

ABSTRACT

Title of Dissertation: **TOWARDS EFFICIENT OCEANIC
ROBOT LEARNING WITH SIMULATION**

Xiaomin Lin
Doctor of Philosophy, 2024

Dissertation Directed by: **Professor Yiannis Aloimonos**
Department of Electrical and Computer Engineering

In this dissertation, I explore the intersection of machine learning, perception, and simulation-based techniques to enhance the efficiency of underwater robotics, with a focus on oceanic tasks. My research begins with marine object detection using aerial imagery. From there, I address oyster detection using Oysternet, which leverages simulated data and Generative Adversarial Networks for sim-to-real transfer, significantly improving detection accuracy.

Next, I present an oyster detection system that integrates diffusion-enhanced synthetic data with the Aqua2 biomimetic hexapedal robot, enabling real-time, on-edge detection in underwater environments. With detection models deployed locally, this system facilitates autonomous exploration. To enhance this capability, I introduce an underwater navigation framework that employs imitation learning, enabling the robot to efficiently navigate over objects of interest, such as rock and oyster reefs, without relying on localization. This approach improves information gathering while ensuring obstacle avoidance. Given that oyster habitats are often in shallow waters,

I incorporate a deep learning model for real/virtual image segmentation, allowing the robot to differentiate between actual objects and water surface reflections, ensuring safe navigation.

I expand on broader applications of these techniques, including olive detection for yield estimation and industrial object counting for warehouse management, using simulated imagery. In the final chapters, I address unresolved challenges, such as RGB/sonar data integration, and propose directions for future research to enhance underwater robotic learning through digital simulation further.

Through these studies, I demonstrate how machine learning models and digital simulations can be used synergistically to address key challenges in underwater robotic tasks. Ultimately, this work advances the capabilities of autonomous systems to monitor and preserve marine ecosystems through efficient and robust digital simulation-based learning.

TOWARDS EFFICIENT UNDERWATER ROBOT LEARNING WITH SIMULATION

by

Xiaomin Lin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:

Professor Yiannis Aloimonos, Chair/Advisor
Associate Professor Behtash Babadi
Professor Dinesh Manocha
Professor Ioannis Rekleitis
Professor Miao Yu

© Copyright by
Xiaomin Lin
2024

Acknowledgments

It is impossible to fully express my gratitude to everyone who played a part in the successful completion of this dissertation. Some of the individuals mentioned here may not even realize the impact they had on my journey, but their contributions have been invaluable and will always be remembered.

On a personal note, I would like to extend my deepest thanks to my advisor, Yiannis Aloimonos. Yiannis has been an exceptional mentor, collaborator, and friend. Beyond his guidance in completing this dissertation, I take immense pride in the work we have accomplished together. I am also incredibly grateful to Miao Yu, whose consistent advice and direction were instrumental throughout this process. Cornelia Fermuller introduced me to the fields of computer vision and robotics, and I owe much of my academic path to her guidance. Additionally, I would like to thank Nare Karapetyan, Nitin Sanket, and Chahat Deep Singh, whose advice and introductions set me on the right path. Without them, I would not have reached this point.

I am also thankful to my friends and collaborators from the Perception and Robotics Group at the University of Maryland for their support and camaraderie.

To my family: my wife Shu Yin, my father Chunxi Lin, my mother Xueqin Shen, and my grandmother Zhouli You—thank you for your constant support throughout my life and for teaching me to be a better person. This dissertation is truly dedicated to you, though I won't let sentimentality overshadow the humor. Finally, a special thank you to my cats, Maki and Mochi, who never fail to remind me that there's more to life than work.

Table of Contents

Acknowledgements	ii
Table of Contents	iii
List of Tables	v
List of Figures	vi
List of Abbreviations	xi
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Challenges and Advances	3
1.2.1 Oceanic Detection	3
1.2.2 Simulation for Robotics	5
1.2.3 Navigation in Marine Environments	7
1.2.4 Water Surface Segmentaiton	10
1.3 Objectives	11
1.4 Contribution	11
1.5 Outline of Thesis	12
Chapter 2: Oceanic object detection	15
2.1 Maritime Object Detection Using Aerial Imagery	15
2.1.1 Introduction	15
2.1.2 Proposed Approach	19
2.1.3 Experiments and Results	22
2.2 Maritime Object Detection Using Satellite Imagery	28
2.2.1 Introduction	28
2.2.2 Proposed Approach	31
2.2.3 Experiments And Results	36
2.3 Marine Object Detection	41
2.3.1 Introductions	41
2.3.2 Synthetic Oyster Generation	43
2.3.3 Experiments And Results	51
2.3.4 Ablation Studies	54
2.4 Edge-based Marine Object Detection	56
2.4.1 Introduction	56

2.4.2	System Overview	58
2.4.3	Experiments And Results	64
Chapter 3:	Marine Information-driven Vision-based Navigation	70
3.1	Introductions	70
3.2	Proposed Approach	72
3.2.1	Intermediate Representation	74
3.2.2	Domain Invariant Navigation Policy	75
3.3	Experimental Evaluation	77
3.3.1	Simulation	78
3.3.2	Results	80
3.3.3	Lab Deployment	81
Chapter 4:	Oceanic Real and Virtual Image Segmentation	85
4.1	Introduction	85
4.2	Underwater Simulator including Dynamic Water-air Interface Distortion	87
4.2.1	<i>AquaSim</i> Simulator	89
4.2.2	Dataset Generation	91
4.3	Real and Virtual Image area Segmentation Pipeline	92
4.3.1	Motion Entropy Kernel	92
4.3.2	Weak Supervision of Epipolar Geometric Consistency	93
4.3.3	Network Architecture: <i>MARVIS</i> Model	94
4.3.4	Objective Function Formulation	96
4.4	Experiments and Results	97
4.4.1	Dataset and Implementation Details	97
4.4.2	Qualitative Evaluation	98
4.4.3	Quantitative Evaluation	100
Chapter 5:	Other Applications	102
5.1	Olive Detection	102
5.1.1	Introductions	102
5.1.2	Synthetic Olive Tree	105
5.1.3	Experiments and Results	113
5.2	Automated Inventory Counting for Warehouse Management	115
5.2.1	Introduction	115
5.2.2	Proposed Approach	116
5.2.3	Result	119
Chapter 6:	Conclusion and future work	122
6.1	Conclusion	122
6.2	Future Work	123
Appendix A:	Permissions and IEEE Copyright Notice	126
Bibliography		127

List of Tables

2.1	The Average Precision (AP) of the training data with various BlueROV image altitudes.	26
2.2	The Average Precision (AP) of the training data with different water colors at an image altitude of 10 m.	26
2.3	The Average Precision (AP) of the training data with different data size at an image altitude of 10 m of Brown water color	27
2.4	Comparison of Semantic Segmentation Results with the Two Different Networks	39
2.5	Comparison of Semantic Segmentation Results with the State-of-the-art.	54
2.6	Ablation Studies Of The Proposed Oyster Geometric Model (Also see Fig. 2.18).	54
2.7	Quantitative Comparison of All YOLOv10 Sub-Model Performance	66
2.8	Comparison of YOLOv10-B, YOLOv10-L, and YOLOv10-X Models with(S) and without(R) Synthetic Data. (S): the model is trained on combined synthetic and real data. (R): the model is trained on real data only	69
4.1	Comparison of the different marine robotics simulators showing <i>AquaSim</i> excels in rendering near-water surface	90
4.2	Quantitative comparison for real-virtual image segmentation performance by <i>MAR-VIS</i> and other SOTA water segmentation pipelines and widely used image segmentation models on both synthetic and real-world test sets.	101
5.1	IoU using real only, synthetic/real, and synthetic IGA/real	115
5.2	F1 Score and Accuracy Mean (using images with 5-50 parts)	121

List of Figures

1.1	BlueROV(in the middle) collecting data in a oyster farm in Olympics, WA	2
1.2	BlueROV collecting oyster data in Horn Point Lab	3
2.1	(a) Typical image examples with varying altitudes, angles of view and background water color. (b) Corresponding ground truth masks for the object of interest (BlueROV).	16
2.2	An overview of our approach: 3D models for the object of interest are used in <i>SeaDroneSim</i> to generate Synthetic datasets. Noted the synthetic dataset would include its ground truth mask for the object of interest. The synthetic dataset generated is then fed into a Neural network to obtain the object detection result. Note: Object Detection images are cropped and enlarged for better visualization.	19
2.3	(a) BlueROV in water near Horn Point Lab, (b) BlueROV in Blender™, (c) BlueROV in water in Horn Point Lab, (d) BlueROV in Blender™.	20
2.4	Different water colors and turbidity levels.	21
2.5	Images from different altitudes. (a) Image from an altitude equal to 20 m, (b) image from an altitude equal to 30 m, (c) image from an altitude equal to 40 m, (d)(e)(f) are the corresponding ground truth masks for (a)(b)(c).	23
2.6	Examples of original and ground truth mask images. These examples are based on different background water colors brown (a, d), blue (b, e), and green (c, f), respectively.	24
2.7	The first row is the examples of testing images. The second row is the testing results for the above images. These examples are collected from Sandy Point Park in Annapolis, MD.	24
2.8	First row presents real whale images taken from space while the second row is the simulated whale images from using <i>SeaDroneSim2</i>	28
2.9	An overview of our approach. (a) Assets: Loads the assets such as water properties, objects, materials, etc. into <i>SeaDroneSim2</i> to generate Synthetic datasets. Note, the synthetic dataset would include its ground truth mask for the object of interest. (b) We modify the properties within the scene such as noise level, rotation of the object, altitude of the camera and etc. (c)The synthetic dataset generated is then fed into a Neural network to obtain the object detection result. We demonstrated the generation of aerial, satellite, and underwater images for two different objects of interest in our study. Note: Object Detection images are cropped and enlarged for better visualization.	29

2.10	These are the synthetic images generated from <i>SeaDroneSim2</i> . In the first row, the first two images showcase the increasing turbidity of the water, and the last two images depict varying lighting conditions. In the second row, the first two images display different watercolors, while the last two images exhibit increasing noise from the satellite images. In the third row, the first two images demonstrate varying altitudes, while the last two images illustrate different whale positions, including lodging, spyhopping, and submerging. In the last row, the first two images demonstrate synthetic images with different water waves. while the last two images illustrate different rocks and hills.	33
2.11	From left to right: Sample real input image, ground truth, segmentation result using Unet without synthetic augmented real data, segmentation result using Unet with synthetic augmented real data, segmentation result using FPN without synthetic augmented real data, segmentation result using FPN with synthetic augmented real data. All networks here are trained with only 10% of real data.	36
2.12	Each row left to right: Input image, the output of the network when trained using only real data, the output of the network (which we call <i>OysterNet</i>) when trained using real data augmented with our synthetic data. Yellow represents the oyster segmentation ground truth and the blue is the predicted segmentation result. Notice how the number of false positives and false negatives drops significantly when the training data is augmented with our synthetic data, <i>All the images in this paper are best seen in color on a computer screen at 200% zoom.</i>	42
2.13	An overview of our approach: The proposed geometric model is used to generate synthetic images which are further fed into a Generative Adversarial Network to enable sim-to-real transfer (domain adaptation) by generating photorealistic oyster images. We then combine the synthetic data with real data to train a <i>OysterNet</i> for oyster detection.	43
2.14	Steps in the geometric modeling of an oyster: (a) Sample image of a real oyster shell, (b) 3D scan of a real oyster, (c) Splines fit to the oyster's bottom layer to model it (each color represents a single spline), (d) Simplified model of one stratified layer of the oyster (single layer of spline curve $S(t)$), (e) all layers of $S^\alpha(t)$, (f) generated 3D model of oyster, (g) final generated 3D model of oyster with added real oyster texture.	44
2.15	(a) One of the synthetic models generated from Sec. 2.3.2-A), (b) Synthetic model with real oyster texture added, (c) image of an oyster farm with 50 synthetic oysters generated in Blender™, (d) masks for oysters in (c).	47
2.16	Domain Adaptation to perform sim-to-real transfer. (a) A single sample of the simulated oyster, (b) A single sample of the real oyster, (c) Simulated oyster farm, (d) Synthetic oyster farm domain adapted to real world for photorealism.	48
2.17	BlueROV collecting data for our real dataset. The inset shows a sample image captured which is a part of our real dataset.	52
2.18	Qualitative demonstration of how different parameters of our oyster model affect the shape of the oyster. The parameters are in the same order as the experiments in Table 2.6.	55

2.19	Oyster detection system deployed on an Autonomous Underwater Vehicle (AUV). The main image shows oyster detection in real-time in a shallow marine environment on the Aqua2 robot. The inset highlights the output bounding boxes around the oysters detected in situ	56
2.20	Overview of the oyster detection system. Simulated oyster assemblages are rendered using Blender and paiblu with ground-truth segmentation masks. Diffusion models are employed to generate synthetic oyster imagery, ensuring consistency with real oyster habitats. Real-world data are collected by the Aqua2 and BlueROV, equipped with a camera for capturing oyster images in the field. The combined dataset is used to train a YOLOv10, which is deployed on an edge computing platform on Aqua2 for real-time oyster detection.	59
2.21	Experimental deployment of the system, surveying a wild near-shore oyster reef in Lewes, DE. Operators can be seen teleoperating the Aqua2 and analyzing the feedback.	62
2.22	Synthetic image generation pipeline using Stable Diffusion and ControlNet. ControlNet uses randomly sampled real underwater images combined with Blender-generated images, depth images, and ground truth masks to ensure consistency. Stable Diffusion model, guided by text prompts, and ControlNet, refines the synthetic data to closely match real-world oyster environments for vision model training.	63
2.23	Qualitative Comparison of All YOLOv10 Sub-Model Performance. <i>This image is best seen in color on a computer screen at 400% zoom.</i>	67
2.24	Qualitative Comparison of YOLOv10-B, YOLOv10-L, and YOLOv10-X Models with and without Synthetic Data. <i>This image is best seen in color on a computer screen at 400% zoom.</i>	68
3.1	Example of two trajectories of BlueROV surveying oyster-reef in a pool. On the bottom left is the current frame in the intermediate representation that the robot observes.	71
3.2	An overview of our approach. UIVNAV consists of two phases: Generating Intermediate Representation (gray box) and Data Invariant Navigation Policy (orange box). Image (I_D) is fed into the depth estimation network and the segmentation network. The output from the depth estimation network (I_D) and the output from the segmentation network (I_S) are used to generate the intermediate representation (I_{DS}). This is fed into an imitation learning network to train a navigation policy that predicts yaw and pitch changes. Due to training on (I_{DS}) the navigation policy is domain invariant and will not require restraining nor labeling with new OOI.	73
3.3	Example of BlueROV surveying reef of oysters in simulation (translucent mask is applied on the oyster reef for better visualization.)	77
3.4	Comparing various exploration strategies across four oyster reef patterns– (a) Grid world, (b) E-shape, (c) Disconnected Paths, (d) Branching Corridor.	79

