do tamanho da imagem.

$$e_{surge} = \left(\frac{3}{5} \right) p_{img} - p_b \quad (4.4)$$

4.3.4 Arquitetura do *Software*

Estando definidas as etapas do projeto, passa-se à descrição da sua arquitetura. Assim, a Figura 4.3 ilustra um diagrama de blocos da arquitetura do *software* desenhada para a execução da navegação visual simulada. Para além de um bloco relativo ao *Stonefish*, o diagrama superior mostra 3 pacotes de ROS, nomeadamente, o IRIS *Stonefish*, o IRIS Control e o IRIS Vision.

O IRIS *Stonefish* trata-se do *package* principal da simulação, já que conta com um orquestrador responsável por lançar e gerir os restantes pacotes. Este interage diretamente com o *Stonefish* por intermédio do *Stonefish ROS*. Para além do mais, integra o cenário oceânico, o modelo do veículo e os modelos das *Ghost Nets*.

O IRIS Vision infere a imagem da câmara do veículo, recebida pelo IRIS *Stonefish*. Por sua vez, envia os dados da deteção para o IRIS Control, onde os mesmos são processados por forma a atuar nos motores do veículo.

Já o diagrama inferior remete para os componentes do sistema, mapeados aos respetivos *packages* pelo código de cores. O IRIS Launch é o *launch file* que funciona como orquestrador do sistema. O mesmo faz a gestão do *Ghost Net Inference* e do *Setpoints Visual Navigation*, os nós de ROS desenvolvidos para a inferência e controlo, respetivamente. Os valores calculados para os *thrusters* são enviados para o *Thruster Setpoints Pub*, sendo posteriormente convertidos e enviados para o modelo do veículo na simulação.

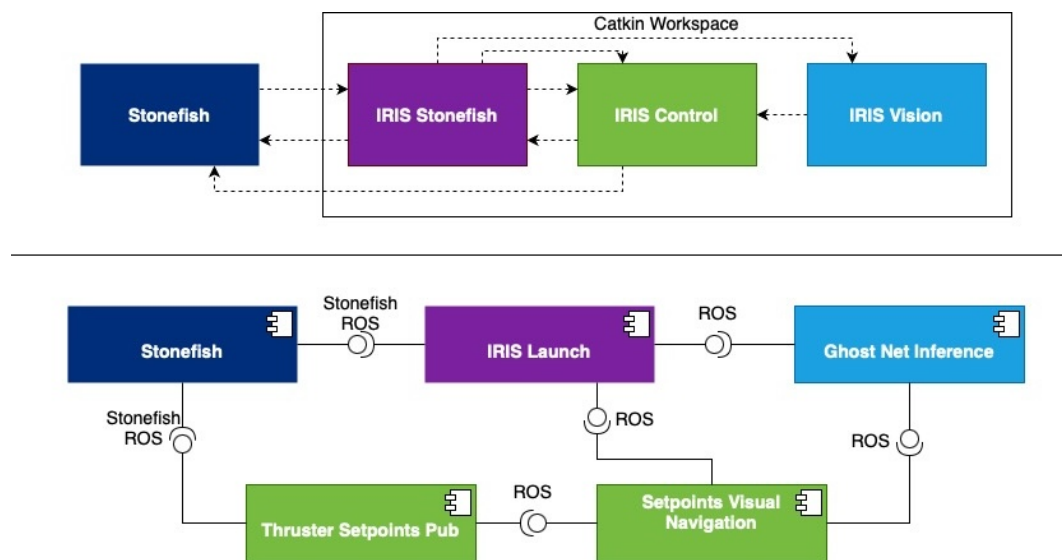


Figura 4.3: Diagrama de blocos da arquitetura

Capítulo 5

Implementação e Resultados

Este capítulo detalha a implementação da solução proposta, descrevendo a configuração do Stonefish, o treino dos modelos e a sua implementação no modelo simulado do AUV. É ainda explicado como é executado o controlo do veículo tendo por base a deteção visual de *Ghost Nets*.

5.1 Configuração da simulação

O processo de instalação do Stonefish é relativamente simples, estando bem descrito na documentação do simulador. Sendo que necessita de algumas bibliotecas previamente instaladas, este consiste em clonar o seu repositório para o sistema pretendido e, posteriormente, executar o *build* do mesmo com recurso a CMake. No entanto, o simulador requer uma GPU dedicada com o suporte da biblioteca OpenGL 4.3 por forma a executar simulações gráficas [41]. Por conseguinte, o sistema no qual a simulação foi desenvolvida é munido de um processador Intel Core i7-8565U e de uma Nvidia GeForce MX250.

A Figura 5.1 mostra o modelo do IRIS já renderizado no Stonefish. O mesmo foi criado no âmbito de outro projeto do LSA e reutilizado nesta dissertação.

A instalação do *package* Stonefish ROS fornece uma interface para a interação entre o simulador e o ROS. A mesma permite a utilização de cenários de simulação criados em ficheiros XML. Deste modo, foi criado o IRIS Stonefish, um pacote de ROS que utiliza esta interface e é responsável pela gestão da simulação.

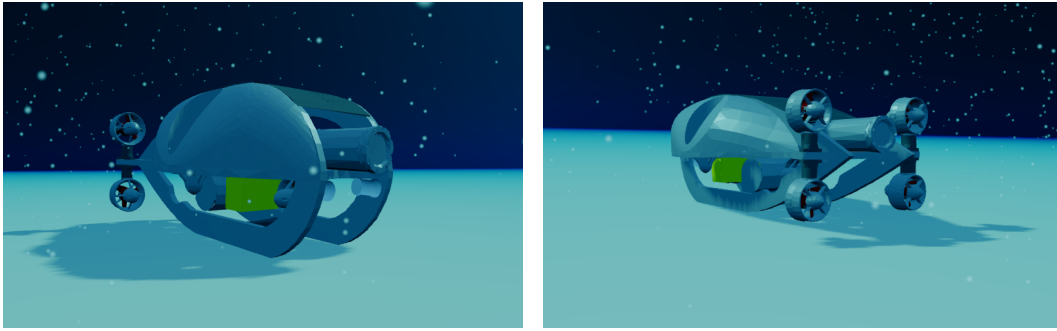
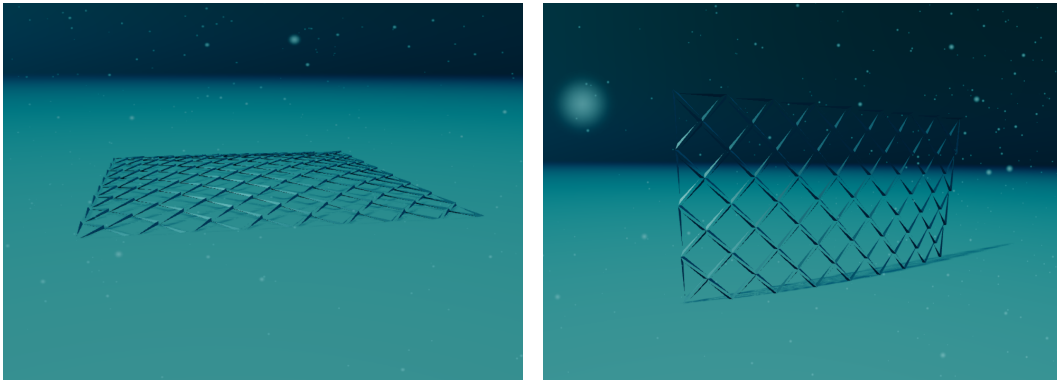


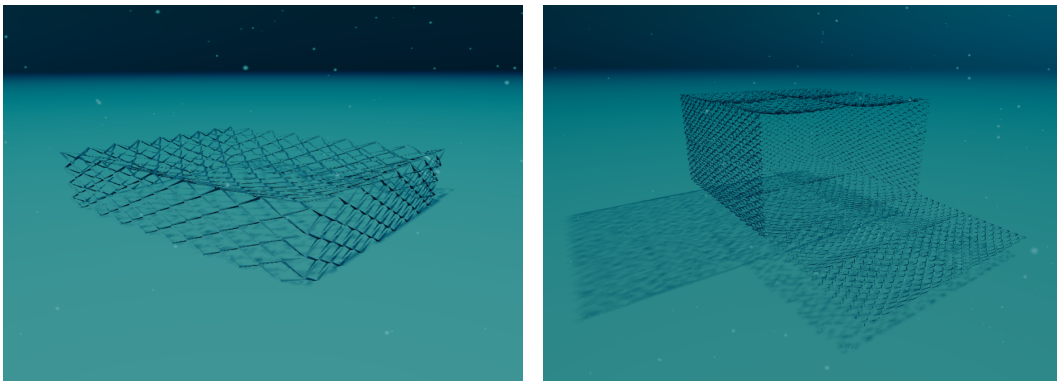
Figura 5.1: IRIS no Stonefish

Desenvolvido no Laboratório de Sistemas Autônomos, o IRIS Stonefish dispõe do orquestrador mencionado em 4.3.4. Este *launch file* faz o lançamento e gestão do cenário da simulação e dos diferentes nós de ROS a ela associados. O *package* contém ainda 4 modelos de artes de pesca criadas em Blender. Retratadas na Figura 5.2, as artes consistem em duas armadilhas, uma rede de emalhar e ainda uma malha caída no solo. Pese embora não sejam as GNs mais detalhadas, servem o propósito de validação do algoritmo.



(a) Malha caída no solo marinho

(b) Rede de Emalhar



(c) Armadilha fechada com malhagem grande

(d) Armadilha aberta com malhagem pequena

Figura 5.2: Modelos de artes de pesca criadas em Blender

5.2 *Dataset*

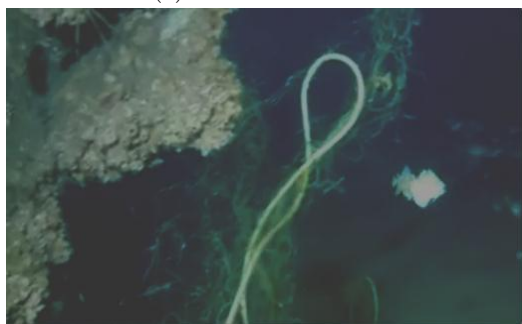
A fim de treinar os modelos e, atendendo aos requisitos impostos no Capítulo 4, foram utilizadas diversas fontes para a coleção do *dataset*, tendo em vista obter um conjunto de dados que fosse o mais abrangente possível. Procurou-se que a origem destas imagens (algumas retratadas na Figura 5.3) fosse fidedigna, na medida em que representasse a realidade do *Ghost Fishing*. A título exemplificativo, parte das mesmas são oriundas de missões da Ghost Diving, uma Organização Não-Governamental (ONG) focada na recuperação de artes de pesca perdidas ou abandonadas no mar [11]. Recorreu-se também a imagens da Healthy Seas, outra ONG que serve o mesmo propósito, e do Centro Interdisciplinar de Investigação Marinha e Ambiental (CIIMAR), sendo que este disponibilizou um vídeo do qual se extraíram alguns *frames*. Foram inclusivé extraídas imagens de vídeos publicados no Youtube por canais de



(a) Rede de Emalhar



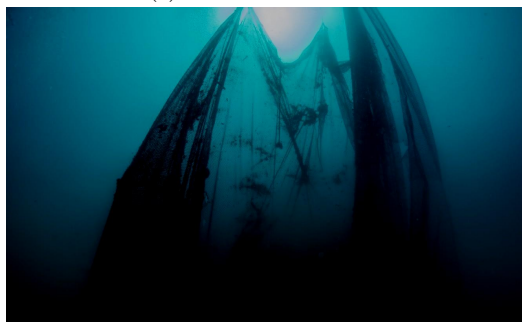
(b) Malha e cabo de uma rede



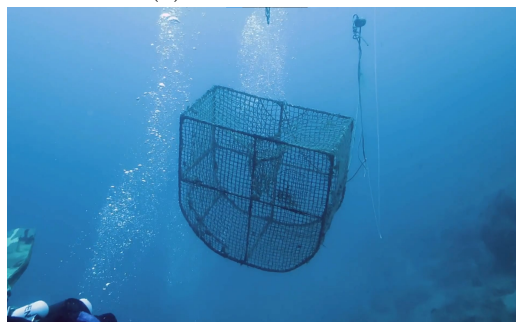
(c) Cabo de uma rede



(d) Rede emaranhada



(e) Recuperação de uma rede de Arrasto



(f) Recuperação de uma Armadilha

Figura 5.3: Imagens do *dataset*

sensibilização para a preservação do oceano.

A criação do *dataset* constou num processo iterativo com diferentes versões do mesmo, já que as imagens coletadas eram periodicamente analisadas a fim de obter o melhor conjunto possível. A versão mais recente conta com 2000 imagens que abrangem diferentes cenários subaquáticos. Ao invés do artigo publicado por Kim et al. [32], não foram consideradas fotografias de redes de pesca captadas fora do ambiente marinho. Esta decisão procura tornar o modelo mais robusto às condicionantes do meio. Para além do mais, a composição do *dataset* pretende englobar o maior tipo de artes de pesca possível em diferentes condições de visibilidade.

O *labelling* do *dataset*, realizado no Roboflow [45], baseou-se em 3 classes, nomeadamente, *ghost_net*, *fish* e *crab*. A primeira engloba os diferentes tipos de artes de pesca, sendo que não diferencia cabos e malhas. As restantes foram criadas já que, durante a coleção dos dados, foram encontradas diversas fotografias com *Ghost Nets* emaranhadas em torno de animais, sendo a maioria composta por peixes e caranguejos. Considerando o facto do IRIS ter sido desenhado para a deteção e recuperação de GNs, poderá ser útil a capacidade do veículo em detetar um animal emaranhado nelas. A manobra de recuperação certamente terá este fator em conta, tendo em vista a salvaguarda do animal. A Figura 5.4 ilustra estes cenários.