3.5	UIVNAV surveying reef of rocks OOI with no additional human-expert labeling or training. (a) The current frame of the robot field of view in OysterSim simulation. (b) Intermediate representation of the current frame that the robot uses to make navigation decisions. (c) The complete trajectory over a rock reef.	79
3.6	Quantitative Evaluation of Different Strategies on Different Environments. BB method travels longer distances moving on fewer areas with oysters compared to UIVNAV. Even though compared to BCD, the UIVNAV method collects on average 29% less oyster data it travels 70% less distance. This shows that UIVNAV trajectories survey on average 36% more oysters than the BCD method when traveling the same distances.	80
3.7	(a) BlueROV2 Deployed in a Neutral Buoyancy Pool with Yaw and Pitch control with E-shape oyster shells distributed at the bottom. (b) BlueROV's field of view.(c) Experimental result of two trajectories of BlueROV surveying oyster shells in a pool.	81
4.1	Various computer vision tasks fail to varying degrees in multi-media scenarios due to virtual images formed by the diverging reflection or refraction of light rays. The real-virtual image segmentation mask is crucial to provide the robot with credible region information in the visual sensor input.	86
4.2	BlueROV collecting data for our synthetic dataset in <i>AquaSim</i> . The inset shows a sample image captured from the camera on a BlueROV. The top left displays the feed from a front-facing RGB camera, while the bottom left shows the real image's corresponding ground truth masks.	88
4.3	Full learning pipeline of <i>MARVIS</i> : initially, two consecutive grayscale video frames (at time $t - \delta t$ and t) are fed into optical flow estimation module [1]. This motion information is forwarded to the proposed Local Motion Entropy (LME) layer for LME feature extraction. The fused spatio-temporal feature maps are then passed to a five-stage encoder to obtain features in high-dimensional latent space. Subsequently, the five-stage decoder progressively upsamples by fusing high-resolution feature maps from the corresponding stage encoder through skip connections to generate the segmentation mask with the original resolution. The learning objective involving pixel-wise losses (EGC loss) is given by Eq. 4.3	88
4.4	The detailed operations in the Attention Module and the Tokenized Block (also described in Eq. 4.4).	95
4.5	A few qualitative comparisons are shown for real-virtual image segmentation by <i>MARVIS</i> and SOTA water segmentation pipelines and widely used image segmentation models on both synthetic and real world testset. As seen, <i>MARVIS</i> infers accurate and consistent segmentations across the samples from different domains.	96
5.1	Each column left to right: Input image, pblueiction using only real data, pblueiction using real and synthetic data, pblueiction using real and synthetic data in IGA. Pblueictions are shown in orange and ground truths are shown in blue. Adding synthetic data to the training set increases the number of correct pblueictions.	103

5.2	An overview of our approach: The proposed geometric model is used to generate synthetic images which are further fed into a VSAIT to enable image-to-image translation. We then combine the synthetic data with real data to train an Unet for olive detection.	103
5.3	(a) the 3D modeling of our ellipsoid. (b) the 3D modeling of an olive (c) an example olive rendeblue with texture.	106
5.4	Fig. 5. (a) a single bezier curve with 2 control points. (b) the bezier curve we use as a branch. (c) the bezier curve with 3D-scanned leaves scatteblue onto it.	108
5.5	Image-to-image translation (a) A single sample of the simulated olive, (b) A single sample of the real olive, (c) Simulated olive image, (d) Synthetic olive image translated into real world for photorealism.	111
5.6	Trays for the background and the real experiment.	117
5.7	Real Experimental setup	119

List of Abbreviations

UAVs	Unmanned Aerial Vehicles
SAR	search-and-rescue
GPU	Graphic processing unit
GAN	Generative Adversarial Network
NASA	National Aeronautics and Space Administration
AP	Average Precision
mAP	mean Average Precision
IR	Intermediate Representation

Chapter 1: Introduction

1.1 Motivation

Oceans are a vital component of Earth's ecosystem, covering over 70% of the planet's surface and supporting an incredible diversity of life. They regulate the climate, provide oxygen, and sustain food chains that billions of people rely on. With approximately 37% of the global population living within 100 kilometers of an ocean [2], the health of marine ecosystems is critical to human survival. Key ecosystems, such as coral reefs and oyster beds, play a particularly important role in maintaining biodiversity, filtering water, and protecting coastlines from erosion. Additionally, aquaculture—the farming of aquatic organisms such as fish, shellfish, and algae—has emerged as a critical industry for food production and economic sustainability. However, both natural ecosystems and aquaculture face growing challenges from overfishing, pollution, and climate change, underscoring the need for advanced technologies to ensure their sustainable management and monitoring.

Addressing the complexity of oceanic environments and the challenges of both natural ecosystems and aquaculture requires efficient, scalable, and non-invasive monitoring methods. Underwater robotics, combined with advanced learning techniques and digital simulation, offers an innovative solution for sustainable aquaculture and environmental conservation. By using digital simulations to train underwater robots, we can enhance their efficiency in exploring, monitor-



Figure 1.1: BlueROV(in the middle) collecting data in a oyster farm in Olympics, WA

ing, and maintaining these aquatic systems. Recent advances in robotic sensing and navigation, coupled with machine learning, allow us to overcome technical limitations that once hindered marine exploration and aquaculture management. As demand for autonomous solutions rises, the development of fully autonomous oceanic robots will be essential to reduce operational costs, mitigate operator fatigue, and ensure safety in these challenging environments.

This dissertation advances the field of underwater robotics by focusing on efficient robot learning through digital simulation. By integrating cutting-edge machine learning techniques and simulation-based training, we aim to enhance the detection and monitoring capabilities of underwater robots for both maritime (above water) and marine (underwater) objects, including critical aquaculture sites. This approach not only improves the robots' operational efficiency but also contributes to the broader goal of preserving and restoring marine ecosystems and supporting sustainable aquaculture practices. Through these advancements, this work supports the creation of more effective, autonomous underwater systems for oceanic exploration, aquaculture

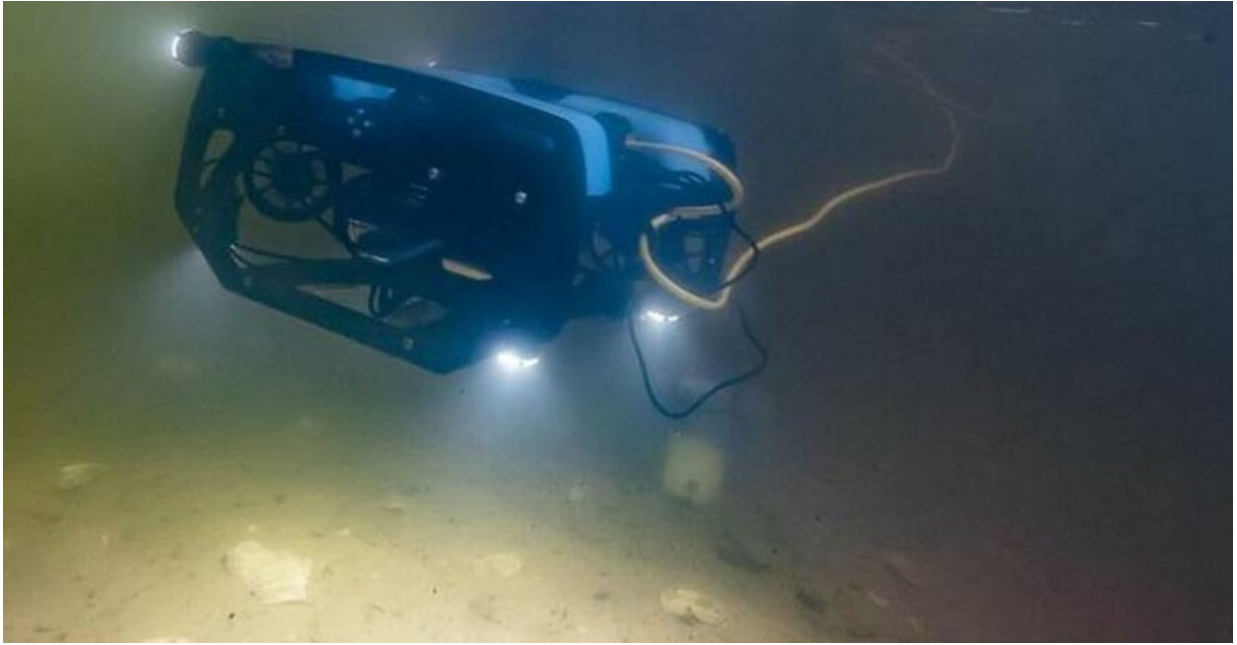


Figure 1.2: BlueROV collecting oyster data in Horn Point Lab

management, and environmental conservation.

1.2 Challenges and Advances

In this dissertation, we use "maritime" to refer to activities and environments above the water, "marine" to describe those under the water, and "oceanic" to encompass both maritime and marine contexts. These distinctions are important when discussing detection challenges, as both above-water (maritime) and underwater (marine) environments present unique obstacles for object recognition and monitoring tasks.

1.2.1 Oceanic Detection

The development of advanced computer vision algorithms necessitates access to extensive datasets, particularly in oceanic environments. Existing datasets predominantly center around

synthetic aperture radar (SAR) satellite imagery for remote sensing tasks, but the focus is shifting toward Very High-Resolution (VHR) images for object detection in maritime and marine settings [3, 4]. Noteworthy contributions include Gallego’s [5] method for autonomous ship detection using aerial images, and Li’s [6] dataset consisting of Google Earth and UAV-based images for ship detection. However, many existing datasets have limitations in object variety, as seen in the work of Lygouras et al. [7], whose UAV-based images focus on human detection but are constrained to near-shore environments and swimming pools.

In whale monitoring, Very High-Resolution (VHR) satellite images are gaining attention as an alternative to traditional ship-based or acoustic methods [8–10]. Deep neural networks have played a pivotal role in analyzing these images, with several studies applying them to whale detection tasks [3, 9, 11]. For example, Kiefer et al. [12] analyzed real and synthetic maritime and terrestrial images for detecting boats and people, while Varga et al. [13] released a UAV-based dataset for water object recognition. Although these datasets provide valuable insights, they are often limited by the number of object classes they cover, such as Varga’s dataset, which focuses on only six classes of water objects.

Collecting and annotating sufficient data for oceanic environments remains a major challenge, especially for underwater tasks like oyster detection, where visibility is reduced, and manual labeling is labor-intensive [14]. Furthermore, these datasets often lack the environmental variety necessary for robust object detection in real-world scenarios.

To address these issues, researchers have increasingly turned to synthetic data generation. By using 3D modeling to simulate oceanic environments, large-scale datasets can be created to train vision models. This approach has been successfully applied in tasks such as ship detection [12] and whale monitoring [8], as synthetic data allows for diverse conditions and object ap-

pearances that improve detection models' robustness. Generative Adversarial Networks (GANs) have also been applied to sim-to-real tasks such as underwater image restoration [15] and AUV pose estimation [16], but their reliance on large ground-truth datasets can limit their applicability in complex marine environments.

Recent advancements in diffusion models [17, 18] provide a promising alternative. These models generate high-quality synthetic images and offer greater flexibility and control over data augmentation, improving detection performance in both maritime and marine environments. For example, Boulent et al. [19] proposed a human-in-the-loop AI-assisted annotation tool that combines automation with expert input, demonstrating the potential for deep learning to enhance detection accuracy and efficiency, particularly in whale monitoring.

In summary, the key challenges in oceanic detection tasks include the lack of diverse, large-scale datasets, poor visibility in underwater environments, and the difficulty of manual data annotation. To overcome these obstacles, researchers are leveraging synthetic data generation, GANs, and diffusion models to improve detection accuracy and robustness across both maritime and marine contexts.

1.2.2 Simulation for Robotics

Simulation environments are a critical part of developing and testing robotic capabilities, especially when operating in complex and hazardous maritime and marine environments. Simulations enable data collection and testing of robotics applications in controlled, repeatable scenarios, reducing the risks and costs associated with real-world trials.

In maritime environments, simulators like RarePlanes [20] have used synthetic data to de-

tect airplanes in very high-resolution (VHR) satellite images, while other simulators have focused on drone control for safety operations [21] and rapid control [22]. Some simulators, such as those developed by Abujob [23], have been used to verify algorithms for landing drones on ships using motion prediction. While UAV simulators like the Matlab UAV Toolbox [24] exist, they focus mainly on terrestrial applications and lack ground truth segmentation for maritime object detection.

In marine environments, simulation plays an even more crucial role. With challenges such as poor visibility, complex terrain, and the need for real-time decision-making, simulations are often the only feasible way to test robotics applications. Zwiilmeyer et al. [25] used Blender [26] to generate marine datasets containing ground truth and sensor data. By creating physical models of vehicles and control systems, they simulated realistic paths in virtual marine environments, including scenes with sandy bottoms, vertical rocks, and occlusions. Although Blender provides robust capabilities for marine simulation, other platforms have also emerged to meet more specific needs.

For instance, UUV Simulator [27], based on ROS and Gazebo, accurately models marine physics and includes a prototype sonar implementation. However, it requires complex ROS setups and has limitations in sonar modeling, as the simulated depth camera's range does not match that of real imaging sonars. Similarly, UWSim [28], based on OpenSceneGraph [29], provides open-source support for multiple agents and sonar modeling, though it is dependent on ROS and is no longer actively maintained.

Several newer simulators offer high-fidelity rendering and better marine dynamics. HoloOcean [30], built on Unreal Engine 4 (UE4), supports multiple agents and various marine sensors. It provides highly realistic marine environments, building on the capabilities of Holodeck [31],

which also offers advanced physics simulations via the PhysX engine. MARUS [32] and UNav-Sim [33] have improved realism further, but they still face challenges when it comes to integrating essential robotics tools, such as more sophisticated robotic controls and sensor models.

While these simulation environments excel in deep marine tasks, they often lack flexibility for handling real and virtual image segmentation across water surfaces. Despite the progress made by these simulators, there is still a need for more comprehensive solutions that combine high-fidelity rendering, realistic dynamics, and integrated robotics tools. Simulations like Isaac-Sim [34] and AirSim [35] have advanced high-fidelity rendering for aerial vehicles, and developments like DAVE [36] aim to fill the gap in marine simulations, although they face rendering limitations.