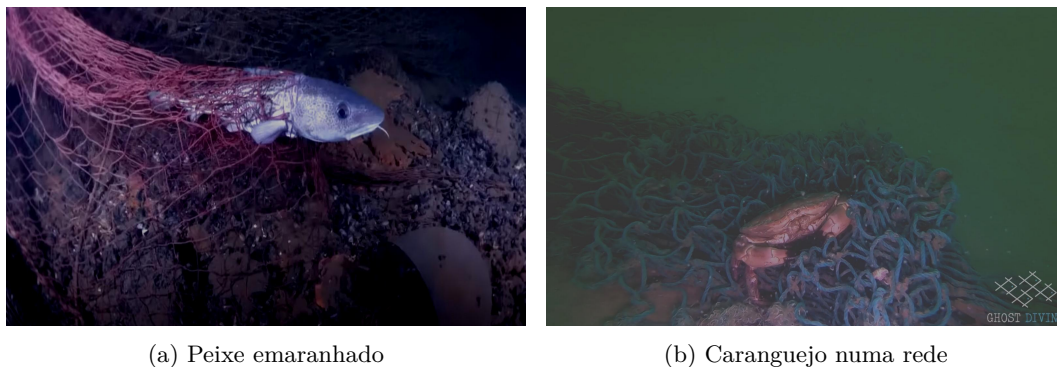


Figura 5.4: Imagens representativas das classes *fish* e *crab*

Foram implementadas *augmentations* no *dataset* por forma a aumentar o seu volume de dados. Assim, este passou a contar com 4800 imagens. Os *frames* adicionados procuram robustecer o modelo ao apresentarem determinadas características sob as quais o mesmo possa ser submetido aquando uma inferência. Como tal, as imagens *augmented* foram o resultado das transformações baseadas em rotação, desfocagem, adição de ruído e alteração do brilho. Por sua vez, estas transformações tentam reproduzir os efeitos obtidos pelo deslocamento do veículo, pela sustentação de partículas na água e por diferentes condições de luminosidade. Alguns exemplos podem ser encontrados na Figura 5.5.

Outra *augmentation* testada passou pelo *Tiling*, um processo que cria imagens

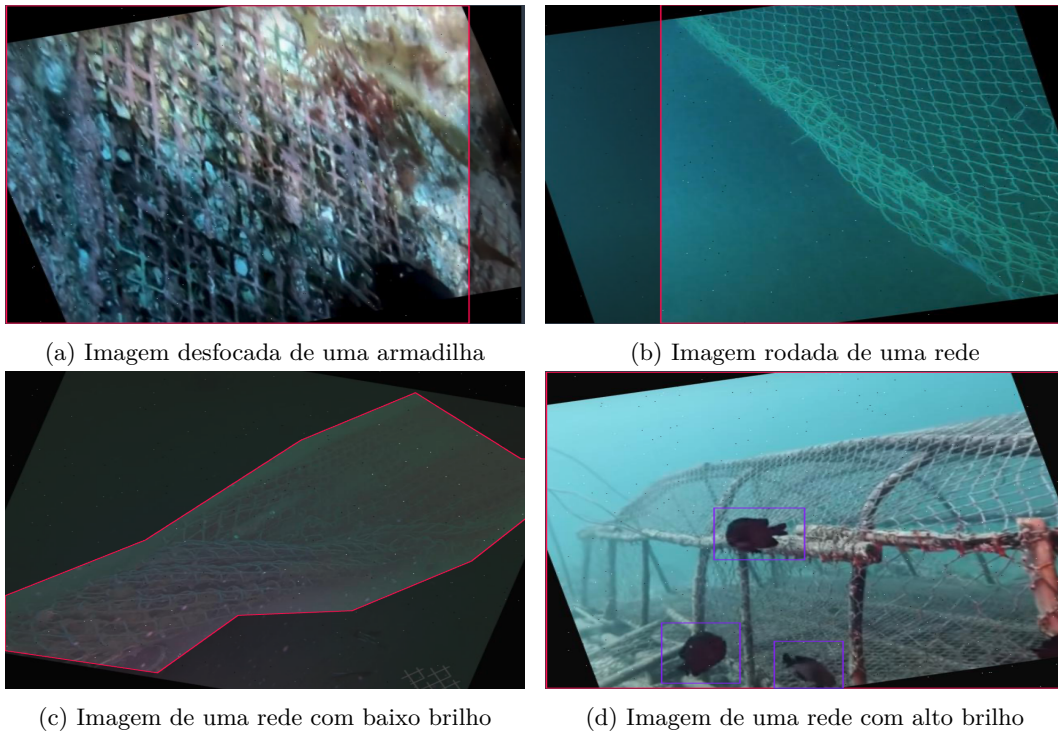


Figura 5.5: Imagens acrescentadas por *augmentation*

novas a partir das originais ao dividi-las e ampliá-las ao formato de entrada do modelo. É recorrente na detecção de objetos de pequenas dimensões e, atendendo às particularidades das *Ghost Nets*, uma possível vantagem na aprendizagem dos diferentes tipos de malhas. Apesar do *dataset* gerado ser composto por 19200 imagens, este foi descartado por mostrar fracas augmentações (exemplo na Figura 5.6).

A divisão do *dataset* contou com 70% das imagens para treino (*train set*), 15% para teste (*test set*) e 15% para validação (*eval set*). Estes valores foram escolhidos devido ao baixo volume de dados, pretendendo evitar o *overfit* do modelo e disponibilizar um maior número de dados quer para a afinação do mesmo, quer para a sua validação final.

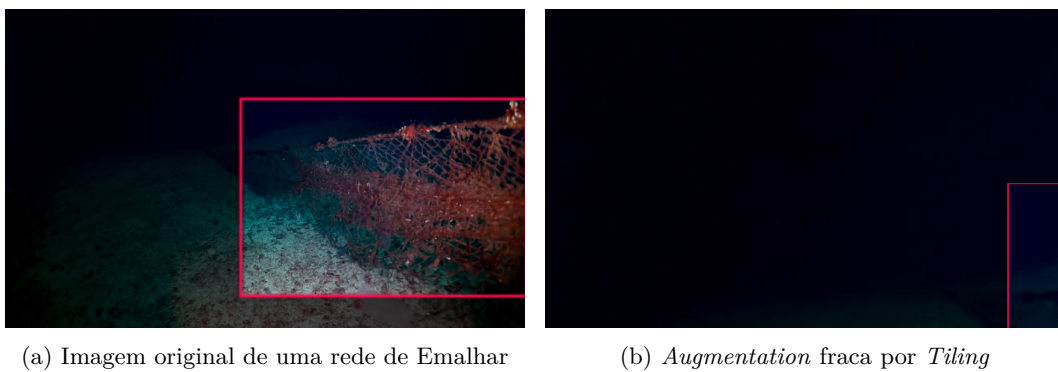


Figura 5.6: *Augmentation* por *Tiling*

5.3 Considerações prévias ao treino dos modelos

É relevante mencionar que os algoritmos de *Deep Learning* caracterizam-se pelo grande número de parâmetros resultantes dos treinos. Para além dos mesmos, cada rede neuronal dispõe de um conjunto de hiperparâmetros que devem ser afinados pelo utilizador [46].

Assim, um valor elevado de *learning rate* pode ser um motivo para o *overfit* do modelo. Este hiperparâmetro atua como um coeficiente nos pesos do modelo, sendo que, quanto maior for, maiores serão as atualizações dos pesos a cada *step*, podendo conduzi-lo à instabilidade e consequente *overfit* [47]. Portanto, o *learning rate* deve ser diminuído com o intuito de permitir atualizações mais pequenas e precisas aos pesos do modelo.

Outro ponto de interesse passa pela implementação de um *learning rate decay*, uma técnica que diminui gradualmente o *learning rate* do modelo ao longo do seu treino. Esta baseia-se num valor inicial mais elevado que acelera o processo de treino e evita o *spurious local minima* (uma zona da *loss function* que impede a progressão do modelo) [48]. O *learning rate decay* pode ser aplicado nos treinos a realizar por intermédio de *Cosine Annealing*, sendo este um método que utiliza uma função cosseno para a redução do hiperparâmetro.

O YOLOv5, por exemplo, utiliza por defeito o *Stochastic Gradient Descent* (SGD) como o algoritmo otimizador para a minimização da *loss function* e para a atualização dos parâmetros do modelo. Embora seja capaz de uma boa generalização, é relativamente lento e necessita de um vasto conjunto de dados. Uma alternativa passa pelo AdamW. É um otimizador mais rápido que requer um menor conjunto de dados. No entanto, é mais propício a *overfit* [49].

A regularização do modelo pode ser igualmente afinada por intermédio do *weight decay*. O mesmo visa a penalização dos maiores pesos do modelo, levando-o a concentrar-se nos menores e, consequentemente, a evitar o *overfit*. Em adição, o *momentum* pode ser também alterado com vista a evitar oscilações nas curvas de *loss*.

É, também, interessante experimentar diferentes valores de *batch size*. Embora um número mais elevado signifique que o modelo analise dados em maior quantidade, em determinadas circunstâncias, a redução deste parâmetro implica a melhoria na sua generalização [50].

5.4 Treino do YOLOv5

Atendendo às limitações computacionais do submarino, a empregabilidade de um modelo mais pesado poderá não ser viável. Assim, a dissertação focou no treino do YOLOv5 nano, realizado numa *workstation* munida de duas Nvidia GeForce RTX 2080 Ti. Foram realizados vários treinos afinados com diversos hiperparâmetros em busca de obter os melhores resultados possíveis. De forma sumária, esta secção descreve o processo registado em 3 treinos com diferentes *batch sizes*.

5.4.1 YOLOv5 nano com *batch size* 64

A Figura 5.7 retrata os resultados obtidos no primeiro treino. Com um *batch size* de 64 e 200 *epochs*, nenhum dos restantes hiperparâmetros do modelo pré-treinado foi alterado. Este treino serve de *benchmark* para os posteriores. Os gráficos de precisão e *recall* apresentam bons resultados, na ordem dos 75% e 70%, respetivamente. As mAPs mostram-se igualmente interessantes. Pese embora as perdas verificadas durante o treino sejam baixas, a análise ao gráfico *val/obj_loss* indica um possível *overfit* do modelo. Tal verifica-se pelo declive positivo que tem início entre as *epochs* 50 e 100. Eventualmente, as futuras inferências que o modelo venha a realizar podem ser caracterizadas por um número elevado de falsos positivos dado o enviesamento do mesmo perante os dados de treino.

A avaliação (ou *evaluation*) executada ao modelo gerou a Matriz de Confusão presente na Figura 5.8. As linhas da matriz representam as previsões do modelo, enquanto que as colunas remetem para o *ground truth*. Verifica-se, na interseção da classe, que apenas 63% das *Ghost Nets* foram corretamente classificadas. Por sua vez, a percentagem de Verdadeiros Positivos de peixes e caranguejos foi superior,

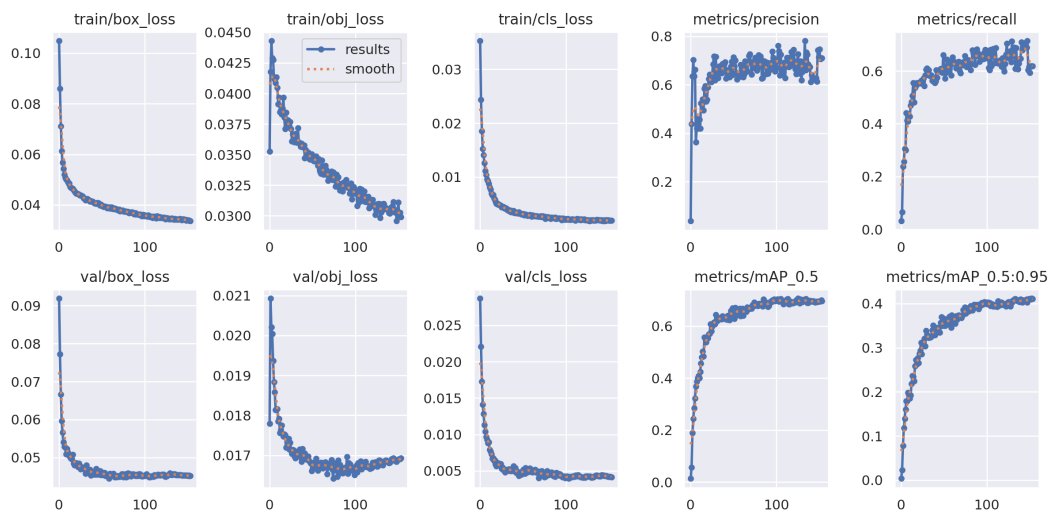


Figura 5.7: Primeiros resultados do YOLOv5 nano

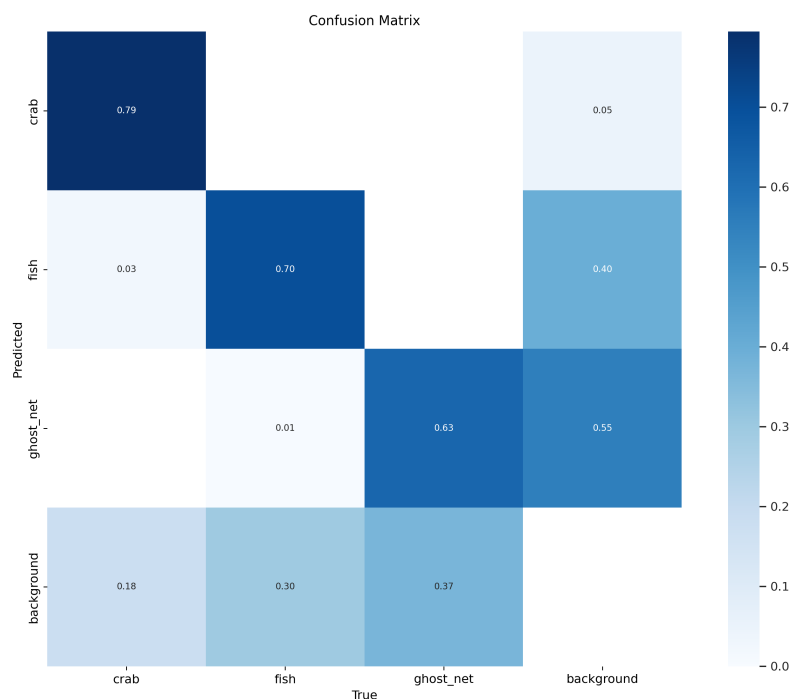


Figura 5.8: Matriz de Confusão do YOLOv5 nano

com 79% e 70%. Não obstante, o número de Falsos Positivos não é promissor, sendo que, na última coluna da matriz, verifica-se que foram estimadas 55% de *labels* de redes de pesca e 40% de peixes em dados de *background*, onde não existe nenhuma classe a ser detetada. Os números de Falsos Negativos, presentes na última linha da matriz, são igualmente altos.

A Figura 5.9 expõe as inferências efetuadas pelo modelo nas imagens do *dataset* de validação. As confianças obtidas variam fortemente, sendo que as mais razoáveis se encontram nos 70% e 80%. Também não são encontrados Falsos Positivos nestas imagens e é de notar ainda os bons *scores* na deteção de peixes e caranguejos. Ainda assim, destacam-se as confianças relativamente baixas em redes que são claramente visíveis, assim como a presença de Falsos Negativos. Outro ponto menos favorável passa pela sobreposição de múltiplas *bounding boxes* em determinadas fotografias.



Figura 5.9: Inferência do YOLOv5 nano no *dataset* de validação

5.4.2 YOLOv5 nano com *batch size* 32

Posto isto, o segundo treino aqui registado, e baseado nos hiperparâmetros discriminados na Tabela 5.1, obteve os resultados explícitos na Figura 5.10. Agora com um *batch size* de 32, a análise às perdas do modelo mostra valores ligeiramente superiores às do anterior. No entanto, o *overfit* do algoritmo já não é tão acentuado, o que indica que o modelo reagirá melhor a dados nunca antes vistos. Também é de realçar a rápida curva de aprendizagem observada nos gráficos das métricas.

Por sua vez, a Figura 5.11 ilustra a Matriz de Confusão deste modelo. Embora a percentagem de Verdadeiros Positivos seja inferior em todas as classes quando comparada ao modelo anterior, o número de Falsos Positivos e Falsos Negativos é de igual modo inferior, indicando a atenuação do *overfit*.

Os resultados da inferência no *dataset* de *evaluation* encontram-se na Figura 5.12. De uma forma geral, os valores de confiança aparentam ser inferiores aos obtidos no primeiro treino. Destacando a classe destinada às artes de pesca, o seu número de Falsos Negativos aumentou paralelamente à descida dos *scores* obtidos nos Verdadeiros Positivos. Não obstante, não se registam *bounding boxes* sobrepostas.

Tabela 5.1: Hiperparâmetros - com *batch size* de 32 - utilizados num treino do YOLOv5 nano

Hiperparâmetro	Valor
<i>Batch size</i>	32
<i>Epochs</i>	200
<i>Learning rate</i>	0,005
<i>Momentum</i>	0,95
<i>Weight decay</i>	0,005
<i>Optimizer</i>	SGD

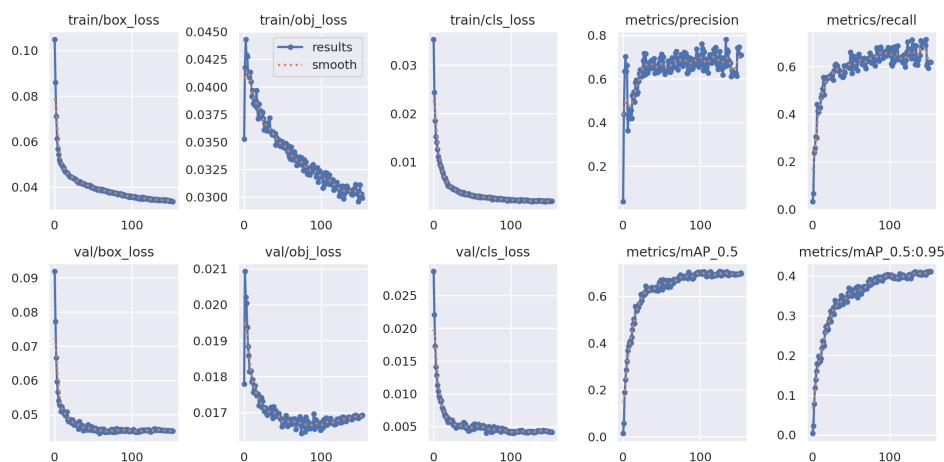


Figura 5.10: Resultado de um treino do YOLOv5 nano com um *batch size* de 32

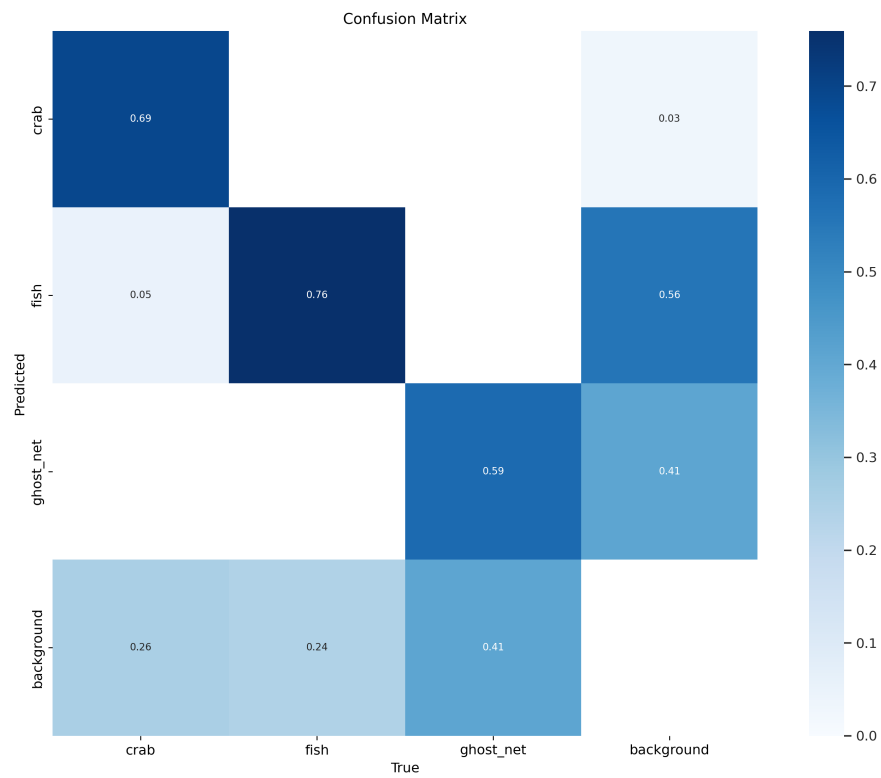


Figura 5.11: Matriz de confusão do YOLOv5 nano com um *batch size* de 32

O mesmo não pode ser dito para a classe dos caranguejos, onde se registam agora *labels* sobrepostas.



Figura 5.12: Inferência do YOLOv5 nano com um *batch size* de 32 no *dataset* de validação

5.4.3 YOLOv5 nano com *batch size* 128

Outro treino realizado recorreu aos hiperparâmetros descritos na Tabela 5.2. Optou-se pelo aumento do *batch size* de modo a explorar a capacidade do YOLO em aprender com um grande número de dados a cada *step* do treino. Além disso, o otimizador escolhido foi o AdamW.

A análise dos resultados (Figura 5.13) mostra perdas, durante o treino, inferiores às registadas anteriormente e também menos oscilatórias. No entanto, as perdas de validação caracterizam-se por um declive acentuado nas primeiras *epochs*. No gráfico relativo à *objectness loss*, verifica-se ainda uma ligeira subida ao longo do tempo, indicando um possível *overfit* do modelo.

A Matriz de Confusão deste treino, retratada na Figura 5.14, continua a expor a existência de Falsos Positivos e Falsos Negativos, representados na última coluna e na última linha, respetivamente. Todavia, e ao invés dos resultados anteriores, não apresenta Falsos Positivos entre as classes definidas para o treino, somente em *background*.

A análise posterior às inferências realizadas no *evaluation set* revelam alguns

Tabela 5.2: Hiperparâmetros - com *batch size* de 128 - utilizados num treino do YOLOv5 nano

Hiperparâmetro	Valor
<i>Batch size</i>	128
<i>Epochs</i>	200
<i>Learning rate</i>	0,0008
<i>Momentum</i>	0,937
<i>Weight decay</i>	0,002
<i>Optimizer</i>	AdamW

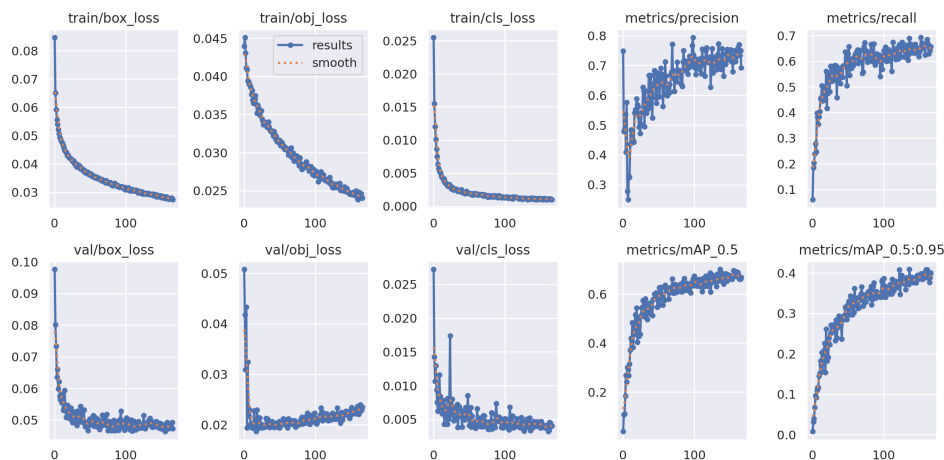


Figura 5.13: Resultado de um treino do YOLOv5 nano com um *batch size* de 128

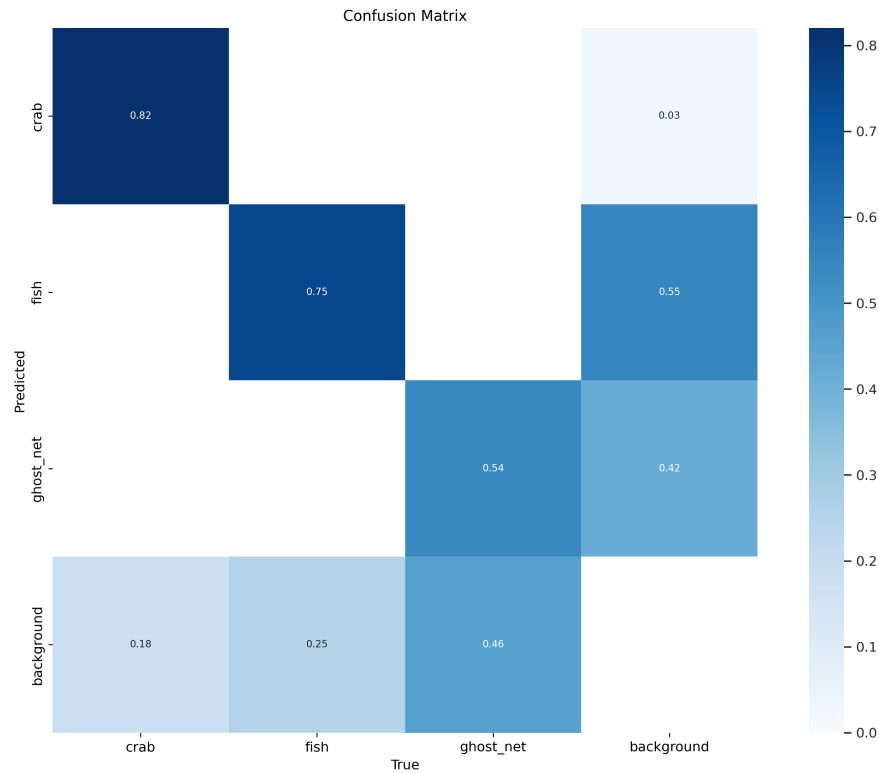


Figura 5.14: Matriz de confusão do YOLOv5 nano com um *batch size* de 128

scores superiores aos obtidos nos outros treinos. Visível na Figura 5.15, não existe sobreposição de *bounding boxes* da mesma classe. O modelo mostra-se capaz de detetar um peixe emaranhado numa *ghost net*, como ilustrado na imagem do canto superior esquerdo. Porém, a mesma mostra igualmente um Falso Positivo para a classe *fish*. A classe *crab* apresenta de igual forma uma melhoria, com confianças na ordem dos 70%. Contudo, são notórios os Falsos Negativos presentes nestas fotografias. A última linha de imagens serve como exemplo deste efeito. À exceção da penúltima fotografia, as *Ghost Nets* presentes não foram devidamente detetadas.



Figura 5.15: Inferência do YOLOv5 nano com um *batch size* de 128 no *dataset* de validação

5.5 Treino do YOLOv8

À semelhança do anterior, a dissertação incide no treino do YOLOv8 nano. As restantes versões do v8 apresentam um melhor desempenho, mas são menos interessantes do ponto de vista do custo computacional. Segue-se a descrição de 3 treinos distintos, novamente com 3 *batch sizes* diferentes e realizados na mesma *workstation* dos YOLOv5.

5.5.1 YOLOv8 nano com *batch size* 64

O primeiro treino efetuado contou com um *batch size* de 64 para 200 *epochs*, mantendo os restantes hiperparâmetros inalterados (Tabela 5.3). Os resultados do mesmo estão na Figura 5.19. Quando comparado com o primeiro treino do YOLOv5, destacam-se os valores mais elevados das perdas. Não obstante, são arquiteturas diferentes, pelo que este não é um fator indicativo de um algoritmo menos capaz.

Por sua vez, as métricas mostram bons resultados, estando a precisão situada entre os 70% a 80%, com o *recall* a ultrapassar os 65%. As mAPs são também

Tabela 5.3: Hiperparâmetros - com *batch size* de 64 - utilizados num treino do YOLOv8 nano

Hiperparâmetro	Valor
<i>Batch size</i>	64
<i>Epochs</i>	200
<i>Learning rate</i>	0,0008
<i>Momentum</i>	0,937
<i>Weight decay</i>	0,002
<i>Optimizer</i>	SGD

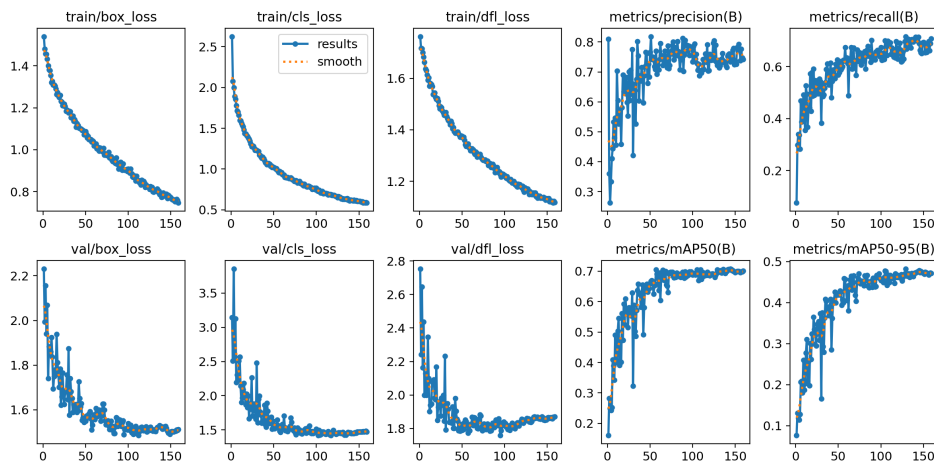


Figura 5.16: Resultados do YOLOv8 nano com um *batch size* de 64

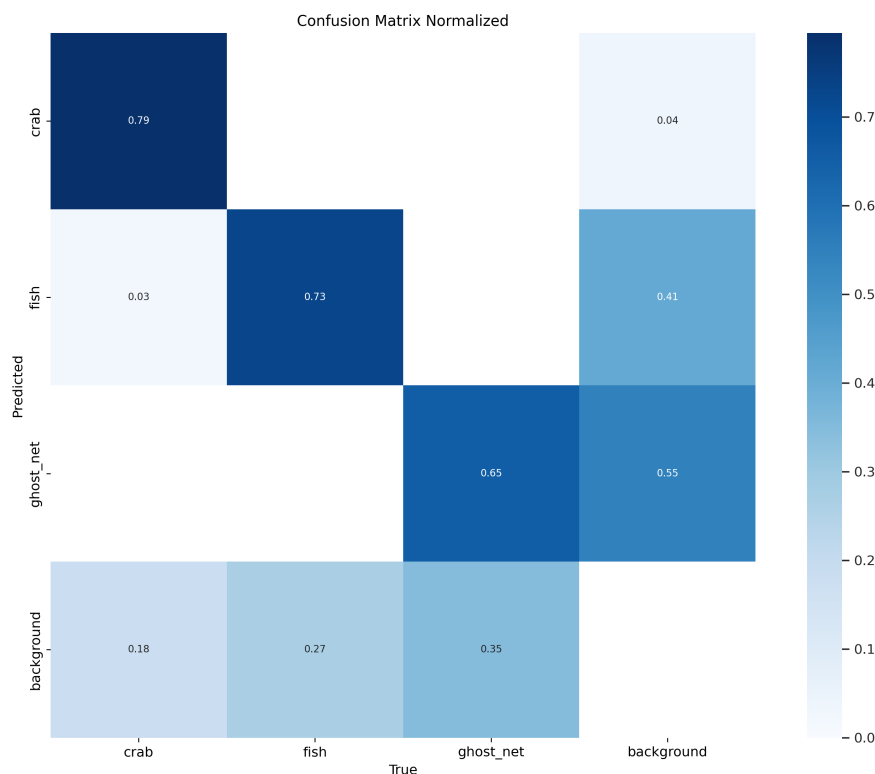


Figura 5.17: Matriz de confusão do YOLOv8 nano com um *batch size* de 64

caracterizadas por valores ligeiramente superiores aos obtidos no primeiro treino do YOLOv5.