In summary, simulations are essential for developing and testing robotic systems in oceanic environments. From high-fidelity rendering to sonar modeling and real-time decision-making capabilities, simulation platforms like Blender, UUV Simulator, Holodeck, HoloOcean, and MARUS provide critical tools for advancing robotic capabilities in complex, hazardous environments. The integration of cutting-edge simulation techniques continues to advance the development of autonomous robotic systems for exploration and monitoring in both maritime and marine settings.

1.2.3 Navigation in Marine Environments

The problem of underwater navigation has been extensively studied in the field [37], as it is crucial for a variety of applications, including underwater exploration, scientific sampling, and underwater photogrammetry. Typically, navigation is treated as an optimization problem, where

the objective is to find an optimal path that minimizes a predefined cost function, such as obstacle avoidance [38] or energy efficiency [39]. Classical planning methods have also been adapted for the underwater domain. For example, Xanthidis et al. [40] augmented a classical optimization package designed for manipulators to perform 3D trajectory planning for underwater vehicles, using either a known map or an online-constructed local map. To further enhance this, they introduced AquaVis [41], an active perception system that identifies visual goals in real time, moves towards the target location, and adjusts its position to optimize the view of nearby visual goals. While these methods have shown promising results, they rely on precise localization to be effective.

Several vision-based navigation systems have been developed to address the challenges of operating in environments with limited visibility. Pérez-Alcocer et al. [42] developed a robust vision-based navigation system for marine environments with poor visibility. Manzanilla et al. [43] used a vision-based localization approach coupled with an Extended Kalman Filter (EKF) and a proportional-integral-derivative (PID) controller to enable autonomous navigation for an underwater vehicle. Bobkov et al. [44] proposed an adaptive method using stereo camera data to enhance navigation performance. These approaches integrate planning with state estimation, but they primarily focus on improving navigation accuracy rather than optimizing data collection.

Few attempts in the literature have addressed the problem of efficient underwater coverage and data acquisition in 3D space. Extending classical coverage methods, Galceran et al. [45] proposed segmenting the environment by depth, and in later work [46], accounted for state estimation uncertainties. Similarly, Vidal et al. [47] proposed a next-best-view approach, but it was constrained by the certainty of state estimation. These methods, while advancing the field of 3D exploration, are often limited by prior knowledge of the environment (such as maps) or by

the accuracy of the state estimation. In some cases, a simplified problem formulation has been adopted, such as the 2D mapping of underwater structures [48].

To overcome the limitations of relying on state estimation, Manderson et al. [49] and Karapetyan et al. [50] proposed end-to-end, vision-based approaches for navigating highly unstructured environments, such as coral reefs and shipwrecks. These methods use supervised imitation learning, where a human diver labels image sequences with actions (such as yaw and pitch changes) to drive the robot toward the target object while avoiding obstacles. Manderson et al. [51] further extended this work to allow user-defined goal conditions, and Liu et al. [52] adapted it to navigate confined areas. However, each of these approaches is tailored to a specific region of interest (ROI), requiring new labeling and retraining for navigation over different environments.

In summary, a variety of methods have been developed for path planning and navigation in marine environments, each with varying degrees of reliance on prior knowledge and the complexity of the environment. The challenge lies in creating a general solution that can efficiently navigate unknown environments, avoid obstacles, and maximize data collection without requiring extensive retraining. Our work is inspired by human diver-driven coverage and navigation approaches [50, 53, 54]. We introduce a novel domain-invariant navigation policy that reduces the need for retraining and manual labeling, offering a more generalized solution that can be employed with or without precise localization, in combination with global planning methods or as a standalone approach.

1.2.4 Water Surface Segmentation

Several recent efforts have investigated sea-sky line inference and water obstacle detection for unmanned surface vehicles (USVs). For example, Chen et al. [55], Bovcon et al. [56], Steccanella et al. [57], and Zhou et al. [58] explored these topics using the Marine Semantic Segmentation Training Dataset (MaSTr1325). However, these approaches are limited to segmenting the sea-sky boundary and detecting water obstacles, without addressing the more complex task of segmenting real and virtual image regions near water surfaces.

Vandaele et al. [59], Muhadi et al. [60], and Wagner et al. [61] worked on automated river level monitoring, but their methods do not leverage information across consecutive frames. This lack of temporal integration can reduce generalization performance, particularly in tasks involving surface segmentation from viewpoints beneath the water. Our proposed method addresses these gaps by functioning both above and beneath the water surface, effectively utilizing temporal cues from consecutive frames to improve segmentation performance.

Unlike standard water-air interface segmentation techniques, which focus on delineating the boundary between water and air, real-virtual image segmentation requires prioritizing temporal motion and geometric details. These factors are critical in marine robotics, where egomotion, proximity to water surfaces, and a first-person robotic viewpoint introduce challenges in distinguishing between real and reflected objects.

Liang et al. [62] developed WaterNet, which utilizes temporal cues from consecutive frames to segment volatile water. However, WaterNet relies on precise initial segmentation and is sensitive to camera and object motion, which can degrade its performance in dynamic environments.

1.3 Objectives

Within this dissertation, we explore the application of advanced machine learning techniques and synthetic data generation in oceanic tasks, focusing on the detection, navigation, and monitoring of both marine and maritime environments. We investigate the development of simulation frameworks, such as SeaDroneSim and SeaDroneSim2, to generate synthetic datasets for training object detection models in maritime environments. Additionally, we introduce Oysternet and ODYSSEE, leveraging sim-to-real transfer techniques to augment real datasets, and deploy ODYSSEE on edge devices for efficient object detection and environmental monitoring. The thesis also presents UIVNAV, an imitation learning-based navigation system enabling autonomous underwater navigation without the need for precise localization. Furthermore, we address the challenges of real-virtual image segmentation in marine environments through MARVIS, enhancing real-time segmentation near water surfaces. Overall, this dissertation advances oceanic robotics by integrating machine learning, simulation, and sensor fusion to improve efficiency and accuracy in underwater exploration and monitoring.

1.4 Contribution

This thesis introduces a series of contributions aimed at advancing the domain of deep learning algorithms within the context of underwater environments. These contributions can be delineated into the following comprehensive categories:

- Developed SeaDroneSim and SeaDroneSim2, novel simulation platforms for generating synthetic datasets for UAV-based object detection in maritime and very high-resolution

remote sensing environments.

- Demonstrated the feasibility of augmenting real datasets with synthetic data using sim-to-real techniques through frameworks like Oysternet and ODYSSEE, significantly improving detection performance in challenging oceanic environments, such as oyster reefs.
- Introduced UIVNAV, an imitation-learning-based navigation framework that uses an intermediate representation to navigate without prior localization. Demonstrated how UIVNAV improves data collection efficiency and obstacle avoidance in both simulated and real environments.
- Developed MARVIS, a real/virtual image segmentation model that enhances navigation near water surfaces by distinguishing between real and virtual images. Demonstrated improved segmentation performance using temporal motion and geometric cues, validated in both simulation and real-world experiments.
- Extended simulation-based methods for object detection to new domains, including olive detection and industrial object counting.

1.5 Outline of Thesis

In Chap. 1, we introduce the motivation behind this research, focusing on the application of machine learning and simulation-based techniques to enhance underwater robotic systems. This chapter also includes a comprehensive review of the related work in underwater robotics, object detection, and the use of simulation-based data augmentation methods. We emphasize the challenges faced in oceanic exploration and object detection, particularly the role of synthetic

data generation and simulation environments in improving the performance of detection models for marine objects. The chapter provides an overview of the current state-of-the-art in underwater object detection and lays out the structure of the dissertation.

In Chap. 2, we present our work on maritime and marine object detection, beginning with the detection of BlueROV using aerial images captured by drones. Leveraging these aerial views, we demonstrate how combining marine object detection techniques with drone imagery can significantly improve detection rates in shallow water environments. Building on this, we extend our approach to remote sensing at a larger scale by utilizing satellite images for whale detection, showcasing the versatility of our methods in tracking marine species from higher altitudes. Following the transition from aerial to satellite-based detection, we shift our focus underwater to marine object detection, specifically addressing oyster detection. We introduce Oysternet, a system that leverages simulated data and Generative Adversarial Networks (GANs) to enhance detection accuracy. Using a sim-to-real transfer technique, synthetic datasets are generated to train the oyster detection model, and the results are evaluated through real-world testing. This approach demonstrates substantial improvements over traditional oyster detection methods in both accuracy and real-time performance.

In Chap. 3, we discuss the underwater navigation framework, UIVNAV, which uses imitation learning to navigate over objects of interest (OOIs), such as oyster and rock reefs, without relying on traditional localization methods. This chapter examines how UIVNAV improves information gathering efficiency while avoiding obstacles, contributing to autonomous exploration in shallow water habitats.

In Chap. 4, we explore the challenges of navigating near water surfaces in shallow environments. MARVIS, a deep learning-based model, is introduced to perform real/virtual image

segmentation, distinguishing between actual objects and reflections on the water's surface. This technique ensures safe navigation for underwater robots in complex environments. The chapter provides a detailed analysis of the model's performance and its integration into the overall system.

In Chap. 5, we broaden the scope of our research by discussing additional applications of the techniques developed throughout the dissertation. These include olive detection for yield estimation and industrial object counting for warehouse management, both of which utilize simulated imagery. The chapter highlights how the frameworks developed for underwater robotics can be adapted to solve problems in agricultural and industrial domains, further demonstrating the versatility of the simulation-based approaches.

In Chap. 6 concludes the dissertation by addressing unresolved challenges, such as the integration of RGB and sonar data for enhanced detection in underwater environments. Directions for future research are proposed, with a focus on improving data fusion techniques and extending the application of machine learning models to more complex underwater environments.

By way of summary, at the end of the activity, I reminded the class of what we'd done: by considering relatively nearby galaxies whose distance we had measured by some other means, we were able to establish a relationship locally between redshift and distance.

Chapter 2: Oceanic object detection

2.1 Maritime Object Detection Using Aerial Imagery

2.1.1 Introduction

The BlueROV, a widely used open-source remotely operated underwater vehicle (ROV), is central to our oceanic tasks such as data collection [63, 64], exploration [65], and 3D water quality mapping [66]. However, underwater position estimation errors can cause it to lose track of its location. To enhance efficiency, coupling the BlueROV with Unmanned Aerial Vehicles (UAVs) offers a powerful solution for multi-robot operations. UAVs, equipped with high-resolution cameras and GPUs, provide versatile support in marine applications like vessel monitoring, search-and-rescue (SAR), aquaculture management, and underwater surveys. Detecting the BlueROV from the UAV’s perspective near the water surface is essential for coordinating underwater operations, such as oyster reef mapping and marine environmental monitoring, by enabling real-time tracking and planning.

Modern computer vision algorithms that detect objects like the BlueROV in dynamic environments require large, high-quality datasets, which are often time-consuming and labor-intensive to collect. To tackle this, we introduce SeaDroneSim, a benchmark suite that generates photo-realistic aerial images and ground truth segmentation masks for object detection.

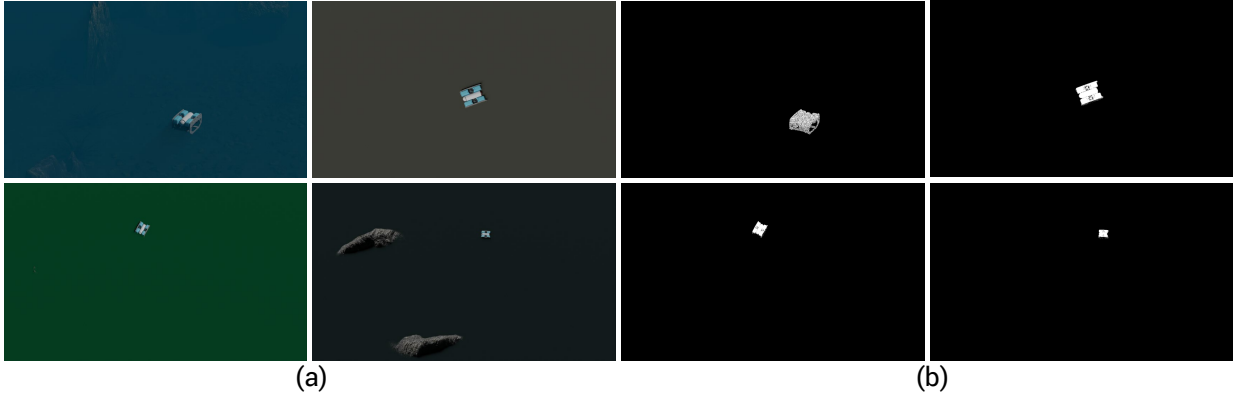


Figure 2.1: (a) Typical image examples with varying altitudes, angles of view and background water color. (b) Corresponding ground truth masks for the object of interest (BlueROV).

Using synthetic data from SeaDroneSim, we achieved a 71 mean Average Precision (mAP) in detecting the BlueROV in real aerial images. These results provide a baseline for improving BlueROV detection in oceanic tasks, enhancing precision aquaculture, and facilitating marine exploration through UAV-assisted monitoring.

Unmanned aerial vehicles (UAVs) equipped with high-resolution cameras and Graphics Processing Units (GPUs) are increasingly used in a variety of tasks, ranging from land-use classification [67, 68], precision agriculture [67, 69, 70], coastal management [71], crowd surveillance [72, 73], and recently in assisting with search and rescue (SAR) missions [74–78] and post-disaster exploration missions by providing a bird’s eye view over the scene. Specifically, in marine scenarios, where a large ocean area needs to be searched quickly, UAVs provide vital aid to emergency search and rescue operations [79]. For SAR missions, a robust vision-based system is needed to locate/track objects of interest in maritime environments with different lighting conditions, image altitudes, viewing angles, and water colors. Currently, most of these are vision-based, data-driven systems employing deep neural networks. These systems are dependent on large datasets to correctly recognize objects under diverse environmental conditions.

However, there are only a few datasets available, which are limited in their size and variety of objects of interest. The process of collecting large-scale datasets for specific object detection tasks is slow, time-consuming, and has poor scalability. Using a manual approach, only a few objects of interest can only be located/tracked within a specific dataset for that task. Varga [13] proposed a relatively large dataset for the Maritime environment, yet with a nearly infinite number of potential objects of interest there is a need for methods to rapidly generate large-scale datasets in maritime environments for a variety of objects, including our object of interest, the remotely operated underwater vehicle - BlueROV Fig. 2.3.

Similar to Kiefer et al. [12], this work aims to help fill the gap of large-scale maritime-based datasets captured from UAVs. We introduce a simulation suit for generating simulated images of maritime-based aerial images, called *SeaDronesSim*. We are capable of generating videos and images of any object of interest with a 3D model Fig. 2.3 in open water with various lighting conditions, image altitudes, viewing angles, and background water colors (see Fig. 2.1(a)). To fully utilize the *SeaDronesSim*, the ground truth for the object of interest is generated at the same time (see Fig. 2.1(b)). Most SAR missions require a vision-based system to be able to detect and track from a great distance. *SeaDronesSim* is able to vary its resolution for the blue, Green, and Blue (RGB) footage captured. To study the feasibility of object recognition purely from synthetic images generated from *SeaDronesSim*, we used our object of interest, the BlueROV, for which we have a detailed 3D computer model on hand for simulating various image poses from Patrick [80]. Moreover, the simulation provides all the ground truth information of the aerial camera (altitude taken, rotation) and the BlueROV (rotation), allowing us to output all the metadata for the aerial camera and the objects of interest. We believe all this metadata could be used in the future to develop multi-modal systems to improve accuracy and speed. Finally,

we conducted extensive experiments to compare the datasets that we generated using a state-of-the-art model and established baseline models for BlueROV detection. These results serve as a starting point for our *SeaDronesSim* simulation suit. To streamline the process of object detection, our goal is to utilize advancements in robotics and artificial intelligence that enable us to gather UAV-based aerial images from the maritime environment to automate the process of object detection and tracking. *SeaDronesSim* is among the first simulations for UAV-based aerial images from the maritime environment [12]. Our main contributions are as follows:

- We propose a novel simulation suit for generating UAV-based aerial images for the maritime environment.
- To the best of our knowledge, we established a baseline for detecting BlueROV in open water.
- We open-source ¹our *SeaDronesSim* and dataset associated with this work to accelerate further research.
- We proposed a pipeline for autonomously generating UAV-based maritime images for objects of interest and detecting them.

With the ultimate goal of developing an autonomous UAV-based surveillance system, we acknowledge the need for object detection methods. Moreover due to the lack of large-scale datasets for the maritime environment and the lack of literature addressing this problem, we are seeking an alternative path for generating maritime datasets through synthetic image generation. To the best of our knowledge, we are among the first to propose a simulator (*SeaDroneSim*) for mar-

¹<https://github.com/prgumd/SeaDroneSim>

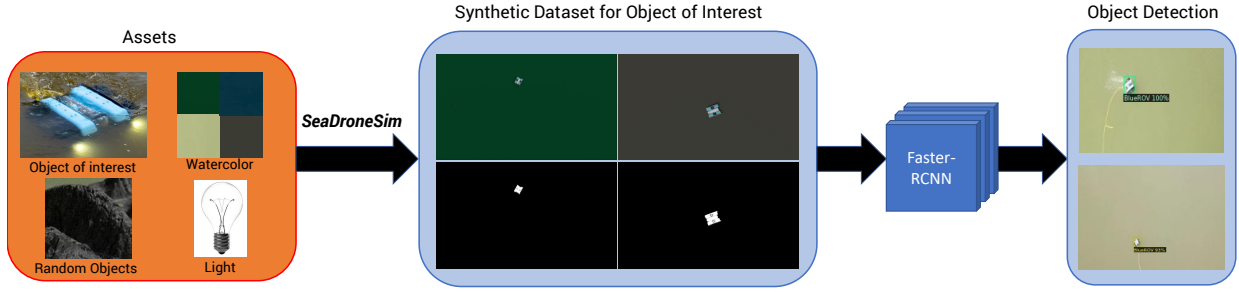


Figure 2.2: An overview of our approach: 3D models for the object of interest are used in *SeaDroneSim* to generate Synthetic datasets. Noted the synthetic dataset would include its ground truth mask for the object of interest. The synthetic dataset generated is then fed into a Neural network to obtain the object detection result. Note: Object Detection images are cropped and enlarged for better visualization.

itime environment datasets and use that data to recognize objects of interest. *SeaDroneSim* will be described in the next Section.

2.1.2 Proposed Approach

In this work, we propose a novel simulation built from BlenderTM [26] game engine. Then we used the simulation for creating UAV-based maritime images which are then used to train a Neural Network for object detection. As we can see in Fig. 2.2), we first choose our object of interest and different assets(i.e. water color, light, random objects) and input them into the simulation engine that we build. Moreover, utilizing the data produced from the simulation engine, we can come up with a detection network for detecting objects of interest. In this section, we will go through some of the details of the implementation of the *SeaDronesSim*.

2.1.2.1 Object of Interest

It is critically important to have an accurate 3D model of the object for this work. Only with an accurate 3D model of the object in the simulation, can we correctly represent the visual appearance from the UAV-based maritime images. In this work, we select BlueROV as our object

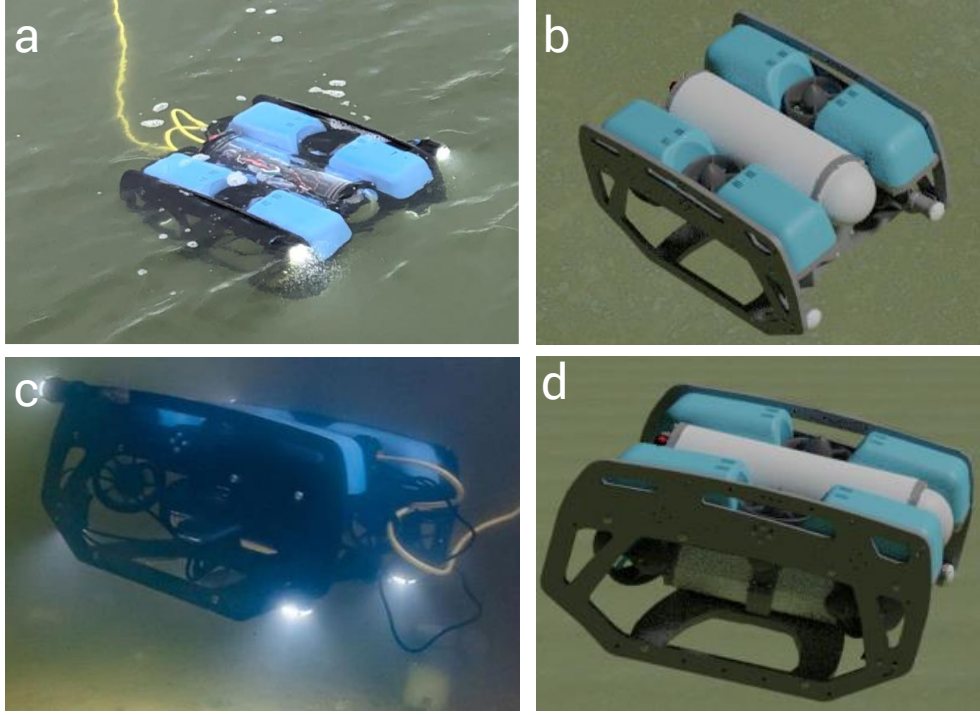


Figure 2.3: (a) BlueROV in water near Horn Point Lab, (b) BlueROV in BlenderTM, (c) BlueROV in water in Horn Point Lab, (d) BlueROV in BlenderTM.

of interest because of the availability of its 3D model [80] and actual object for collecting real datasets. BlueROV can also be used for many marine tasks, i.e., oyster monitoring [63] and navigation. Our group potentially wants to do marine-aerial navigation like Wang [81] has done for ground-aerial navigation. Detection of the BlueROV becomes necessary for such tasks.

As we can see in Fig. 2.3, the 3D model in the simulation (Fig. 2.3(b)(d)) looks very similar to real BlueROV in the water (Fig. 2.3(a)(c)). Although there might be some visual difference between the images from the real data and the synthetic data, the actual object size in the UAV-based image is relatively small, so we can neglect the visual difference here. The relatively small size of the object in UAV-based images makes it possible for rendering the datasets in *SeaDronesSim*, and directly use the datasets for training a network to detect the real object.

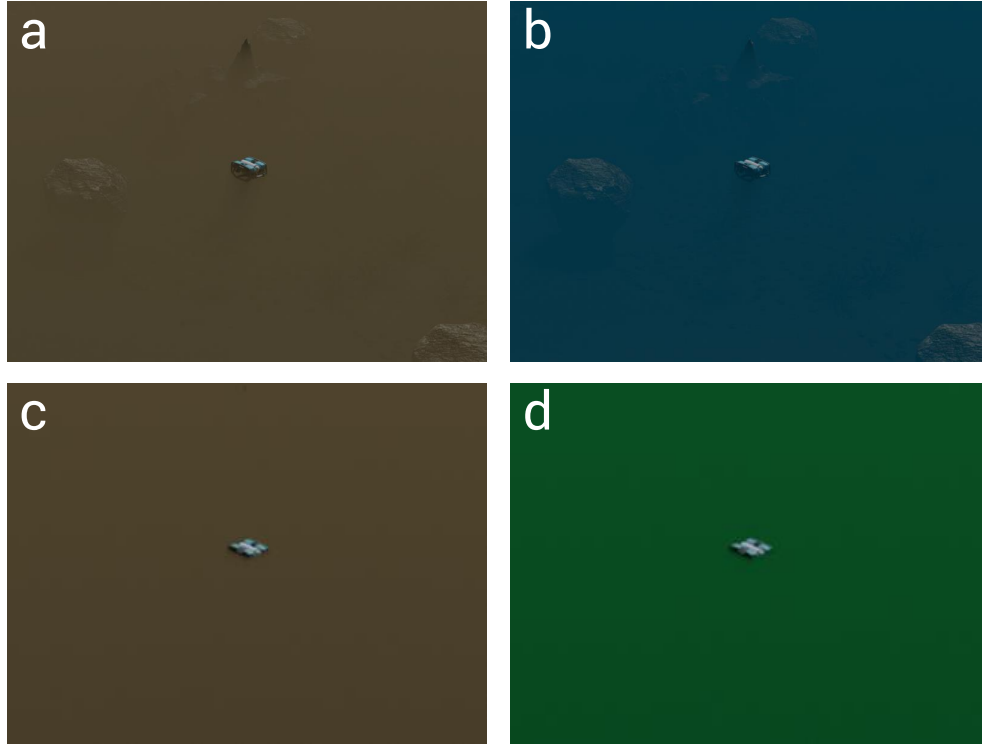


Figure 2.4: Different water colors and turbidity levels.

2.1.2.2 Water Volume

The major distinction between an on-land simulation and a maritime simulation is the water volume. In the BlenderTM [26] environment, the render engine "CYCLE" uses path-tracing of the lights to generate images. In other words, it passes light with a certain number of bounces and receives light to generate images of different brightness, colors, and shades. By customizing particles in the water volume, we get a photo-realistic maritime image for different water colors and turbidity levels. The color of the water volume can also be modified to mimic a specific maritime environment's color scheme. As we can see in Fig. 2.4(a) and Fig. 2.4(b), these are UAV-based maritime images of different water colors in low turbidity. As we increase the turbidity level for the water, the object we added at the bottom (visible in Fig. 2.4(a)) is no longer visible in Fig. 2.4(c). Another image with different water colors in high turbidity is shown in Fig. 2.4(d).

2.1.2.3 Aerial Image

We want to simulate the aerial Images in *SeaDroneSim*. However, we do not care about the UAV's appearance and its physical model. The only thing that we care about is the camera attached to it. So we basically modeled the drone as a point object with a camera attached to it (we will just refer to the point object as the drone). Instead of moving the drone or the object in the simulation, we choose to move the object and fixate the camera on the object. In this case, when the object is closer to the drone, the object will look larger so that we can obtain multiple sizes within one run. However, this will not cover all possible sizes that we may want, so we adjust the altitude of the drone to obtain different images. As we can see in Fig. 2.5, the object and its corresponding ground truth mask in the image become smaller when we increase the altitude when taking the images. The resolution of the image could go up to 30,000 x 30,000 pixels depending on the computational power of the hardware that is running the simulation.

2.1.2.4 Objects of Non-Interest

To simulate some of the scenarios in that some objects not of interest are included, we add a few objects at the bottom of the seafloor and some hills above the sea level as we can see in Fig. 2.1(a).

2.1.3 Experiments and Results

We perform a comprehensive comparison with various training data by utilizing Faster R-CNN [82]. By rendering the training data in *SeaDroneSim*, it can provide any number of images under different sizes, altitudes, angles, and colors. We resized the images to 416 x 416 pixels. For each masked image, the bounding boxes can be determined by the highest and the lowest

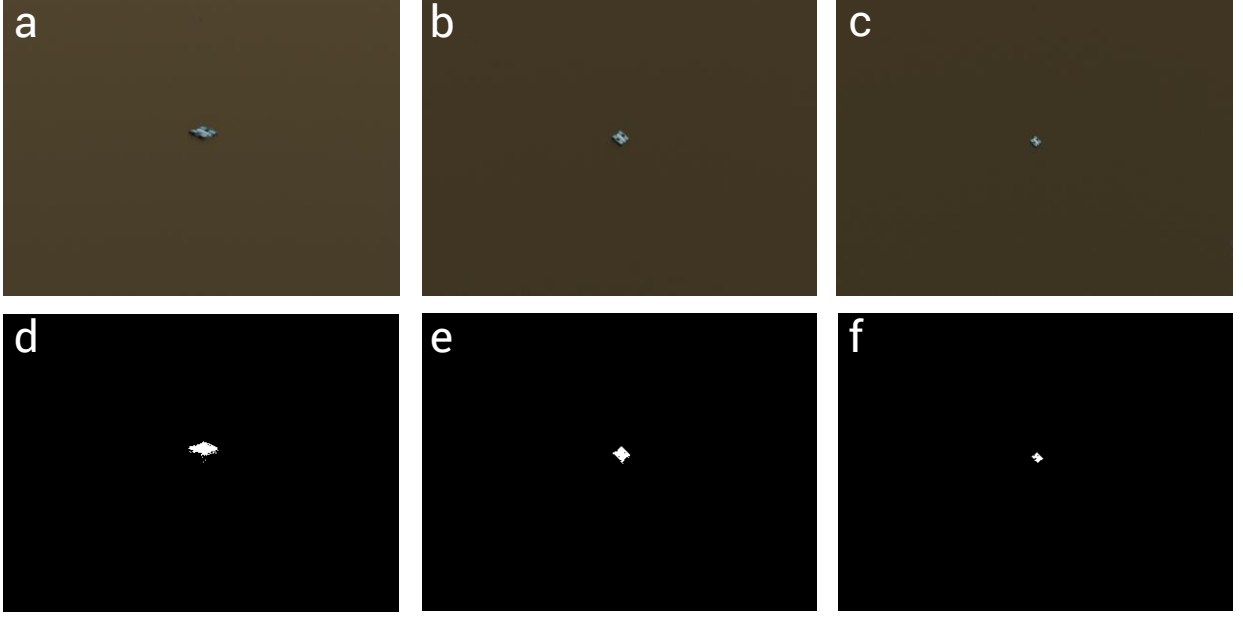


Figure 2.5: Images from different altitudes. (a) Image from an altitude equal to 20 m, (b) image from an altitude equal to 30 m, (c) image from an altitude equal to 40 m, (d)(e)(f) are the corresponding ground truth masks for (a)(b)(c).

x and y coordinates of BlueROV in the image. With these x and y coordinates, we added a BlueROV category and the corresponding bounding boxes. Therefore, each image receives a COCO-dataset format label. Instead of implementing a Faster R-CNN model from scratch, we used Detectron2 [83] which allows us to accelerate the developing process. Detectron2 is a novel system created by Facebook AI Research (FAIR) for computer vision research such as object detection and segmentation.