Ainda que as percentagens de Verdadeiros Positivos sejam aceitáveis, especialmente a da classe de caranguejos, o modelo continua a gerar bastantes Falsos Positivos e Falsos Negativos, como se verifica pela Figura 5.17.

Não obstante, as inferências no *dataset* de avaliação (Figura 5.18) mostram confianças elevadas. O algoritmo consegue produzir deteções em todas as fotografias à exceção de uma que remete para uma rede quase transparente e, de um modo geral, as *bounding boxes* geradas ajustam-se aos objetos de interesse. Porém, algumas das imagens mostram sobreposição das caixas.

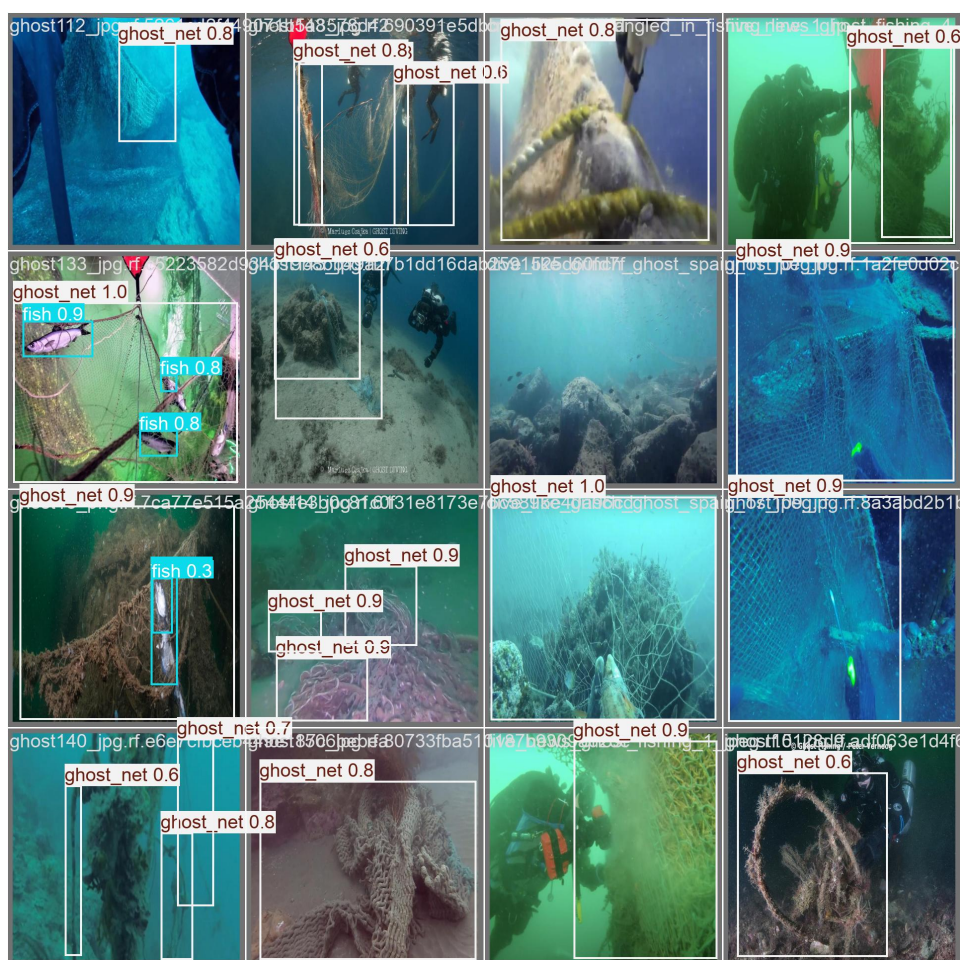


Figura 5.18: Inferência do YOLOv8 nano com um *batch size* de 64 no *dataset* de validação

5.5.2 YOLOv8 nano com *batch size* 32

Este segundo treino de YOLOv8 nano recorreu aos hiperparâmetros que constam na Tabela 5.4, agora com um *batch size* inferior. Os resultados verificam-se na Figura 5.19. Apesar de apresentar curvas semelhantes às do primeiro treino, os gráficos das perdas da avaliação do modelo sugerem um pequeno *overfit* ao mostrarem um declive positivo após a *epoch* 50. As métricas obtidas continuam a mostrar oscilações elevadas.

O *overfit* é mais visível quando analisada a Matriz de Confusão da Figura 5.20. A mesma retrata maiores percentagens de Falsos Negativos e de Falsos Positivos, não só para *background*, como também entre diferentes classes. Registou-se, também, uma subida no valor de Verdadeiros Positivos da classe *ghost_net*. As inferências no *evaluation set* não mostram diferenças significativas (Figura 5.21).

Tabela 5.4: Hiperparâmetros - com *batch size* de 32 - utilizados num treino do YOLOv8 nano

Hiperparâmetro	Valor
<i>Batch size</i>	32
<i>Epochs</i>	200
<i>Learning rate</i>	0,0005
<i>Momentum</i>	0,95
<i>Weight decay</i>	0,005
<i>Optimizer</i>	SGD

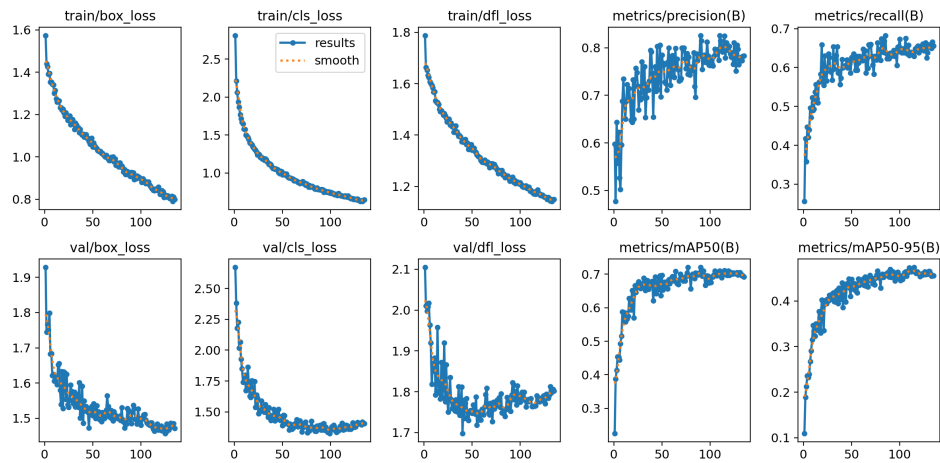


Figura 5.19: Resultado do treino do YOLOv8 nano com *batch size* 32

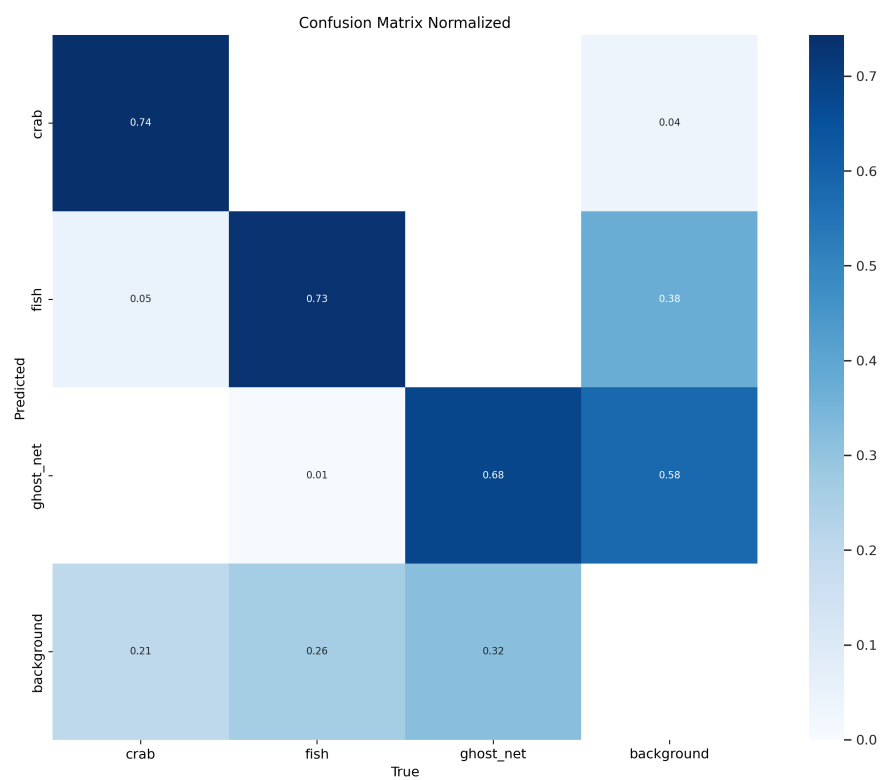


Figura 5.20: Matriz de confusão do YOLOv8 nano com *batch size* 32

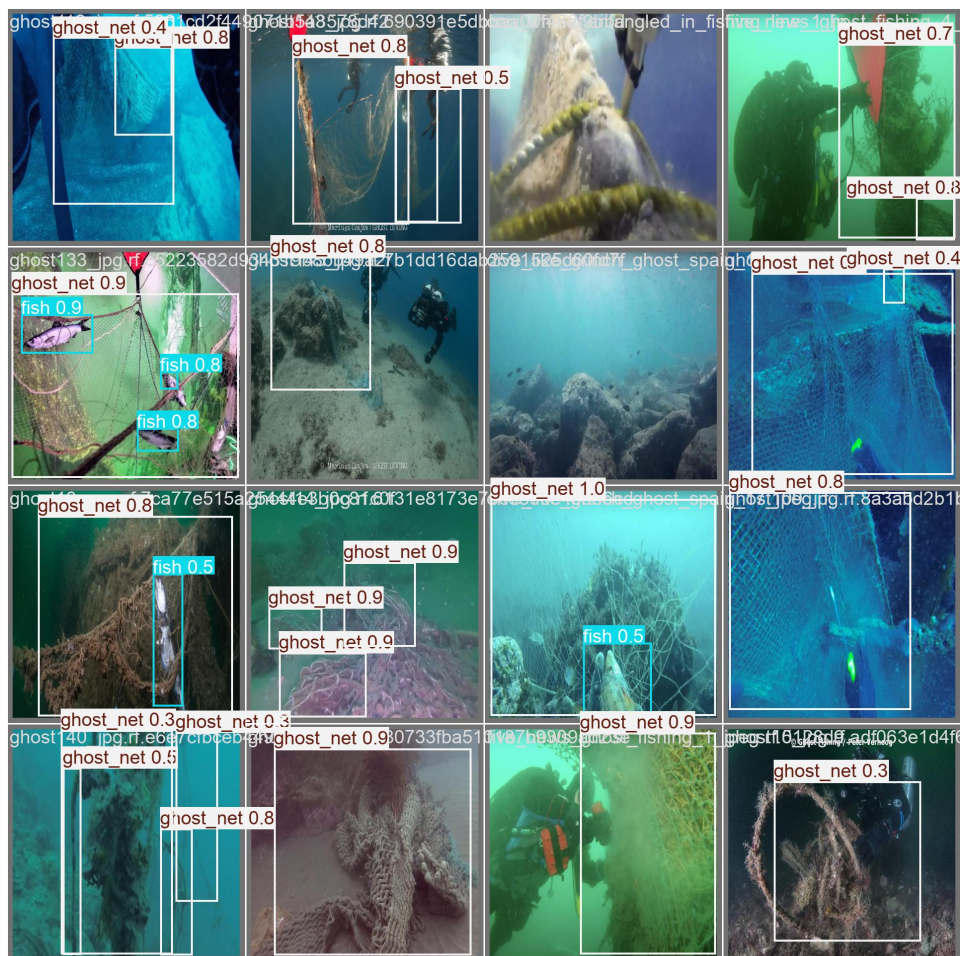


Figura 5.21: Inferência do YOLOv8 nano com *batch size* 32 no *dataset* de validação

5.5.3 YOLOv8 nano com *batch size* 128

Por fim, o modelo foi treinado com um *batch size* de 128, adotando igualmente o otimizador AdamW e os restantes hiperparâmetros registados na Tabela 5.5.

Ao analisar a Figura 5.22, verifica-se uma oscilação bastante elevada nos gráficos de *evaluation loss*. Uma vez que os *plots* das perdas do treino apresentam uma descida estável, a oscilação paralela na avaliação pode, mais uma vez, representar o *overfit* do modelo. Não obstante, o *recall* e as mAPs parecem ter um bom resultado ao descrever curvas ascendentes estáveis. Por outro lado, o gráfico relativo à precisão mostra uma curva de aprendizagem acentuada. Este conjunto de fatores pode sugerir um modelo que evita Falsos Positivos, mas que também falha a deteção de Verdadeiros Positivos.

A análise à Matriz de Confusão (Figura 5.23) mostra, efetivamente, um menor valor de Falsos Positivos de *Ghost Nets*. No entanto, o seu número de Verdadeiros Positivos é igualmente inferior. As precisões das outras classes são igualmente afetadas, comprovando um menor número de Verdadeiros Positivos em todas.