2.1.3.1 Synthetic Dataset

There are 626 images for each dataset generated with the same parameters/constraints unless specified differently. For all the images from the simulation, we took 80% for training and 20% for validation. Fig. 2.5 shows the altitude images in the training dataset. To avoid the influence of noise, we removed the environment in the simulation. Therefore, it was not affected by other factors except for BlueROV. In the second part, we created a few datasets with different

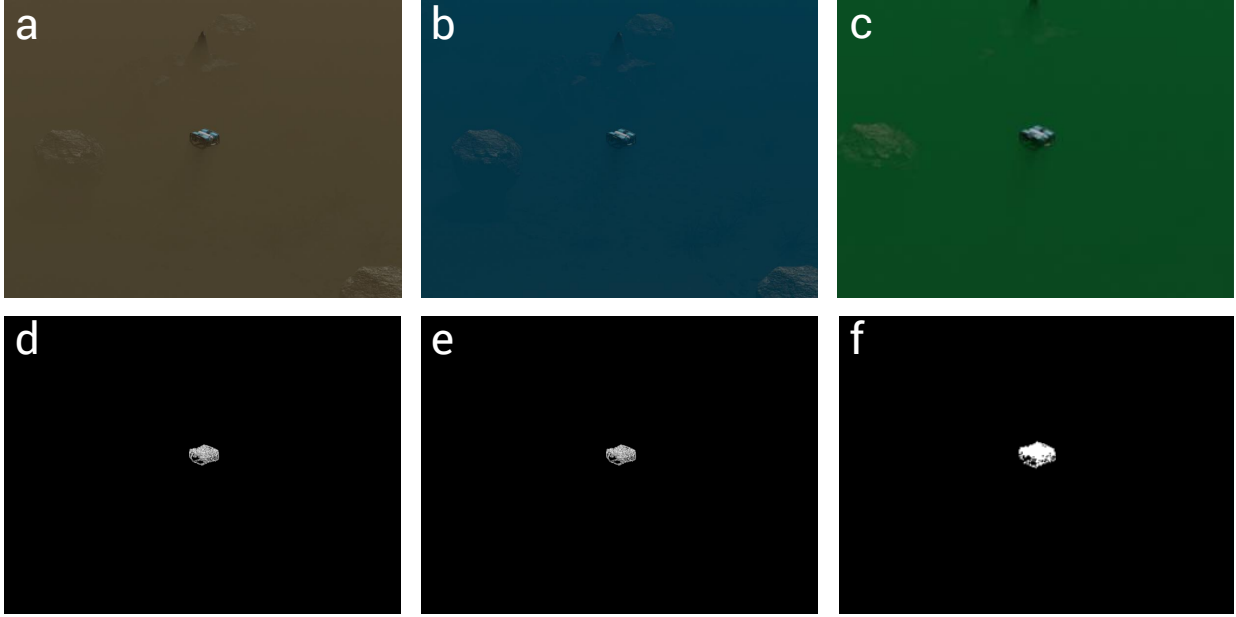


Figure 2.6: Examples of original and ground truth mask images. These examples are based on different background water colors brown (a, d), blue (b, e), and green (c, f), respectively.



Figure 2.7: The first row is the examples of testing images. The second row is the testing results for the above images. These examples are collected from Sandy Point Park in Annapolis, MD.

water colors in the simulation environment. There were 626 images for each color with multiple angles and altitudes in the maritime environment. Fig. 2.6 shows the example images of different water colors in the training dataset. We could mimic the real UAV-based maritime images by modifying the water colors and adjusting the altitude for the image taken.

2.1.3.2 Base model

In this experiment, we used Faster R-CNN with X101-FPN as the backbone to train our dataset. Though X101-FPN takes more time and memory during training, it can produce higher accuracy results comparable to other models in Faster R-CNN. In the model, we set up the learning rate of the model as 0.001. To prevent overfitting of the model to the synthetic data, the maximum number of iterations was 1,500, which can be adjusted according to the performance of validation. Since we only focused on detecting BlueROV in the image, there is only one class in this recognition task.

With dataset and model, it is convenient to train custom models in Detectron2. Registering the coco instance function, we uploaded training, validation, and testing data to the Detectron2 platform. Then we created a class and imported Faster R-CNN as our training model. A function exists that allows us to access the parameter of the model. From this function, we can assign the value to the parameter. The checkpoint and weight were saved in the output file. The checkpoint was provided to us to continue training the data after interrupting the training process.

2.1.3.3 Evaluation

To prevent the convolutional neural network from overfitting to the synthetic dataset for a relatively easy task, we did not provide a large quantity of training data. However, these training data performed at least 95% Average Precision (*AP*) in the validation. Therefore, the next step is to evaluate the testing data. Testing data was collected from Sandy Point Park in Annapolis, MD. Fig. 2.7 shows some of the real UAV-based maritime images. We used DJI pro2 and took a few video clips. We then carefully labeled 2,000 image frames selected from the video as the testing data to produce convincing results.

Table 2.1 displays the output of the testing data with different training data obtained from various altitudes. Samples of images taken from altitudes 20 m, 30 m, and 40 m can be seen in Fig. 2.5(a), Fig. 2.5(b), and Fig. 2.5(c), respectively. The highest mean Average Precision (mAP) is 58.41 at an altitude of 10 meters. Higher altitude results in a smaller object. Therefore, 10 meters altitude images contribute to the higher accuracy to match the testing data compared with others.

Table 2.2 presents the evaluation results for different water color training. As shown in Fig. 2.6, we applied three common water colors: brown, blue, and green. Each color of training data contains different sizes and angles of the BlueROV, as mentioned in Sec. 2.1.2.3, to evaluate the result. Some of the results are displayed in the second row of Fig. 2.7. The brown training data achieved the highest mAP in Table 2.2, followed by blue, whereas green was much lower. This may be because the water color of our field data collected from Sandy Point Park was more brown, which may have made the training more robust to detect with a brown background. Alternatively, brown may simply provide a more distinguishable contrast color to that of the

Table 2.1: The Average Precision (AP) of the training data with various BlueROV image altitudes.

Altitude	AP	AP_{50}	AP_{75}	AP_s	AP_m
10 m	58.41	90.90	72.62	57.82	84.72
20 m	47.46	95.03	31.28	47.27	57.41
30 m	32.70	91.04	1.41	32.63	40.79
40 m	16.81	78.68	0.07	16.78	30.76
50 m	6.87	48.32	0.01	6.82	22.50

Table 2.2: The Average Precision (AP) of the training data with different water colors at an image altitude of 10 m.

Water Color	AP	AP_{50}	AP_{75}	AP_s	AP_m
Brown	58.41	90.90	72.62	57.82	84.72
Blue	54.72	89.12	65.86	54.70	73.88
Green	57.71	89.24	71.33	57.51	78.73

BlueROV, making it easier to detect. The blueuced performance using the green dataset could be an issue with the model’s ability to distinguish between the color of the BlueROV and the color of the maritime environment.

We also perform a few experiments with different data sizes. We suspect that if the dataset is too large for our detection task, the network is likely overfitting to recognize objects in the synthetic domain. We use image capture for brown water color with an altitude equal to 10 meters. Then, we use the same network with the same number of iterations for training and only vary the images used in the training. As we can see, more data does not necessarily produce better results. A dataset of size 2500 is overfitting the network for the detection of the BlueROV. We also use 137 images generated from green water color, namely "G" in Table 2.3. Data from a different domain(water color) could improve the neural network’s robustness and versatility. The result we obtained with a data size of 308 added G has 71 mAP which has a 20.5% improvement over the result without adding different water color data. We have also performed training with only real data which is added in the appendix Table 2.3. Without actually training on any real data, we are able to get very close to the training with real dataset(77.56% mAP). The same experiments for Table 2.1 and Table 2.2 are performed with more training data. These results are shown in the appendix.

Table 2.3: The Average Precision (AP) of the training data with different data size at an image altitude of 10 m of Brown water color

Dataset Size	AP	AP_{50}	AP_{75}	AP_s	AP_m
308	58.90	94.75	72.32	58.73	80.59
626	58.41	90.90	72.62	57.82	84.72
2500	45.38	95.52	25.13	45.06	76.88
308+G	71.00	96.42	87.24	70.94	80.08
626+G	64.87	98.27	77.75	65.04	70.69

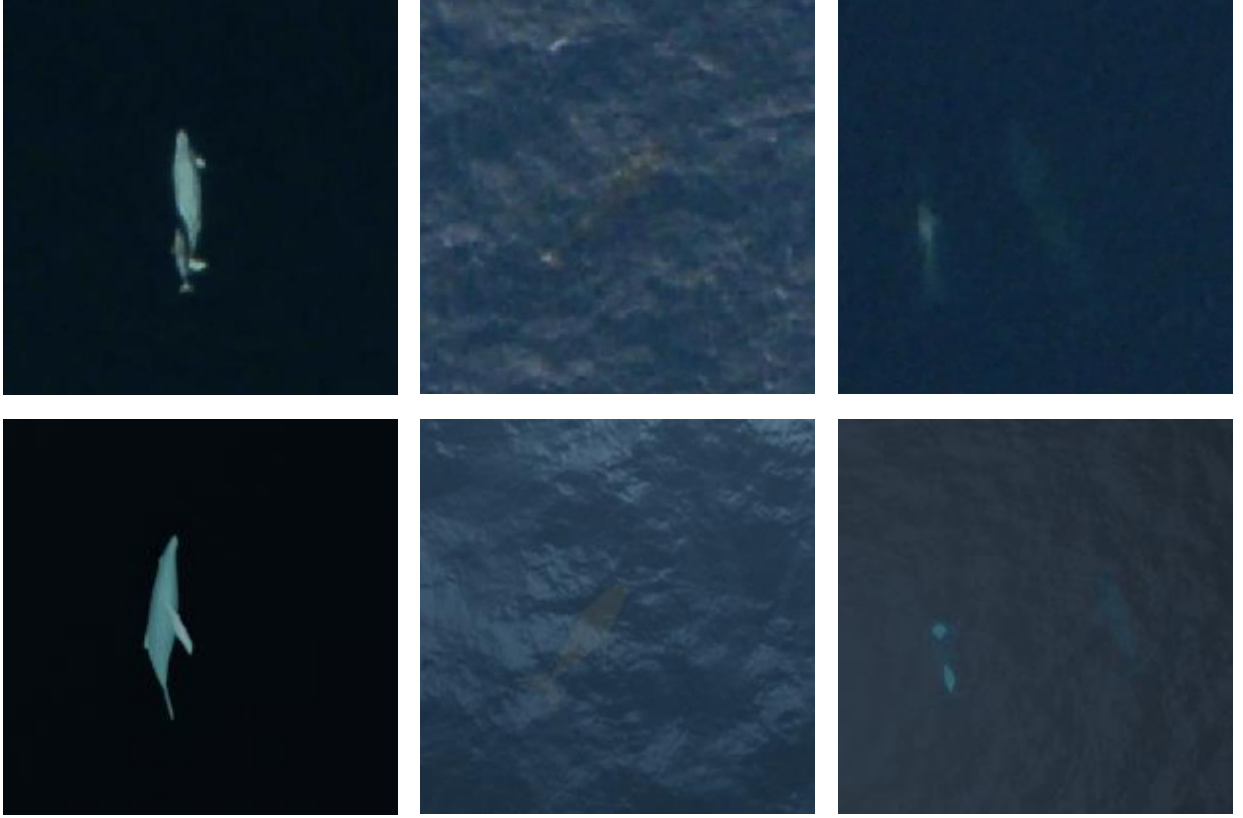


Figure 2.8: First row presents real whale images taken from space while the second row is the simulated whale images from using *SeaDroneSim2*.

2.2 Maritime Object Detection Using Satellite Imagery

2.2.1 Introduction

Satellite images play an increasingly crucial role in diverse tasks such as land-use classification [67], precision agriculture [69, 70], coastal management [71], search and rescue missions [74], and environmental monitoring [4, 84, 85]. These images provide an accessible and comprehensive tool for marine monitoring, offering extensive coverage and detailed Earth surface insights. Remote sensing techniques using these images empower scientists and authorities to enhance marine ecosystem understanding, support conservation endeavors, and improve emergency response capabilities.

In maritime operations the significance of a dependable vision-based system cannot be

overstated. In particular whale detection missions encounter various challenges such as lighting effects on mammals’ visibility, image altitudes affecting appearances, underwater clarity blueuc- tion due to turbidity, varied watercolor grading, and diverse backgrounds. Additionally, object pose and texture variations are crucial for precise detection and tracking.

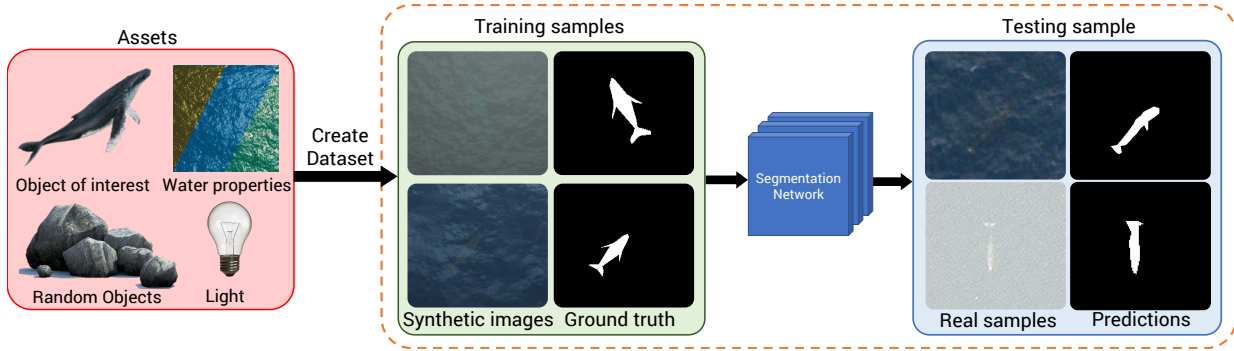


Figure 2.9: An overview of our approach. (a) Assets: Loads the assets such as water properties, objects, materials, etc. into *SeaDroneSim2* to generate Synthetic datasets. Note, the synthetic dataset would include its ground truth mask for the object of interest. (b) We modify the properties within the scene such as noise level, rotation of the object, altitude of the camera and etc. (c) The synthetic dataset generated is then fed into a Neural network to obtain the object detection result. We demonstrated the generation of aerial, satellite, and underwater images for two different objects of interest in our study. Note: Object Detection images are cropped and enlarged for better visualization.

To address these challenges, the integration of robust algorithms becomes imperative, particularly within the realm of deep neural networks. However, the availability of datasets for whale detection in maritime environments, despite some efforts from Cubaynes [86], remains limited in terms of both size and diversity. Acquiring basic remote sensing images for smaller regions can cost over \$100,000 [87]. Manually analyzing and labeling 3357 to 5534 whale images could demand around 1328 to 2016 hours [19]. Gathering aerial images through field operations adds to expenses. Additionally, labeling objects of interest in dynamic and complex maritime environments poses considerable difficulties [50]. Hence, there is a pressing need for alternative methods to generate large-scale datasets rapidly, encompassing a wide variety of objects.

To overcome the scarcity of datasets in maritime environments, we introduce a simulation platform called *SeaDroneSim2* to generate synthetic data and enhance object detection quality. By leveraging our simulation suit *SeaDroneSim2*, synthetic aerial and satellite images can be created to replicate a wide range of objects and environmental conditions. Customizable virtual scenes can be rendered to replicate genuine maritime situations, expediting the creation of varied datasets. Synthetic data generation allows for variations in lighting conditions, altitudes, viewing angles, watercolors, and more. This provides a large variety of training examples that strengthen object detection and tracking systems. *SeaDroneSim2* overcomes the challenges of manual data collection and facilitates the creation of large-scale datasets encompassing a variety of objects, including whales.

Our main contributions are as follows:

- We have made improvements to our novel simulation suit for generating aerial and satellite images for the maritime environment, adding enhanced functionality for noise and water properties.
- We conducted experiments to evaluate the synthetic datasets and compare performances.
- We open-source *SeaDronesSim2* and dataset associated with this work to accelerate further research. To the best of our knowledge, we are among the first to share the segmentation labeling for whale detection.
- We proposed a complete pipeline for autonomously generating aerial and satellite maritime images for objects of interest and detecting them.

2.2.2 Proposed Approach

SeaDroneSim2 is built based on the BlenderTM [26] game engine. After creating the simulated environment, the object of interest is incorporated to generate a synthetic dataset for training a Neural Network in object detection. As depicted in Fig. 2.9), the chosen object and various parameters for the maritime environment (like water texture, lighting, and random objects) are inputted into *SeaDroneSim2*. The open-source tool then produces a training dataset along with corresponding ground truth masks. This generated data facilitates and enhances the development of a detection network for recognizing the object of interest. In the remainder of this section, we will go through some of the details of this image generation and object detection pipeline.

For the maritime object detection application of *SeaDroneSim2*, the Neural Network must be trained to detect a range of shapes and colors for specific objects. It should also account for diverse oceanic variables, such as water texture, watercolor (including different levels of turbidity), and lighting conditions. Training the detection network to be effective for these varying environments requires large training datasets, which, as aforementioned, are often costly in nature [87]. At the time of writing, there are very limited training datasets for maritime object detection, one of which is a dataset from the British Antarctic Survey [86]. Existing datasets, even the one from the British Antarctic Survey, lack ground truth masks for image segmentation of the targeted object. Therefore, images must be labeled by hand to train the Neural Network. Here, *SeaDroneSim2* proves advantageous as it generates accurate ground truth image masks for objects within the simulation.

2.2.2.1 Object of Interest

Generating realistic synthetic environments relies heavily on an accurate 3D model of the object of interest. The selection of the object of interest is based on two key factors: the increasing conservation efforts for marine life, including whales, and the high costs associated with obtaining state-of-the-art datasets. In addition, the use of whales as our primary object of interest is also easily translatable to other marine creatures, such as dolphins and sea lions, due to their presence near the surface of the ocean.

As depicted in Fig. 2.8, the 3D whale model closely resembles an actual whale when observed from a satellite or aerial perspective. Although some differences in detail are evident, the 3D model offers a realistic portrayal of whales in satellite imagery.

Our synthetic data generation tool offers parameterization options specific to the object of interest. This includes the capability to rotate the object along three axes and move the object across three dimensions, as demonstrated in the first two images in the third row in Fig. 2.10. In addition to the physical location of the object of interest, customizable textures can also be applied to the object to simulate different environments, including different species of marine animals and marine objects(in the third row in Fig. 2.10). For each synthetically generated image of the object of interest, *SeaDroneSim2* provides functions to generate image masks. The masks aid in image segmentation and alleviate the requirement for manual image labeling, which can be labor-intensive. Moreover, *SeaDroneSim2* can generate 1000 training images (with resolution 140x140), with their masks, in about 25 minutes. This value varies depending on the resolution of the images being generated.

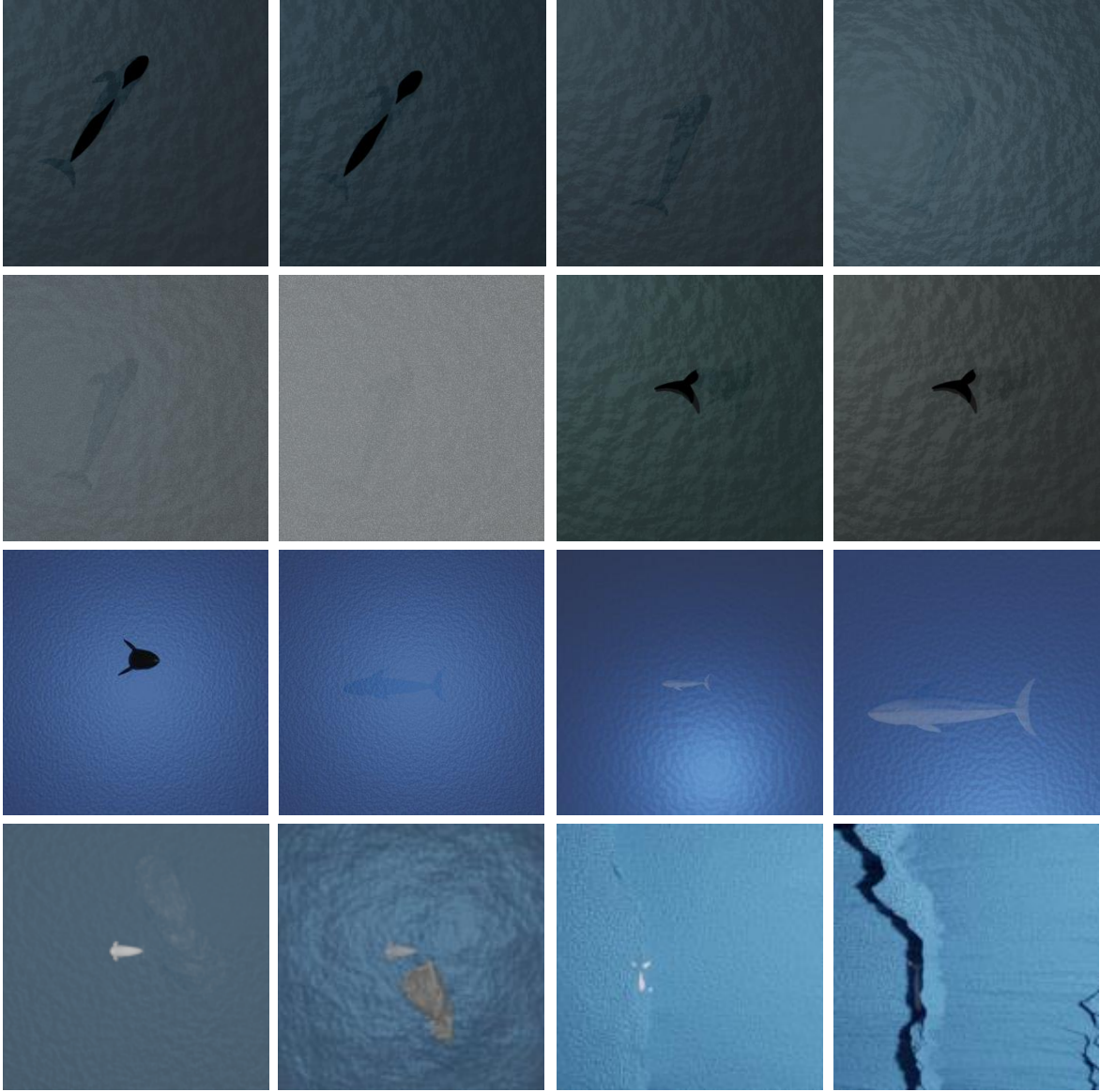


Figure 2.10: These are the synthetic images generated from *SeaDroneSim2*. In the first row, the first two images showcase the increasing turbidity of the water, and the last two images depict varying lighting conditions. In the second row, the first two images display different watercolors, while the last two images exhibit increasing noise from the satellite images. In the third row, the first two images demonstrate varying altitudes, while the last two images illustrate different whale positions, including lodging, spyhopping, and submerging. In the last row, the first two images demonstrate synthetic images with different water waves. while the last two images illustrate different rocks and hills.

2.2.2.2 Water Volume

SeaDroneSim2 uses BlenderTM's render engine CYCLES to create the synthetic oceanic environment. The render engine uses path-tracing of the light object to generate image renders. As

a result, through several customization mechanisms, we can attain realistic oceanic environments for different watercolors, textures, and turbidity levels.

A notable advantage of *SeaDroneSim2* is its automated data generation process, requiring minimal user interaction. By utilizing a local installation of Blender™, the tool generates a Blender™ file containing an oceanic plane equipped with pre-configurable lighting and camera placements.

SeaDroneSim2 provides utilities for both customized and pre-built implementations, enabling users to fine-tune maritime environments. For instance, users can change water color using specific *RGB* value ranges (in the last two images in the second-row Fig. 2.10), generating corresponding images and masks. The default implementation offers 1030 different colors, ranging from blue to green water shades. Users can also adjust water turbidity, affecting object visibility. The first two images in the first-row Fig. 2.10 show increasing turbidity which decreases the whale's visibility.

In addition, water texture is another key parameter in the tool suite. As we can see in Fig. 2.10, there are several parameterization options for water texture, depending on different factors like detail, dimension, scale, metallic, *lacunarity*, and strength. First, the water's detail fine-tunes the roughness of the water, while scale and strength play the most important factors in defining the roughness, with a larger scale resulting in calmer waters. The water's dimension and lacunarity compress or expand the water's patterns to also affect the roughness. Finally, the water's metallic reflects a varying level of refraction of the water itself.

Next, considering the prevalence of white noise in many available satellite oceanic datasets, there is often an element of white noise in datasets. *SeaDroneSim2* also can simulate and render the noise through Blender™. Employing both *White Noise* and *Gaussian Noise*, the tool repli-

cates different noise levels within images, as demonstrated in the first two images of the second row in Fig. 2.10. Similarly, *SeaDroneSim2* also enables varying levels of lighting of the ocean, mimicking varying levels of sunlight, depicted in the last two images in the first row of Fig. 2.10

Lastly, we incorporate the capability to simulate diverse wave characteristics in oceanic settings, as illustrated in the last two images of the last row in Fig. 2.10. These simulated waves encompass variations in height, tilt, sharpness, and textures, offering a range of parameterized functions.

2.2.2.3 Objects of Non-Interest

Moreover, for a comprehensive and immersive environment, enhancements can be made to the surroundings beyond the water. This could involve adding buoys and markers, seabed features, rocks, reefs, or other relevant elements to provide context and realism to the oceanic setting. Such inclusions can contribute to a more realistic simulation and better represent the intricacies of maritime environments. As depicted in the first two images in the last row of Fig. 2.10, an additional rock has been included on the seafloor, showcasing how submerged objects can be incorporated into the simulation. The differing visibility levels and rock height demonstrate how water turbidity influences the clarity of submerged objects.

2.2.2.4 Aerial Image

One of our goals is to simulate aerial images in *SeaDroneSim2*. While other camera angles are easily achievable through BlenderTM's capabilities, most images take advantage of a simple aerial view. However, there does exist variance in camera altitude, which yields larger objects of interest at lower altitudes and smaller objects of interest at higher altitudes. The last two images in the third row Fig. 2.10 illustrate this variance, showcasing how the camera can be parameter-

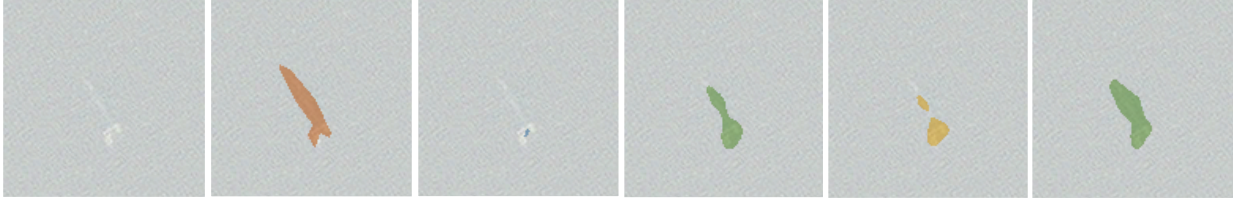


Figure 2.11: From left to right: Sample real input image, ground truth, segmentation result using Unet without synthetic augmented real data, segmentation result using Unet with synthetic augmented real data, segmentation result using FPN without synthetic augmented real data, segmentation result using FPN with synthetic augmented real data. All networks here are trained with only 10% of real data.

ized based on the object’s specifications, while typically being fixated on the object of interest by default. While the majority of training images maintain a resolution of 140×140 pixels, the image resolution can extend to $30,000 \times 30,000$ pixels, depending on the computational capabilities of the underlying hardware.