As inferências no *set* de avaliação encontram-se na Figura 5.24.

Tabela 5.5: Hiperparâmetros - com *batch size* de 128 - utilizados num treino do YOLOv8 nano

Hiperparâmetro	Valor
<i>Batch size</i>	128
<i>Epochs</i>	200
<i>Learning rate</i>	0,0005
<i>Momentum</i>	0,937
<i>Weight decay</i>	0,0002
<i>Optimizer</i>	AdamW

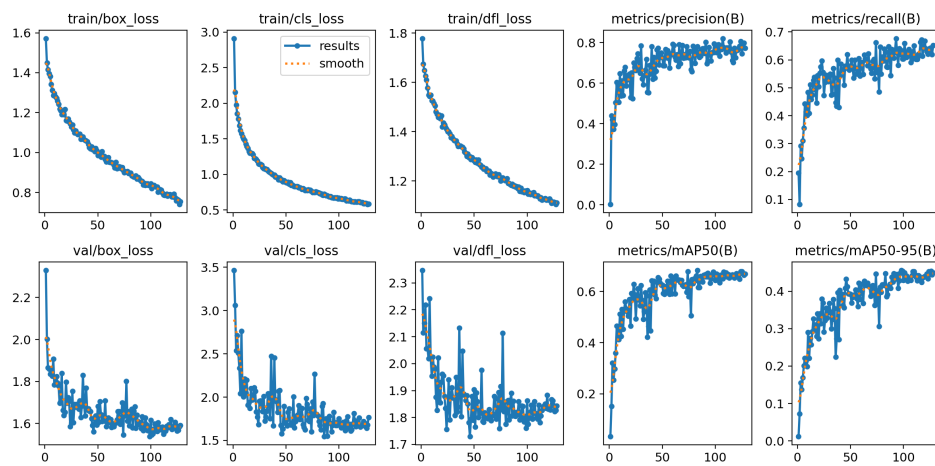


Figura 5.22: Resultado do treino do YOLOv8 nano com *batch size* 128

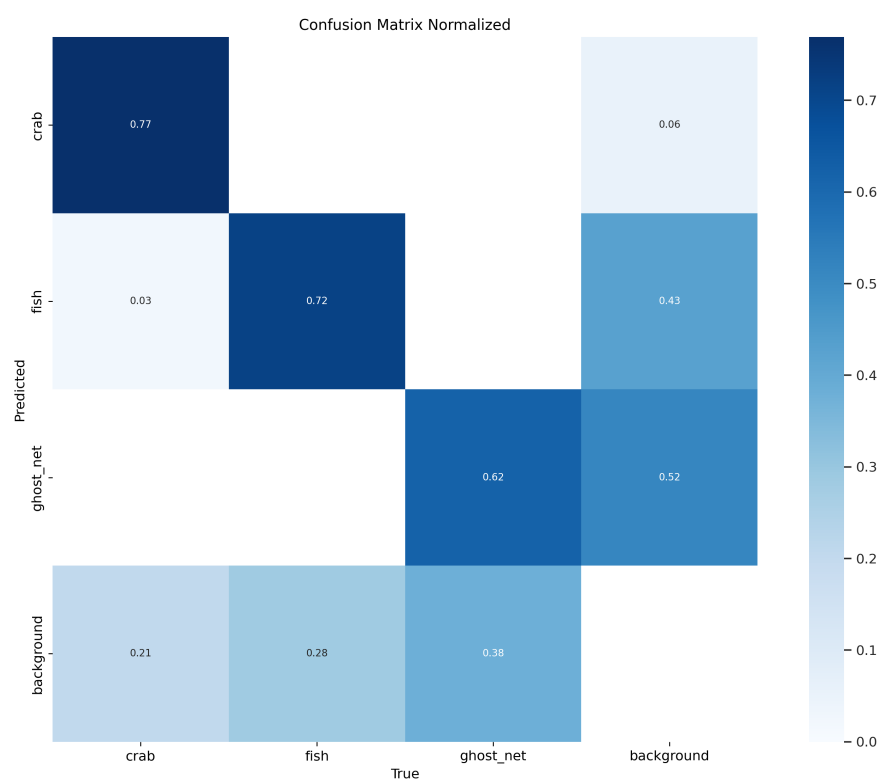


Figura 5.23: Matriz de confusão do YOLOv8 nano com *batch size* 128

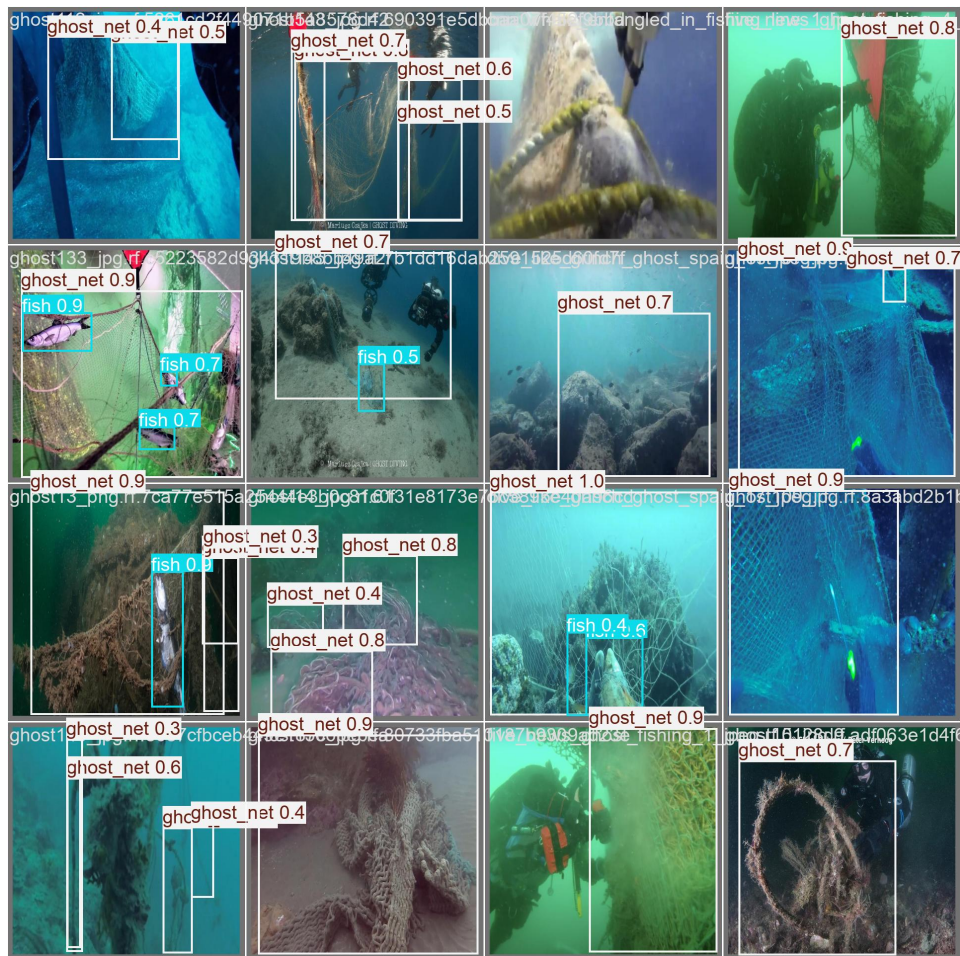


Figura 5.24: Inferência do YOLOv8 nano com *batch size* 128 no *dataset* de validação

5.6 Inferência nas imagens de teste

Considerando a análise efetuada aos treinos, numa primeira instância, optou-se por prosseguir com os testes ao YOLOv5 nano de *batch size* 128 e ao YOLOv8 nano de *batch size* 32.

A Figura 5.25 mostra as inferências dos modelos em fotografias do *test set* captadas em águas com excelente visibilidade. A imagem do tubarão mostra uma confiança na deteção com YOLOv8 de 93%, bastante superior à do outro modelo. Em adição, a *bounding box* gerada cobre a totalidade da rede, inclusive o cabo que se estende à esquerda. O YOLOv5 desconsiderou a rede visível nessa parte da imagem.

Ainda na mesma Figura, o YOLOv5 voltou a apresentar um *score* de apenas 48% na fotografia da rede de emalhar. Para além do resultado superior do v8, a *bounding box* ajusta melhor à arte de pesca. De notar ainda uma segunda caixa para outra malha detetada a uma maior profundidade.

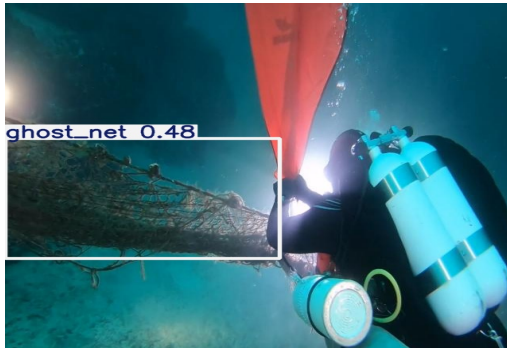
De seguida, seguiu-se a inferência dos modelos em imagens de fraca visibilidade. A Figura 5.26 apresenta diferentes testes realizados aos YOLOv5/v8. Observando a primeira fotografia, verifica-se o *score* de 70% na deteção da rede por parte do YOLOv5. É um valor bastante superior aos obtidos com boa visibilidade. Não obstante, o YOLOv8 volta a bater esta inferência com 87% de confiança.



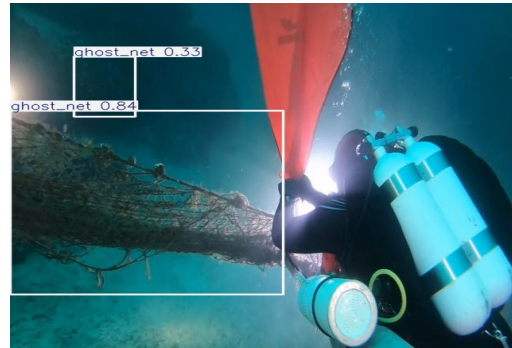
(a) Tubarão emaranhado - YOLOv5 - 50%



(b) Tubarão emaranhado - YOLOv8 - 93%



(c) Rede de Emalhar - YOLOv5 - 48%



(d) Rede de Emalhar - YOLOv8 - 84%

Figura 5.25: Inferência dos modelos em imagens do *test set* com boa visibilidade. YOLOv5 à esquerda e YOLOv8 à direita

A segunda linha de imagens mostra cabos de artes de pesca num naufrágio. Foram geradas várias detecções por parte dos dois algoritmos e, como retrata a imagem, o YOLOv5 mostrou-se capaz de detetar os cabos mais pequenos à direita. De notar ainda que o YOLOv8 gerou uma caixa para cada um dos cabos localizados à esquerda, enquanto que o outro modelo considerou apenas uma caixa para o conjunto com um *score* inferior.

A última linha de imagens ilustra uma rede suspensa. Ambos os algoritmos deteram-na com confiança excelentes, tendo o YOLOv5 registado a maior.

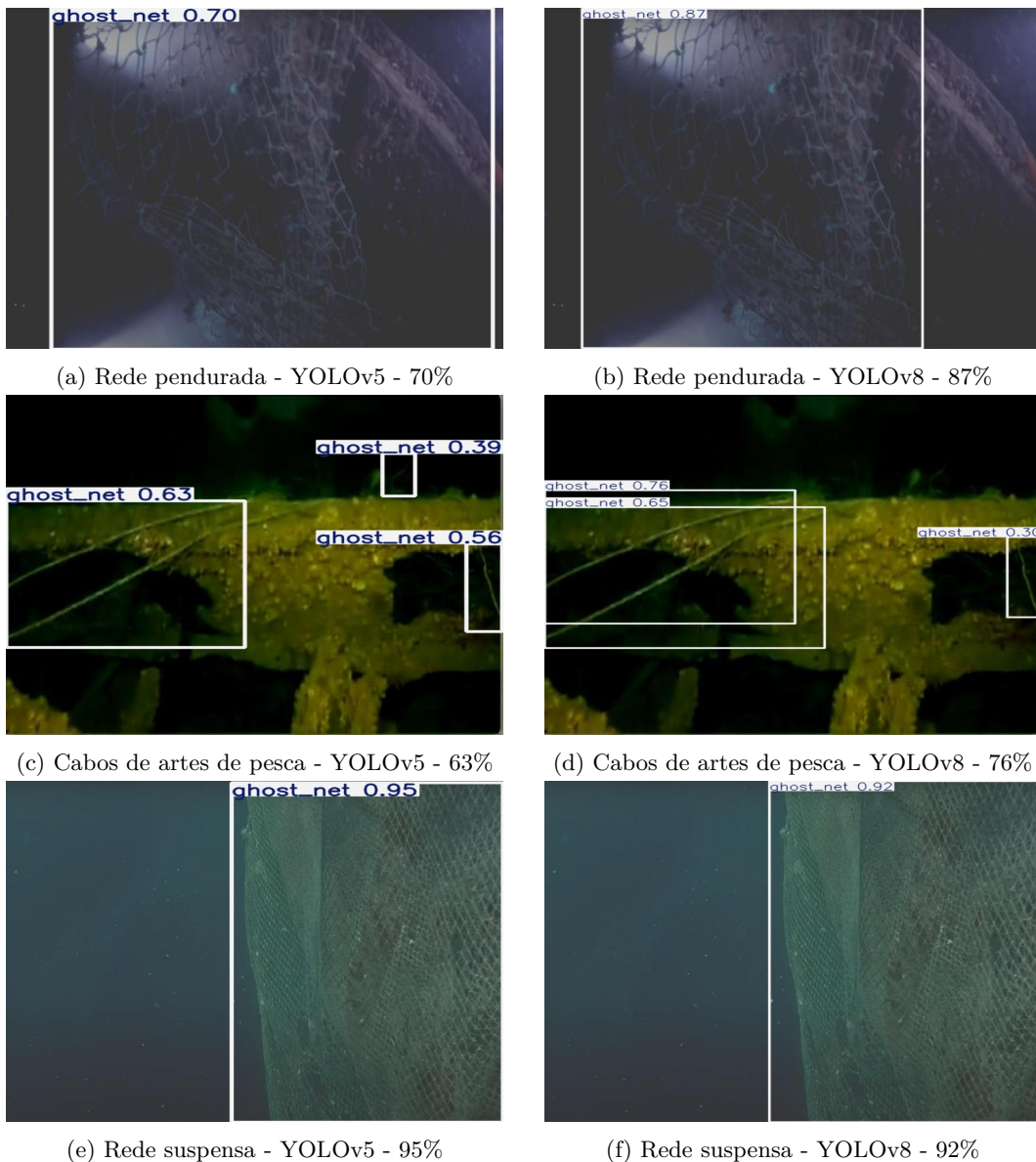


Figura 5.26: Inferência dos modelos em imagens do *test set* com fraca visibilidade. YOLOv5 à esquerda e YOLOv8 à direita

5.7 Inferência no ambiente simulado

A simulação em Stonefish não conta com fotografias do mundo real, mas apenas com as 4 artes de pesca criadas em Blender. Assim, foram obtidas imagens destes objetos no simulador, a fim de inferir com os YOLOv5 e YOLOv8. Como tal, a primeira linha da Figura 5.27 mostra os resultados numa imagem de uma destas armadilhas. O YOLOv8 apresenta uma confiança de 78%, enquanto que a do YOLOv5 é de apenas 47%. Apesar de um valor mais alto, a *bounding box* gerada pelo v8 engloba também a sombra da armadilha.