In summary, by combining realistic rendering of water properties, such as transparency and wave dynamics, with the integration of diverse underwater elements, *SeaDroneSim2* can serve as a valuable tool for exploring maritime situations, enhancing and evaluating detection algorithms, contributing to cetacean conservation, and various other applications.

2.2.3 Experiments And Results

First, we describe the dataset we used in these experiments. Then we compare the results obtained by two different segmentation networks of detecting the whales utilizing datasets generated from *SeaDroneSim2*.

2.2.3.1 Synthetic Dataset

During our rendering process, we produced a total of 2000 synthetic images to construct a diverse and representative dataset. To mimic real-world conditions, we introduced variations

in lighting, altitudes, orientation, water color, Gaussian noise levels, water turbidity, and wave patterns. These modifications were implemented to approximate the realistic characteristics commonly observed in satellite images, thereby creating a more photo-realistic environment for the network to learn from.

Moreover, for training the network to segment whales in diverse body orientations of the whale, we incorporated synthetic whale instances with varying body orientations of the whale in the simulations. Using this approach, we enforced the network to segment whales more accurately, regardless of their body orientation or appearance.

2.2.3.2 Real Dataset

To obtain whale images from the satellite view, we accessed a collection of 633 images through the UK Polar Data Centre (PDC) [86]. These images played a crucial role in evaluating the effectiveness of our approach. However, during our dataset review, we noticed that the majority of these images only captured a small portion of the whale above the sea surface, and in some cases, only a splash of water from the whales.

Due to the unique characteristics of these satellite images, we conducted a meticulous examination to identify suitable ones that could be included in our test dataset. After careful selection, we were able to compile a set of 508 images that met our criteria and could be effectively utilized for testing and evaluating our approach.

Although this subset of real satellite images may have limitations in terms of whale visibility and coverage, it provides us with valuable test data to assess the performance and robustness of our method in segmenting and identifying whales under challenging scenarios.

2.2.3.3 Evaluation Metrics

To evaluate with the real dataset, we use the Intersection over Union (IoU) which is a common evaluation metric used to assess the performance of image segmentation algorithms. It measures the similarity between the predicted segmentation and the ground truth segmentation. The term *Intersection* refers to the region where the predicted segmentation and the ground truth segmentation overlap. On the other hand, *Union* encompasses the entire area covered by both segments, whether overlapping or not. To assess the model’s performance, we define the Success and Detection Rate (DR) for each cluster based on the IoU as

$$\text{Success} := \text{IoU} \geq \tau; \text{DR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.1)$$

where TP represents the true positive count, which corresponds to the number of correctly segmented instances by the model, and FN represents the false negative count, signifying the number of instances that were present in the ground truth but were not segmented by the model. We assign DR with two different thresholds τ of 0.5 and 0.6, which are denoted as DR_{50} and DR_{60} in Table 2.4. Finally, we conduct a comprehensive analysis by computing a series of results for varying percentages of real data size, allowing us to make a thorough comparison between the method with and without the incorporation of *SeaDroneSim2*.

2.2.3.4 Experimental Results

During our testing phase, we evaluated two types of networks: Unet [88] and Feature Pyramid Networks (FPN) [89]. For both networks, we used a learning rate of 0.001 with decay. The optimization process utilized the Adam optimizer in conjunction with the Jaccard loss [90] as our loss function.

Table 2.4: Comparison of Semantic Segmentation Results with the Two Different Networks

Method	DR_{50}	DR_{60}	Real Data size
<i>Unet</i>	0.615	0.488	10%
<i>Unet + SeaDroneSim2</i>	0.710	0.512	10%
<i>Unet</i>	0.849	0.687	50%
<i>Unet + SeaDroneSim2</i>	0.833	0.710	50%
<i>FPN</i>	0.679	0.476	10%
<i>FPN + SeaDroneSim2</i>	0.746	0.551	10%
<i>FPN</i>	0.861	0.714	50%
<i>FPN + SeaDroneSim2</i>	0.861	0.683	50%

Before training, we augmented the dataset using various techniques, including a rotation range of 90 degrees, width shift range of 0.3, height shift range of 0.3, shear range of 0.5, zoom range of 0.3, horizontal flip, and vertical flip. These data augmentation techniques are applied to the images to artificially increase the dataset’s diversity and expose the model to various transformations that may occur in real-world scenarios. By incorporating these augmentations, we aim to enhance the model’s ability to generalize and improve its performance on unseen data during training.

Throughout the training process, we used a batch size of 32 and conducted training for 100 epochs. To ensure reliable and consistent results, each training was performed at least 10 times (tests), and the best result was selected for further analysis and comparison. By conducting these extensive tests on both Unet and FPN networks, we aimed to determine the usability of our rendebblue synthetic image and identify the most suitable model for *SeaDroneSim2* application.

Given the significant time and financial expenses involved in collecting real datasets, our strategy focuses on minimizing the reliance on real data for training. To achieve this goal, we performed tests using different proportions of the real dataset, specifically 10% and 50% of the

available real data. The remaining 50% of the real dataset (our held real data set) is reserved for testing purposes.

Both the Unet and FPN methods were tested, and we obtained DR_{50} and DR_{60} scores using only 10% of the real dataset, which serves as our baseline for the task. For the Unet method, the DR_{50} and DR_{60} scores were 0.615 and 0.488, respectively. As for the FPN method, the DR_{50} and DR_{60} scores were 0.679 and 0.476, respectively. The results are tabulated in Table 2.4 After training the model solely on synthetic datasets and conducting the testing, we observed that the results were lower than the baseline. We think both networks have learned to recognize whales in the synthetic domain but the sim-to-real domain transfer is lacking in this case.

We achieve better results when including the synthetic dataset rendered by *SeaDroneSim2*. The DR_{50} and DR_{60} results from ‘Unet+*SeaDroneSim*’ (Unet with synthetic augmented real data) are 0.71 and 0.512 against the human-labeled ground truth which is 15.4% and 4.91% better than just using a 10% real dataset for training. The DR_{50} and DR_{60} results from ‘FPN+*SeaDroneSim*’ (FPN with synthetic augmented real data) are 0.746 and 0.551 against the human-labeled ground truth which is 9.86% and 15.7% better than just using a 10% real dataset for training. Moreover, the DR_{60} results from ‘Unet + *SeaDroneSim* (Unet with synthetic augmented real data) are 0.71, which is 3.34% better than just using a 50% real dataset for training.

As depicted in Fig. 2.11, the underwater submerged portion of the body of the whale would be overlooked if the network were not trained with synthetic augmented real data. Consequently, this situation results in numerous false negatives. Our synthetic data includes numerous samples where either a portion of the whale’s body or the entire body is submerged in water. Thus, training the network using real data augmented with our synthetic data leads to a substantial reduction in false negatives. This improvement leads to more accurate predictions and a higher success rate

even in challenging scenarios.

Augment the dataset with synthetic does not always guarantee improvement, as evidenced by the slight drop in the DR_{60} result from both "*FPN+SeaDroneSim*" and DR_{50} result from "*Unet + SeaDroneSim*" when using 50% of the real dataset in Table. 2.4. However, this drop could be attributed to the fact that our dataset is relatively small, which leads to limited robustness in handling diverse real-world scenarios for whale detection.

2.3 Marine Object Detection

2.3.1 Introductions

Oyster reefs are filter feeders and they provide crucial benefits for the benthic marine ecosystem(s) such as increasing the richness for a variety of species and providing living habitat, food, and protection for numerous marine species. However, the standing stocks for the oysters near the Chesapeake Bay [91] and North Sea [92] have dropped significantly due to over-fishing, global warming and the effect of diseases across the 19th century. To tackle this devastating ecological problem, massive efforts are being carried out to restore oyster habitats across the United States [93–96] and Europe [92].

One of the core challenges to advancing, improving, and adapting the restoration process is monitoring the progress of oyster restorations effectively. Beck et al. [97] proposed to standardize monitoring metrics, units, and performance criteria for the evaluation of the oyster reefs. A set of environmental variables including water salinity, temperature, and dissolved oxygen are being monitored to determine the well-being of oyster habitats. For the oyster reefs, universal parameters such as "reef areal dimensions, reef height, oyster density, and oyster size-frequency distribution" are monitored and reported in the literature [92, 94–97].

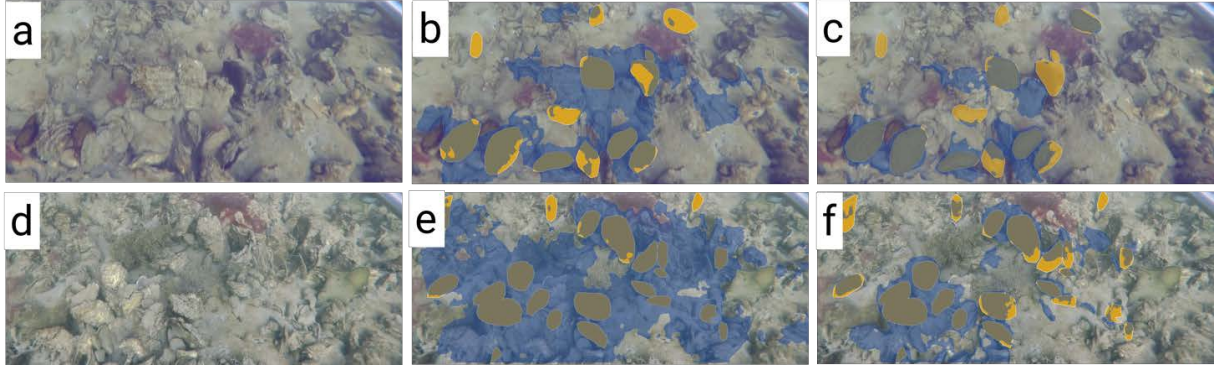


Figure 2.12: Each row left to right: Input image, the output of the network when trained using only real data, the output of the network (which we call *OysterNet*) when trained using real data augmented with our synthetic data. Yellow represents the oyster segmentation ground truth and the blue is the predicted segmentation result. Notice how the number of false positives and false negatives drops significantly when the training data is augmented with our synthetic data. *All the images in this paper are best seen in color on a computer screen at 200% zoom.*

However, these metrics rely on recognizing and counting oysters, which is currently largely done by expert manual labor. Such a process is slow, time-consuming, and has poor scalability.

Using such a manual approach, the oyster reefs can only be monitored within a restricted area with few samples, e.g., 100 oysters per sample site [95]. Furthermore, the material for the oyster’s surface is similar to the seabed sediment (See Fig. 2.12a, Fig. 2.12d.). And it is significantly different from the washed oysters in Fig. 2.14a, which makes it very challenging to train new people or algorithms to perform oyster counting.

To streamline the process of oyster mapping, the goal is to utilize the advancements in robotics and artificial intelligence that can enable us to gather images from underwater Remotely Operated Vehicles (ROVs) and then automate the oyster detection and density calculation. The central part of this process is to build an oyster detection system. In this work, we present a mathematical model to generate oyster models and further use Generative Adversarial Networks to enable sim-to-real transfer. To the best of our knowledge, this is the first attempt to geometrically model oysters. The contributions of this paper are as follows:

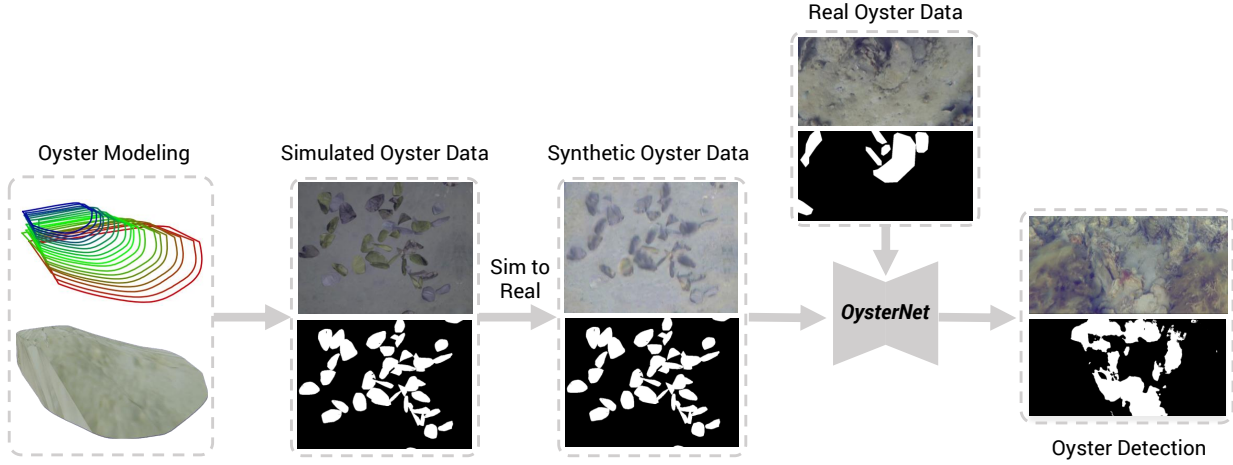


Figure 2.13: An overview of our approach: The proposed geometric model is used to generate synthetic images which are further fed into a Generative Adversarial Network to enable sim-to-real transfer (domain adaptation) by generating photorealistic oyster images. We then combine the synthetic data with real data to train a *OysterNet* for oyster detection.

- We propose a novel mathematical model for the 3D shape of oysters.
- We simplify the geometric model of an oyster for the projection on the image plane which is used to generate photorealistic synthetic oyster images. These images are used to train a deep segmentation network *OysterNet* for oysters that achieves the new state-of-the-art.
- We open-source our oyster generation model and dataset associated with this work to accelerate further research.

2.3.2 Synthetic Oyster Generation

In this work, we propose a novel method for modeling oysters which are then used to generate a large dataset of oyster reef images. In this section, we will talk about our proposed method (which is summarised in Fig. 2.13). First, we will describe the process of modeling oysters. Next, we will talk about how this model can be used to generate photorealistic images of oyster reefs using domain adaptation. Last but not least, we train a semantic segmentation model



Figure 2.14: Steps in the geometric modeling of an oyster: (a) Sample image of a real oyster shell, (b) 3D scan of a real oyster, (c) Splines fit to the oyster's bottom layer to model it (each color represents a single spline), (d) Simplified model of one stratified layer of the oyster (single layer of spline curve $S(t)$), (e) all layers of $S^\alpha(t)$, (f) generated 3D model of oyster, (g) final generated 3D model of oyster with added real oyster texture.

using the *OysterNet* for detecting the oysters in the real world. We present each sub-part in the following sub-sections.