Mencionado em 4.2.1, a constante de *Jerlov* foi aumentada em 29% a fim de simular condições de visibilidade adversas. Mais uma vez, as inferências resultaram em *scores* superiores aos obtidos na simulação em águas limpas. Verificável pela segunda fila da Figura 5.27, o v5 conseguiu 72% de certeza, enquanto que o YOLOv8 alcançou 85%, embora volte a abranger a sombra da armadilha na *bounding box*.

Posto este teste, foram igualmente executadas inferências sob a rede de Emalhar. Os resultados, ilustrados na Figura 5.28, mostram confianças inferiores às registadas nas Armadilhas. Para um valor de *Jerlov* nulo, ambos os modelos consideraram a sombra da rede na *bounding box* e, uma vez mais, a do YOLOv5 é a que melhor se ajusta à rede. O aumento do valor de *Jerlov* resultou no consequente aumento das

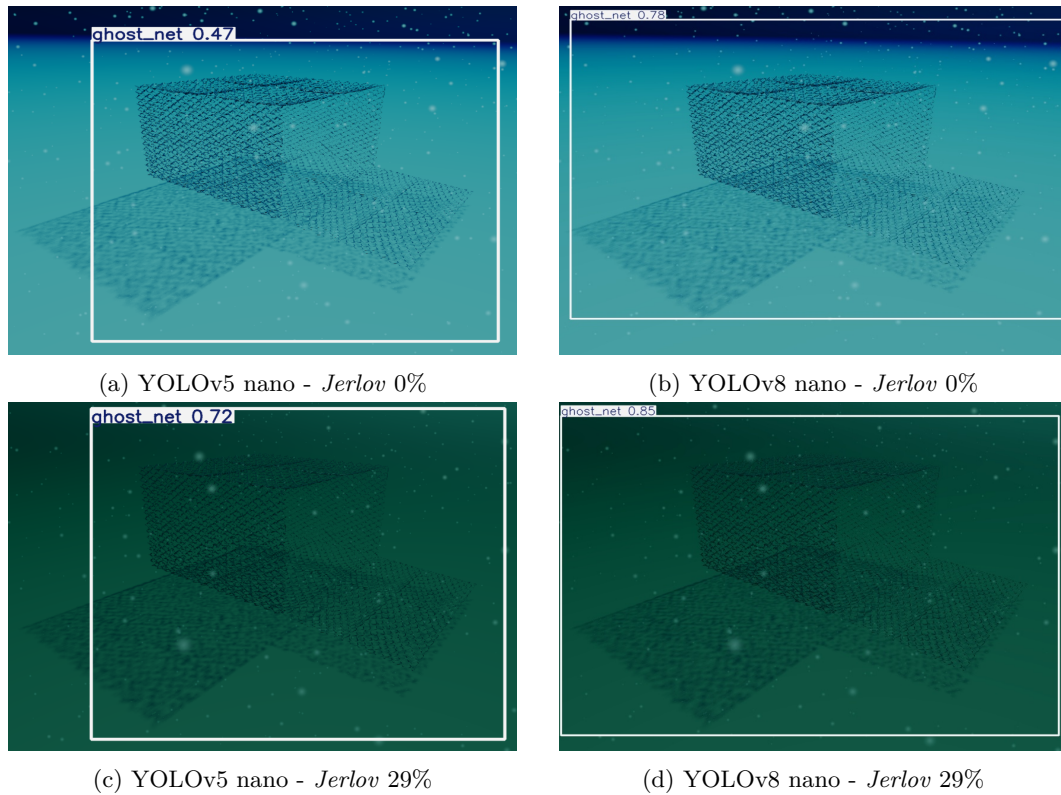


Figura 5.27: Armadilhas no Stonefish em diferentes condições de visibilidade

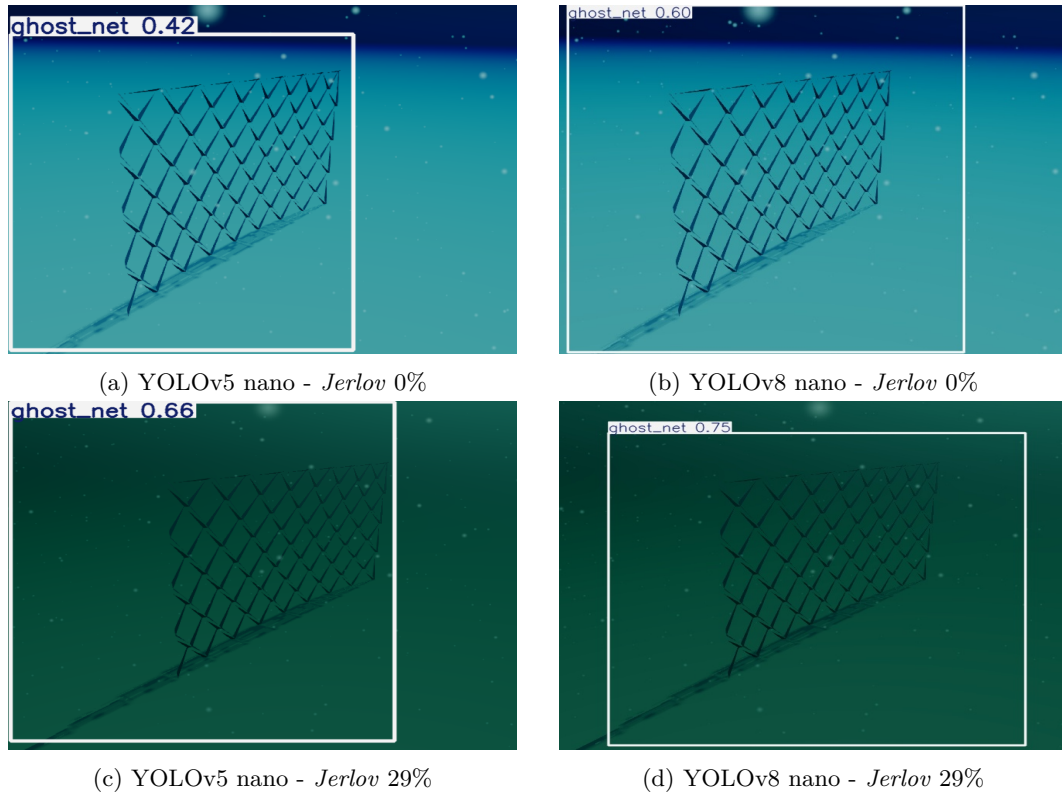


Figura 5.28: Rede de Emalhar no Stonefish em diferentes condições de visibilidade

confianças, registrando 66% na inferência do YOLOv5 e 75% no v8.

5.8 Controlo visual da IRIS

Uma vez comprovada a deteção de Ghost Nets com base nos algoritmos de *Deep Learning*, foi necessário desenvolver uma solução para integrá-los na simulação por forma a efetuar o controlo da IRIS.

A arquitetura do sistema, elaborada em 4.3.4, inclui os *packages* IRIS Vision e o IRIS Control. O IRIS Vision é responsável pela perceção visual do veículo ao inferir em tempo-real no *feed* da câmara e transmitir os dados de navegação relevantes ao pacote de controlo. Por sua vez, o foco do IRIS Control passa pelo processamento desses mesmos dados, para a posterior aplicação de *inputs* aos *thrusters* do veículo. Já mencionado nesta dissertação, relembra-se que ambos são lançados e geridos pelo orquestrador do IRIS Stonefish.

5.8.1 IRIS Vision

Este *package* conta com o Ghost Net Inference (Figura 5.29). Fundamentalmente, trata-se de um nó de ROS que subscreve ao tópico `/iris/proscilica_front/image_color`, no qual está a ser publicada a imagem da câmara frontal da IRIS.

Para cada frame subscrito, a imagem é convertida de ROS para *cv2*. As dimensões da *frame* são imediatamente extraídas, assim como são calculadas as coordenadas do seu ponto central, sendo armazenadas no vetor *center_points_coordinates*.

O modelo YOLOv5 treinado encontra-se presente neste *package*. A sua inferência gera, para cada *frame*, as *bounding boxes* nos objetos de interesse, com a respetiva classe e confiança. A partir daqui, o algoritmo identifica a *bounding box* com o maior *score*. Esta imagem é continuamente publicada numa janela para a visualização do utilizador, permitindo-lhe ver a inferência com melhor resultado quase em tempo-real.

Posteriormente, as coordenadas do ponto central da *box* são calculadas e armazenadas no *center_points_coordinates*, juntamente com as dimensões da caixa. O vetor é então publicado no tópico `/iris/proscilica_front/ghost_detection_data` a fim de ser utilizado pelo IRIS Control.

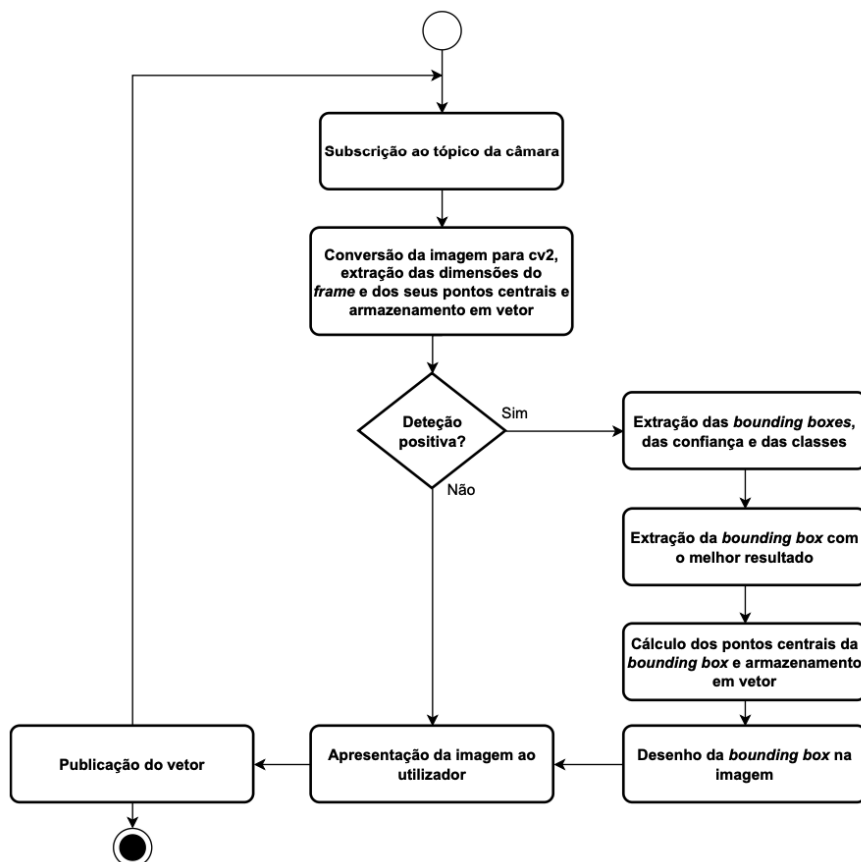


Figura 5.29: Diagrama de fluxo do Ghost Net Inference

5.8.2 IRIS Control

Em paralelo, o IRIS Control executa o nó Setpoints Visual Navigation (Figura 5.30) que subscrive ao `/iris/proscilica_front/ghost_detection_data`. Os dados extraídos daqui são empregues na lógica de controlo de *Surge*, *Heave* e *Yaw*.

Os controladores de *Heave* e *Yaw* são PI. A atuação em *Surge* recorre a um controlador PID, onde o erro utilizado é dado pela diferença entre o perímetro da imagem e o da *bounding box*, tendo sido este cálculo já apresentado na Equação 4.4. Posteriormente, o valor resultante do PID é normalizado, sendo este o *input* direto nos 4 *thrusters* de cauda. A Figura 5.31c retrata a imagem da câmara frontal durante a manobra de aproximação à rede com base neste cálculo. Verifica-se um retângulo verde na orla da imagem, representativo do limite de aproximação à rede.

Por sua vez, e já explícito na Equação 4.3, o erro em *Heave* é dado pela diferença entre a coordenada *y* do ponto central da *frame* com a mesma do ponto central da *bounding box*. Já o erro em *Yaw* é dado pela mesma diferença na componente *x*. Os *outputs* destes controladores são então normalizados e guardados num vetor de *setpoints*, tendo em vista atuar nos 2 *thrusters* verticais e nos 2 laterais. O efeito é o ilustrado na Figura 5.31. De notar o vetor de aproximação lateral dado pelo erro em *Yaw*, na Figura 5.31a. Embora a Figura 5.31b seja descritiva do vetor de aproximação vertical, dado pelo erro em *Heave*, verifica-se igualmente uma componente em *x*. Este é um exemplo que ilustra o funcionamento dos dois controladores em simultâneo.

Todos os valores calculados de *thrust* são armazenados num vetor *setpoints* que é publicado para o tópico `/iris/controller/thruster_setpoints`. O mesmo é subscrito pelo nó Thrust Setpoints Pub que atua diretamente nos motores do IRIS.

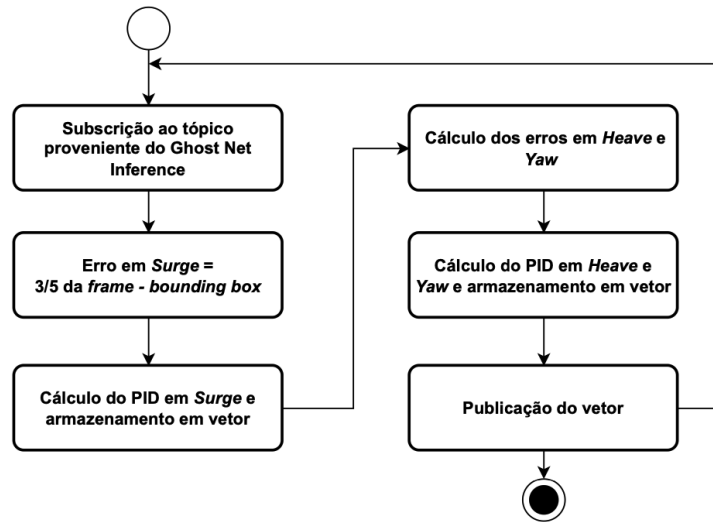
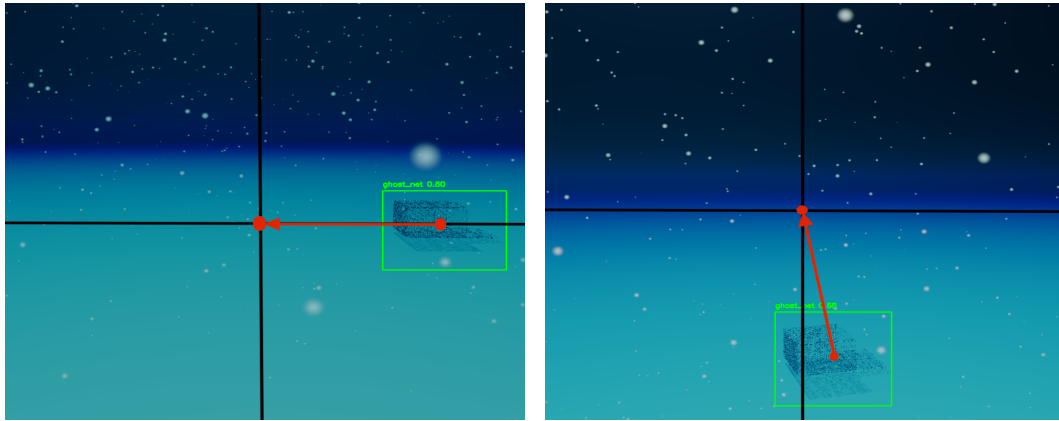
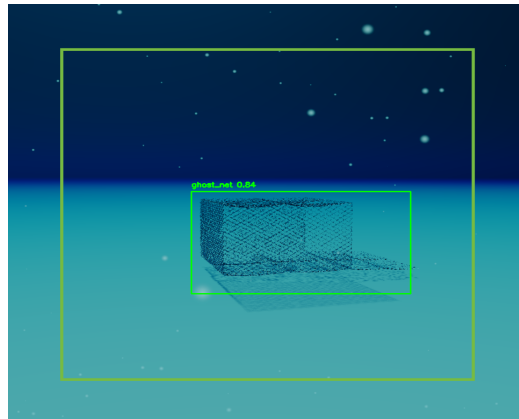


Figura 5.30: Diagrama de fluxo do Setpoints Visual Navigation



(a) Vetor de aproximação lateral do centro da *box* com o centro da *frame* (b) Vetor de aproximação vertical do centro da *box* com o centro da *frame*



(c) Exemplo do limite estipulado para a aproximação à GN com base na diferença entre as dimensões da *frame* com a *box*

Figura 5.31: Estratégia para o controlo visual do veículo

5.9 Teste da simulação do IRIS

O término do desenvolvimento dos algoritmos de visão e de controlo deu seguimento à integração dos mesmos numa simulação. A implementação do YOLOv8 nano não foi possível no IRIS Vision dada a aparente incompatibilidade com o ROS Noetic. Ainda que o modelo treinado tenha sido exportado para o formato *Open Neural Network Exchange* (ONNX), a sua integração continuou por se realizar, dada a incompatibilidade de algumas das suas bibliotecas com o Python3.8. Não obstante, a integração do YOLOv5 realizou-se sem problemas no IRIS Vision.

Ao ser lançado, o Stonefish fornece ao utilizador uma janela do cenário e outra para a visualização da inferência executada no IRIS. A Figura 5.32 ilustra uma simulação em águas limpas, nas quais o veículo se encontra posicionado a cerca de 4,7 m da armadilha. Nestas condições, o YOLOv5 mostra confiança na deteção na ordem dos 80%. A partir do momento em que a rede é detetada, os controladores atuam em simultâneo nos *thrusters*, de acordo com as estratégias já explicadas. Verificam-se, portanto, alterações imediatas em *Surge*, *Heave* e *Yaw*. De notar ainda que os parâmetros PID foram afinados por tentativa-erro a cada simulação executada.

O IRIS descreve, então, uma trajetória estável até à distância de segurança para com a rede, sendo que estabiliza nesse ponto, como mostra a Figura 5.33. Apesar do sucesso na navegação à GN simulada, a *bounding box* encontra-se ligeiramente abaixo do centro da *frame*. Não obstante, a simulação valida a empregabilidade do YOLOv5 para este tipo de missão, sendo que o *stream* de imagens na janela de inferência é quase contínuo.

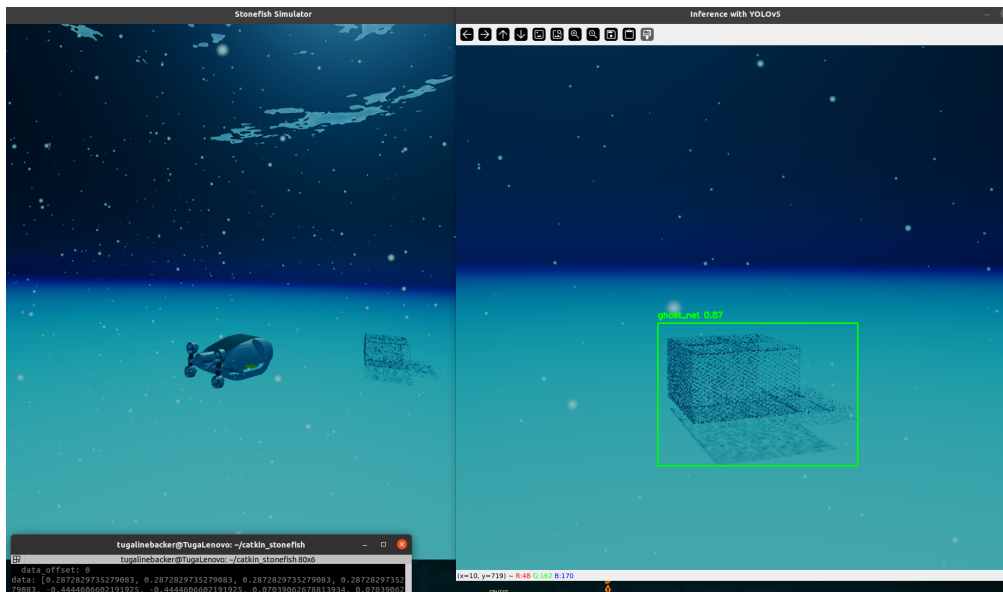


Figura 5.32: Simulação da navegação visual do IRIS. Encontra-se à esquerda o cenário de simulação e os valores publicados nos *thrusters*.

À direita, visualiza-se a inferência que o veículo executa

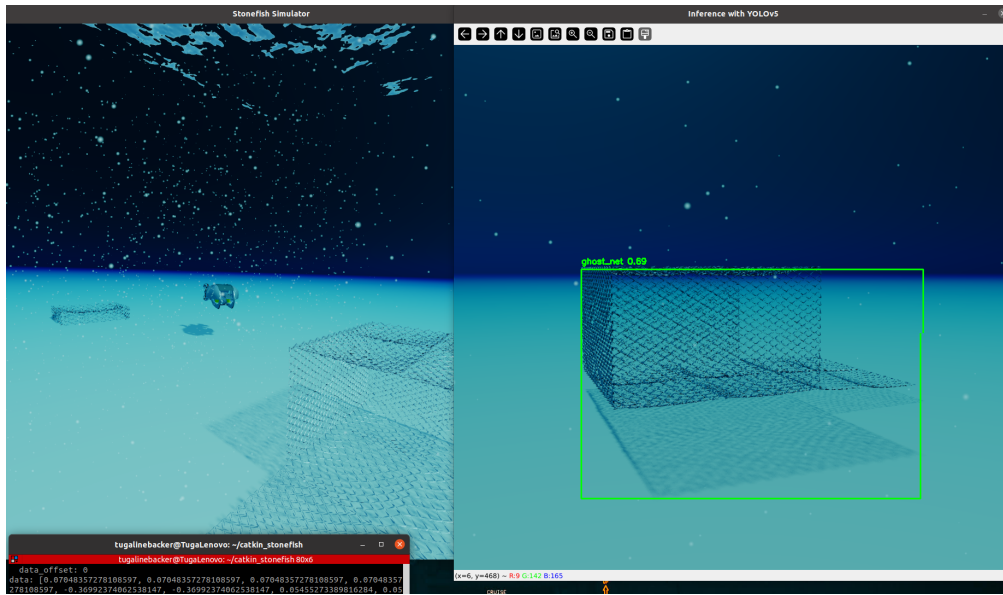


Figura 5.33: Estabilização do IRIS na distância de segurança à rede

Outros testes realizados contaram com a alteração do valor de *Jerlov*, por forma a validar a navegação visual em condições de visibilidade adversas. A Figura 5.34 ilustra o início de uma aproximação à armadilha num ambiente mais turvo, caracterizado por uma constante de *Jerlov* de 18%. Apesar do *score* inferior na inferência, o modelo deteta a *Ghost Net* e conduz o submarino à mesma.

Durante uma simulação em boas condições de visibilidade, na qual o submarino se encontrava estável na distância de segurança, a constante de *Jerlov* foi abruptamente

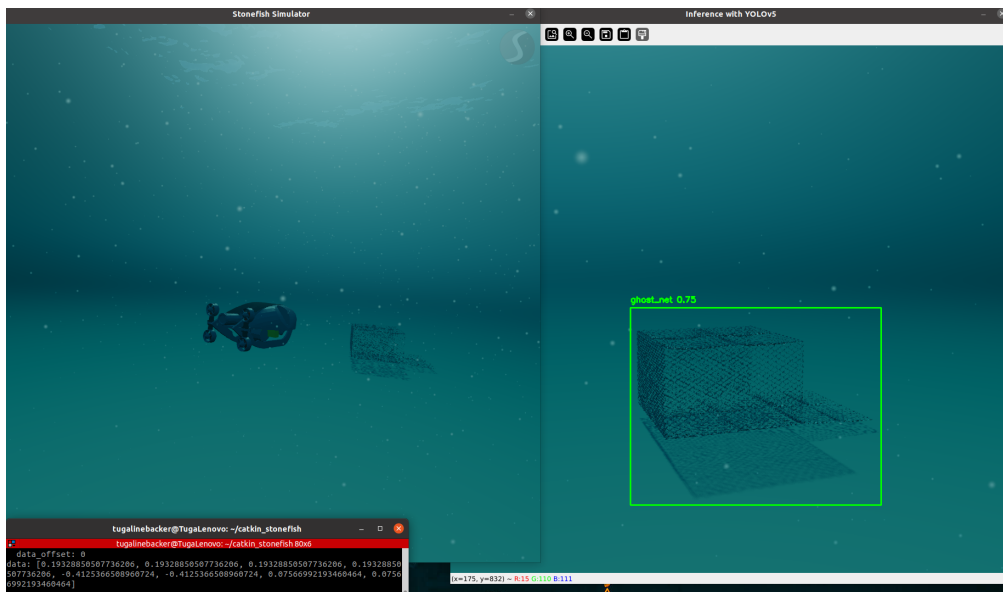


Figura 5.34: Simulação da navegação visual do IRIS com um valor de *Jerlov* de 18%

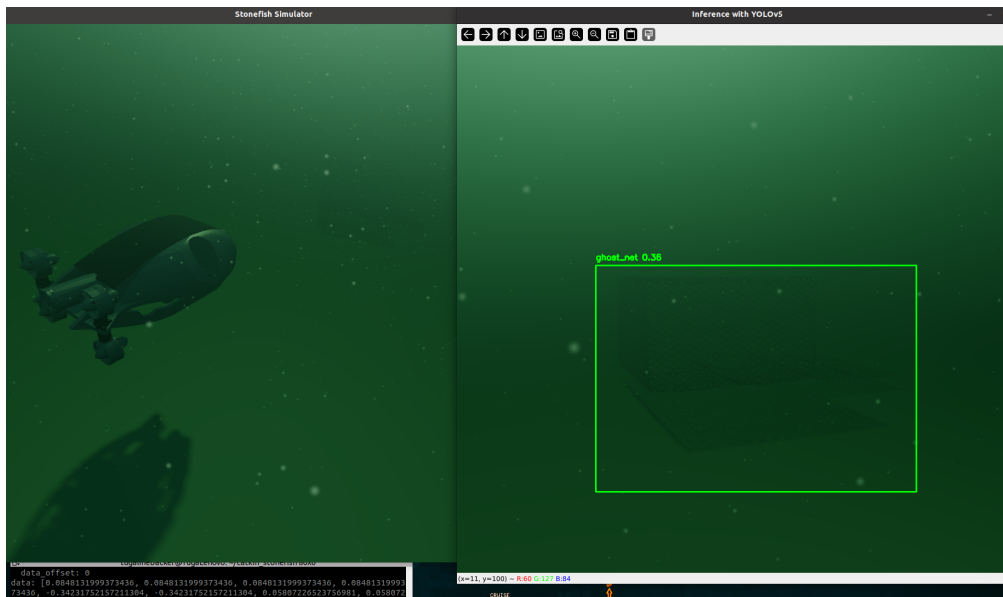


Figura 5.35: Simulação da navegação visual do IRIS com um valor de *Jerlov* de 38%

alterada para 38%. Como esperado, a confiança na inferência diminuiu, sendo mesmo que o algoritmo não detetou a armadilha nalguns *frames*. Esta ação induziu uma oscilação no veículo devido a inferências ocasionais e consequente ação interrupta nos motores. Estes fatores levaram ao afastamento do IRIS em relação à armadilha. No entanto, e talvez devido aos valores conservadores dos controladores, o deslocamento não foi o suficiente para que o veículo perdesse a armadilha de vista.

Capítulo 6

Conclusões

Esta dissertação aborda a simulação da navegação de um *Autonomous Underwater Vehicle*, tendo por base a deteção visual de *Ghost Nets* com recurso a *Deep Learning*. Inserida no âmbito do projeto Nettag, a mesma procura fornecer uma solução que habilite o submarino a navegar de forma autónoma ao encontro das GNs, tendo em vista a posterior manobra de recuperação.

De uma forma geral, os objetivos propostos foram cumpridos. A manobra de aproximação visual do IRIS às *Ghost Nets* foi executada com sucesso, tendo comprovado a viabilidade da solução a ser implementada no veículo real.

Relativamente ao simulador, o Stonefish foi fundamental para a qualidade da simulação devido ao seu módulo de hidrodinâmica e à simples interação com o ROS. O simulador permitiu verificar o efeito da inércia e do atrito no veículo. A interface com o ROS habilitou o controlo do IRIS com recurso a um orquestrador responsável por gerir os diferentes pacotes de cenário, perceção e controlo. É, no entanto, de lembrar que a máquina utilizada nestes testes possui uma Nvidia MX250, já que é recomendada uma GPU dedicada para o *rendering* da simulação.

A coleção do *dataset* constou num processo iterativo, tendo sido criadas várias versões com diversas augmentações por forma a robustecer o modelo. A iteração mais recente conta com 2000 imagens originais e outras 2800 augmentadas. Ainda assim, é com certeza um ponto de melhoria. Por via de regra, em *Deep Learning*, quanto maior for o conjunto de dados, melhor será o modelo resultante.

Por sua vez, o desenvolvimento do algoritmo de *Deep Learning* passou pela implementação do YOLO, tendo sido experimentados o YOLOv5 nano e o YOLOv8

nano. São as versões mais leves destas arquiteturas e, portanto, as mais indicadas a operarem num sistema computacionalmente limitado. Embora não tenha sido possível a integração do YOLOv8 na simulação, o mesmo mostrou um grande potencial para a detecção das artes de pesca. Não obstante, e apesar de resultados inferiores, o YOLOv5 foi validado no simulador.

Embora fundamentados num conceito simples, os controladores desenvolvidos executaram o seu propósito. O controlo em *Heave* e *Yaw* é obtido pela diferença entre as coordenadas do ponto central da imagem com as da *bounding box* detetada. Simultaneamente, o controlo em *Surge* é dado pela diferença entre o tamanho da imagem e da *bounding box*.

A integração destes componentes numa simulação foi alcançada com sucesso. Aquando uma detecção de uma *Ghost Net* por parte do submarino, o mesmo navega de forma autónoma até à rede. A inferência do YOLOv5 é executada em quase tempo-real, mostrando um pequeno atraso na visualização da imagem. O controlador do veículo apresenta uma resposta rápida aos dados da inferência, atuando quase de imediato nos *thrusters*. Pese embora as diferentes afinações executadas aos controladores, o deslocamento do veículo é lento com os parâmetros atuais, sendo este outro ponto de melhoria.

6.1 Trabalho Futuro

Como trabalho futuro, pretende-se robustecer esta simulação em todas as suas vertentes, procurando atingir melhores resultados em condições mais complexas.

O cenário de simulação deverá não só incluir versões mais detalhadas dos modelos atuais das artes de pesca, mas também incorporar outros tipos das mesmas, como é o caso do Palangre. Para além do mais, deve acrescentar algum relevo ao solo marinho por forma a assemelhar-se o mais possível com a realidade e a reduzir o contraste entre a *Ghost Net* e o solo.

Como mencionado, o *dataset* é um ponto a trabalhar. A coleção contínua de dados deve ser levada a cabo, procurando englobar os diversos cenários encontrados no mar. Mais ainda, a classe atual *ghost_net* engloba redes, armadilhas e cabos. Portanto, deverá ser partida em 3, nomeadamente, em *ghost_net*, *ghost_trap* e *ghost_line*.

Atendendo aos melhores resultados obtidos na inferência, a integração do YOLOv8 na simulação deverá ser revisitada. Ainda assim, deve ser igualmente analisada a migração do sistema de ROS1 para ROS2, procurando mantê-lo atualizado com os algoritmos e as bibliotecas mais recentes.

Por fim, a afinação do controlador do IRIS deve ser revista a fim de alcançar uma manobra mais precisa.

Referências

- [1] U. N. E. Programme, “Marine plastic debris and microplastics - Global lessons and research to inspire action and guide policy change,” tech. rep., United Nations, Oct. 2016. Available at: <https://wedocs.unep.org/handle/20.500.11822/7720>. [Citado nas páginas 1 e 6]
- [2] A. Löhr, H. Savelli, R. Beunen, M. Kalz, A. Ragas, and F. Van Belleghem, “Solutions for global marine litter pollution,” *Current Opinion in Environmental Sustainability*, vol. 28, pp. 90–99, 2017. Sustainability governance. [Citado nas páginas 1 e 7]
- [3] J. Jambeck, R. Geyer, C. Wilcox, T. Siegler, M. Perryman, A. Andrady, R. Narayan, and K. Law, “Plastic waste inputs from land into the ocean,” vol. 347, Feb. 2015. Available at: <https://science.sciencemag.org/content/347/6223/768>. [Citado na página 1]
- [4] C. Arthur, A. E. Sutton-Grier, P. Murphy, and H. Bamford, “Out of sight but not out of mind: Harmful effects of derelict traps in selected U.S. coastal waters,” *Marine Pollution Bulletin*, vol. 86, pp. 19–28, July 2014. Available at: <https://www.sciencedirect.com/science/article/pii/S0025326X14004305>. [Citado nas páginas 1 e 5]
- [5] J.-E. Chang, S.-W. Huang, and J.-H. Guo, “Hunting ghost fishing gear for fishery sustainability using autonomous underwater vehicles,” in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pp. 49–53, 2016. [Citado na página 1]
- [6] M. Ludvigsen, T. Thorsnes, R. E. Hansen, A. J. Sørensen, G. Johnsen, P. A. Lågstad, Ødegård, M. Candeloro, S. M. Nornes, and C. Malmquist, “Underwater vehicles for environmental management in coastal areas,” in *OCEANS 2015 - Genova*, pp. 1–6, 2015. [Citado na página 1]
- [7] S. Li, B. Pan, Y. Cheng, X. Yan, C. Wang, and C. Yang, “Underwater fish object detection based on attention mechanism improved ghost-yolov5,” in *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pp. 599–603, 2022. [Citado na página 2]

- [8] N. Tag, “NETTAG - Sobre.” Available at: <https://nettag.ciimar.up.pt/about.php>. [Citado na página 2]
- [9] A. Martins, C. Almeida, P. Lima, D. Viegas, J. Silva, J. M. Almeida, C. Almeida, S. Ramos, and E. Silva, “A robotic solution for NETTAG lost fishing net problem,” 2020. Available at: <https://globaloceans2020.org/proceedings/session.cfm?sid=33&SG=33>. [Citado nas páginas 2, 3, 5 e 28]
- [10] N. Graham, N.-R. Hareide, P. A. Large, P. MacMullen, M. Mulligan, P. J. Randall, D. Rihan, and D. Peach, “Recuperation of fishing nets lost or abandoned at sea,” tech. rep., The European Commission Directorate - General for Fisheries and Maritime Affairs, Sept. 2009. Available at: <https://webgate.ec.europa.eu/maritimeforum/en/node/1081>. [Citado nas páginas 6 e 8]
- [11] G. Diving, “News and publications,” 2021. Available at: <https://ghostdiving.org/news>. [Citado nas páginas 6, 7 e 39]
- [12] D. G. de Recursos Naturais Segurança e Serviços Marítimos, “Cerco,” 2018. Available at: <https://www.dgrm.mm.gov.pt/web/guest/cerco>. [Citado nas páginas 7 e 8]
- [13] D. G. de Recursos Naturais Segurança e Serviços Marítimos, “Artes de Pesca,” 2018. Available at: <https://www.dgrm.mm.gov.pt/artes-de-pesca>. [Citado nas páginas 7 e 9]
- [14] D. G. de Recursos Naturais Segurança e Serviços Marítimos, “Arrasto,” 2018. Available at: <https://www.dgrm.mm.gov.pt/pt/web/guest/arrasto>. [Citado na página 8]
- [15] D. G. de Recursos Naturais Segurança e Serviços Marítimos, “Redes de emalhar e de tresmalho,” 2018. Available at: <https://www.dgrm.mm.gov.pt/web/guest/redes-de-emalhar>. [Citado na página 8]
- [16] D. G. de Recursos Naturais Segurança e Serviços Marítimos, “Envolvente arrastante - arte xávega,” 2018. Available at: <https://www.dgrm.mm.gov.pt/web/guest/envolventes-arrastantes>. [Citado nas páginas 8 e 9]
- [17] A. Martins and P. Kinas, “Boletim estatístico da pesca marinha do sul do rio grande do sul - 2017. 2º semestre. universidade federal do rio grande,” 01 2018. [Citado na página 9]
- [18] S. Lino, “E se a cura para o cancro estiver no mar profundo!?,” Dec. 2013. Available at: <https://blogmarprofundo.wordpress.com/2013/12/17/e-se-a-cura-para-o-cancrocancer-estiver-no-mar-profundo/>. [Citado na página 10]

-
- [19] “Deep Learning vs. Machine Learning: Beginner’s Guide | Coursera.” Available at: <https://www.coursera.org/articles/ai-vs-deep-learning-vs-machine-learning-beginners-guide>. [Citado na página 11]
- [20] J. Nelson, “How to select the right computer vision model architecture,” 2020. Available at: <https://blog.roboflow.com/yolov3-vs-mobilenet-vs-faster-rcnn/>. [Citado na página 11]
- [21] P. Potrimba, “What is a convolutional neural network?..,” 2023. Available at: <https://blog.roboflow.com/what-is-a-convolutional-neural-network/>. [Citado na página 12]
- [22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 2014. Available at: <https://arxiv.org/abs/1311.2524>. [Citado na página 12]
- [23] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016. Available at: <https://arxiv.org/abs/1506.01497>. [Citado nas páginas 12 e 13]
- [24] X. Zhou, H. Liu, C. Shi, and J. Liu, “Chapter 3 - model design and compression,” in *Deep Learning on Edge Computing Devices* (X. Zhou, H. Liu, C. Shi, and J. Liu, eds.), pp. 39–58, Elsevier, 2022. Available at: <https://www.sciencedirect.com/science/article/pii/B9780323857833000119>. [Citado na página 13]
- [25] F. Sultonov, J.-H. Park, S. Yun, D.-W. Lim, and J.-M. Kang, “Mixer u-net: An improved automatic road extraction from uav imagery,” *Applied Sciences*, vol. 12, p. 1953, 02 2022. [Citado na página 13]
- [26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017. Available at: <https://arxiv.org/abs/1704.04861>. [Citado na página 14]
- [27] Roboflow, “Mobilenet ssd v2,” 2024. Available at: <https://roboflow.com/model/mobilenet-ssd-v2>. [Citado na página 14]
- [28] J. Nelson, “What is yolo? the ultimate guide [2024],” 2024. Available at: <https://blog.roboflow.com/guide-to-yolo-models/>. [Citado na página 14]
- [29] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 2018. Available at: <https://arxiv.org/abs/1804.02767>. [Citado nas páginas 14 e 15]

- [30] J. Solawetz, “What is yolov5? a guide for beginners,” 2020. Available at: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>. [Citado na página 15]
- [31] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics YOLO,” Jan. 2023. [Citado nas páginas 15 e 33]
- [32] J. Kim, T. Kiml, J. Kim, S. C. Yu, and T. Kim, “Manipulation purpose underwater agent vehicle for ghost net recovery mission,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3905–3910, 2019. [Citado nas páginas 16, 17 e 40]
- [33] X. YE and X. Wang, “Deep generative network and regression network for fishing nets detection in real-time,” in *2018 37th Chinese Control Conference (CCC)*, pp. 9466–9471, 2018. [Citado na página 18]
- [34] F. Xu, X. Ding, J. Peng, G. Yuan, Y. Wang, J. Zhang, and X. Fu, “Real-time detecting method of marine small object with underwater robot vision,” in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, pp. 1–4, 2018. [Citado na página 19]
- [35] Z. Bi, Y.-L. Wang, and L. Ma, “Perception and recognition of underwater cable based on deep learning,” in *2020 International Conference on System Science and Engineering (ICSSE)*, pp. 1–5, 2020. [Citado nas páginas 20 e 21]
- [36] T. T. Chungath, A. M. Nambiar, and A. Mittal, “Transfer learning and few-shot learning based deep neural network models for underwater sonar image classification with a few samples,” *IEEE Journal of Oceanic Engineering*, vol. 49, no. 1, pp. 294–310, 2024. [Citado na página 22]
- [37] J. McKay, I. Gerg, V. Monga, and R. G. Raj, “What’s mine is yours: Pretrained cnns for limited training sonar atr,” in *OCEANS 2017 - Anchorage*, pp. 1–7, 2017. [Citado nas páginas 22 e 23]
- [38] Y.-P. Zhao, L.-J. Niu, H. Du, and C.-W. Bi, “An adaptive method of damage detection for fishing nets based on image processing technology,” *Aquacultural Engineering*, vol. 90, p. 102071, 2020. Available at: <https://www.sciencedirect.com/science/article/pii/S0144860919301876>. [Citado na página 23]
- [39] B. Chen, R. Li, W. Bai, X. Zhang, J. Li, and R. Guo, “Research on recognition method of optical detection image of underwater robot for submarine cable,” in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1973–1976, 2019. [Citado na página 24]

- [40] C. A. Williamson and R. C. Hollins, “Depth profiles of jerlov water types,” *Limnology and Oceanography Letters*, vol. 8, no. 5, pp. 781–788, 2023. Available at: <https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.1002/lol2.10338>. [Citado na página 29]
- [41] P. Cieślak, “Stonefish: An advanced open-source simulation tool designed for marine robotics, with a ros interface,” in *OCEANS 2019 - Marseille*, jun 2019. [Citado nas páginas 31 e 37]
- [42] P. Cieślak, “Welcome to ros package for stonefish,” 2023. Available at: <https://stonefish-ros.readthedocs.io/en/latest/>. [Citado na página 31]
- [43] N. Pereira, “Pereiraaslnet: Asl letter recognition with yolox taking mean average precision and inference time considerations,” in *2022 2nd International Conference on Artificial Intelligence and Signal Processing (AISP)*, pp. 1–6, 2022. [Citado na página 33]
- [44] G. Jocher, “YOLOv5 by Ultralytics,” May 2020. Available at: <https://github.com/ultralytics/yolov5>. [Citado na página 33]
- [45] N. J. H. T. Dwyer, B., “Roboflow,” 2024. Available at: <https://roboflow.com>. [Citado na página 40]
- [46] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023. <https://D2L.ai>. [Citado na página 42]
- [47] K. Santosh, N. Das, and S. Ghosh, “Chapter 2 - deep learning: a review,” in *Deep Learning Models for Medical Imaging* (K. Santosh, N. Das, and S. Ghosh, eds.), Primers in Biomedical Imaging Devices and Systems, pp. 29–63, Academic Press, 2022. Available at: <https://www.sciencedirect.com/science/article/pii/B978012823504100012X>. [Citado na página 42]
- [48] K. You, M. Long, J. Wang, and M. I. Jordan, “How does learning rate decay help modern neural networks?,” 2019. Available at: <https://arxiv.org/abs/1908.01878>. [Citado na página 42]
- [49] U. Glossary, “Stochastic gradient descent (sgd),” 2024. Available at: <https://www.ultralytics.com/glossary/stochastic-gradient-descent-sgd>. [Citado na página 42]
- [50] D. Masters and C. Luschi, “Revisiting small batch training for deep neural networks,” 2018. Available at: <https://arxiv.org/abs/1804.07612>. [Citado na página 42]