2.3.2.1 Geometric Modelling Of Oysters

In order to model the geometry of an oyster, we first 3D scanned ten washed oysters (sample image example shown in 2.14a and scanned model shown in 2.14b) which were used to build a mathematical model. Furthermore, the parameters in the proposed mathematical model can be adjusted to generate models of various oysters. We will describe the mathematical model next.

It is important to note that the geometrical shape of an oyster is extremely complex, with each one having a different shape and thickness. Oysters grow following a general oyster shape but expand somewhat randomly along their margins. In the first step, we will model our oysters in 3D. Each oyster can be approximated as a series of mathematical functions in a stratified manner (akin to the way a 3D printer prints). We start with each 'horizontal' layer (slice of the cross-section of the oyster) by looking at the top view of our scanner oysters. We call this the perimeter of the oyster. We noticed that the perimeter can be easily modeled using two cubic B-splines. Let the $n + 1$ control points for the splines be $\{c_0, \dots, c_n\}$ and $m + 1$ knot vectors be $\{t_0, \dots, t_m\}$, then the spline curve $S(t)$ of degree k is given by

$$S(t) = \sum_{i=0}^n c_i B_{i,k}(t) = 1, \quad (2.2)$$

where, $B_{i,k}(t)$ denotes the basic function of degree k and is computed recursively as

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_{i+1} \geq t \geq t_i \\ 0, & \text{otherwise,} \end{cases} \quad (2.3)$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t). \quad (2.4)$$

In our case here, $m = n + k + 1$ and we selected $k = 3$ for a cubic spline. Particularly, we utilize two cubic B-splines: one for the top half of the shell, and one for the bottom of the shell (See Figs. 2.14c and 2.14d). We will call this perimeter model (Eqs. 2.2, 2.3 and 2.4) as the 2D model since it models only a single layer of the oyster.

Now, we want to extend the 2D model to 3D in the stratified manner we described before. However, having a high resolution in depth is computationally prohibitive due to all the nooks and crannies (high-frequency edges) on the oyster shell (See Fig. 2.14b). In order to make this computation tractable, we simplify the shape of the oyster by assuming that it is smooth since the variation on the shell is much smaller than the distance to the oyster. It is important to highlight that the visual changes these high-frequency edges created are approximated by the visual textures we place on our generated oysters. The 3D model of the oyster (Fig. 2.14e) follows the same perimeter model from before but also adds changes to depth. Let $c_0^\alpha, \dots, c_n^\alpha$ denote the control points for α -th layer of the 3D oyster. And the knot vectors for the α -th layer

be $\{t_0^\alpha, \dots, t_m^\alpha\}$. We define the c_n^α, t_m^α as follows

$$c_n^\alpha = c_n + X_1 \quad (2.5)$$

$$t_m^\alpha = t_m + X_2 \quad (2.6)$$

where X_1 and X_2 are the Gaussian Noise used to model the high frequency edges of the oyster's perimeter. Formally, $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$.

We can then substitute c_n, t_m with c_n^α, t_m^α in Eqs. 2.2 to 2.4 to get the spline curve $S^\alpha(t)$ for every single layer.

For each spline curve $S^\alpha(t)$, we can use $[x'_\alpha, y'_\alpha]^T$ to represent spline curve points as follows:

$$[x_\alpha, y_\alpha]^T = [x'_\alpha, y'_\alpha]^T r^\alpha \quad (2.7)$$

where r is the in-growth rate as a fixed number.

Finally, the growth rate of the oyster is defined as follows:

$$z_\alpha = \alpha d, \quad (2.8)$$

where d is the depth at a fixed number here.

Now we can use all the α layers with $[x_\alpha, y_\alpha, z_\alpha]^T$ points to generate 3D model (Fig. 2.14f) by using pyvista [98] data visualizer. By varying the parameters $\Theta = \{\sigma_1, \sigma_2, \mu_1, \mu_2, \alpha\}$ we obtain different oyster shapes (Fig. 2.18). Further, we use image textures from real oysters collected in the Chesapeake Bay to be warped on the generated 3D model (See Fig. 2.14g).

In the next section, we will talk about how the actual images are rendered from the 3D models we just constructed.

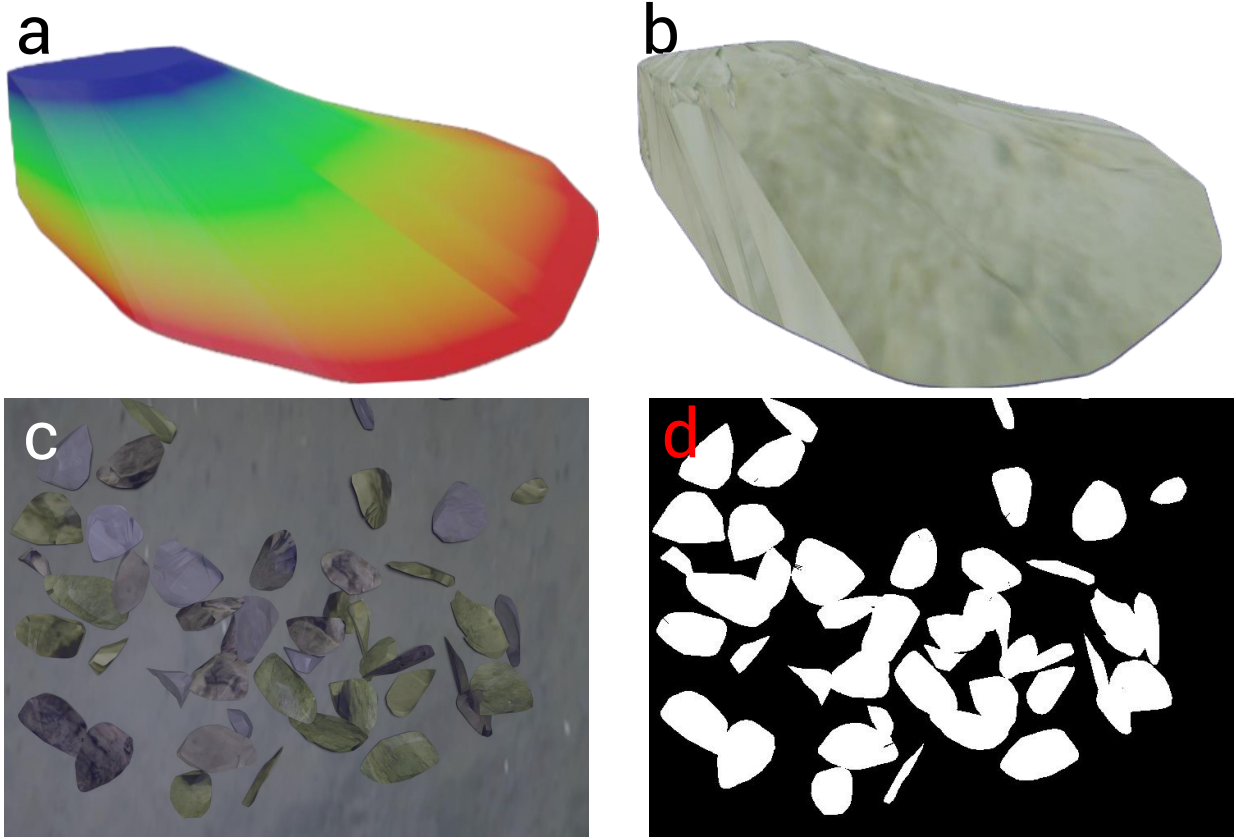


Figure 2.15: (a) One of the synthetic models generated from Sec. 2.3.2-A), (b) Synthetic model with real oyster texture added, (c) image of an oyster farm with 50 synthetic oysters generated in BlenderTM, (d) masks for oysters in (c).

2.3.2.2 Synthetic Image Generation

- **Simulation Image Rendering** A 3D model is not sufficient for creating images for oyster segmentation. We utilized the BlenderTM [26] game engine to simulate the oysters on a seabed. We rendered 13K synthetic oysters with different 3D models (we will just call it the synthetic model for simplicity) by varying parameters Θ as described in Sec. 2.3.2. Then we used these synthetic 3D models for synthetic data generation. First, we employed 14 real oyster texture images and applied them randomly to all the synthetic models (Fig. 2.15a). A sample of the synthetic model with applied texture image can be seen in Fig. 2.15b. To generate a simulated oyster reef image, we placed the oysters

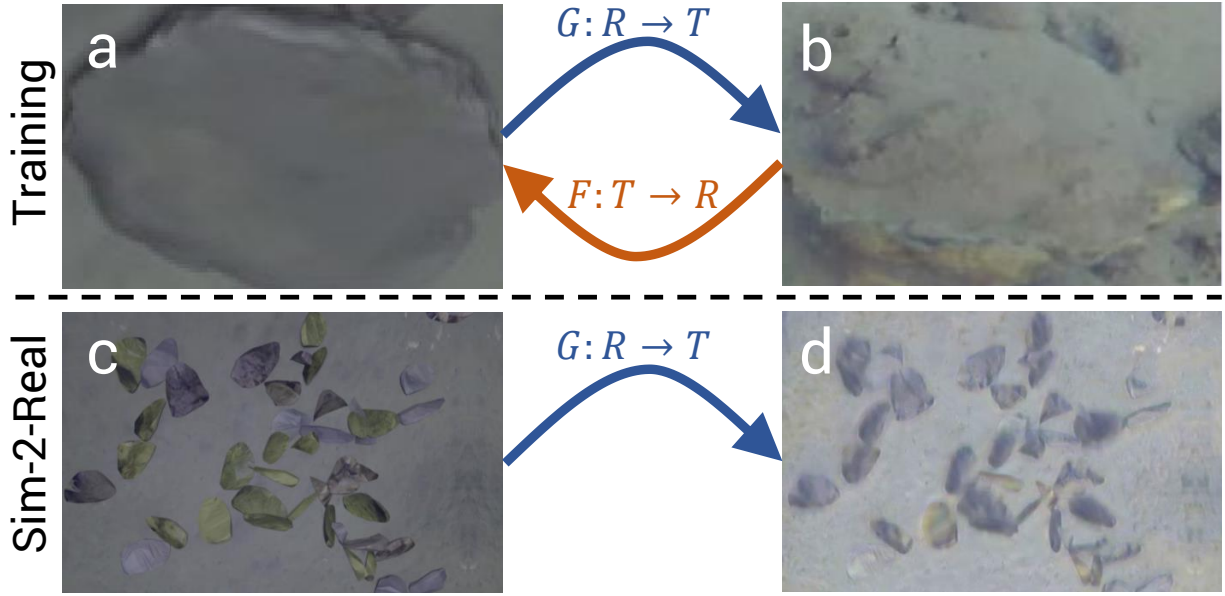


Figure 2.16: Domain Adaptation to perform sim-to-real transfer. (a) A single sample of the simulated oyster, (b) A single sample of the real oyster, (c) Simulated oyster farm, (d) Synthetic oyster farm domain adapted to real world for photorealism.

with random poses onto a flat surface with the seabed textures as shown in Fig 2.15c.

Images of the oyster reef are then rendered in Blender™ along with segmentation masks (Fig. 2.15d).

However, Fig. 2.15c is not photorealistic and does not look like a real underwater image which will lead to a poor detection performance when deployed. We describe how we use a Generative Adversarial Network (GAN) to perform sim-to-real domain adaptation such that our images are photorealistic such that they can generalize to the real world after being trained in simulation.

- Domain Adaptation To render a realistic oyster image, we employ contrastive unpaired translation (CUT) [99] for unpaired image-to-image translation by learning functions to map from the Blender [26] domain R to the target domain T . There are overall three

different components for the loss function: GAN loss, PatchNCE loss, and externalNCE loss.

GAN Loss: Generators G and F are used to transfer domains: $G : R \rightarrow T$ and $F : T \rightarrow R$. Discriminator, D_R , is designed to distinguish between Blender rendebblue images in R , and translated fake images $F(T)$. Discriminator, D_T , aims to separate target images in T and translated fake images $G(R)$. To improve image-to-image translation in CUT, cycle consistency is maintained by ensuring the reconstructed images $F(G(R)) \approx R$ in addition to the adversarial loss. To calculate adversarial loss, G tries to generate $G(R)$, which is so similar to T that can fool the discriminator D_T . The loss for G and D_T is called adversarial loss and is defined as follows:

$$L_{GAN}(G, D_T, R, T) = E_{t \sim T}[\log D_T(t)] + E_{r \sim R}[\log(1 - D_T(G(r)))] \quad (2.9)$$

where t and r denotes the instances from domain T and R respectively. E is the loss function here as the L1-norm in our approach. In addition, the overall loss has two more components. One is PatchNCE loss, the other one is called externalNCE loss. We first break the function G into two parts, one encoder, and one decoder: $\hat{T} = G(R) = G_{dec}(G_{enc}(R))$. Since the encoder, G_{enc} , is used for image translation, its feature stack each layer and spatial location of the feature stack from the encoder G_{enc} has been used to represent a patch of the input images. L layers from the feature stack are being selected to pass through a two-layer MLP network H_l to produce a stack of features $\{z_l\}_L = \{H_l(G_{enc}^l(r))\}_L$. G_{enc}^l stands for the output from the l -th layer where $l \in \{1, 2, \dots, L\}$. Let's denote $s \in \{1, 2, \dots, S_l\}$, where S_l is the the spatial location for the l -th layer. These features are named as $z_l^s \in$

R^{C_l} and other features from different layer as $z_l^{S_s} \in R^{(S_t-1) \times C_l}$. C_l here is the number of channels for each layer l . We could also encode the translated image $\hat{t}$ into $\{\hat{z}_l\}_L = \{H_l(G_{enc}^l(G(x)))\}_L$. We define the PatchNCE loss as follows:

$$L_{PatchNCE}(G, H, R) = E_{r \sim R} \sum_{l=1}^L \sum_{s=1}^{S_l} \delta(\hat{z}_l^s, z_l^s, z_l^{S_s}) \quad (2.10)$$

where δ here is the cross-entropy loss. We can also use the image patches from the rest of the dataset while training. A random negative image from the dataset $\hat{r}$ is encoded into $\{\hat{z}_l\}$ and is used to define externalNCE loss:

$$L_{externalNCE}(G, H, R) = E_{r \sim R} \sum_{l=1}^L \sum_{s=1}^{S_l} \delta(\hat{z}_l^s, z_l^s, \hat{z}_l) \quad (2.11)$$

The overall loss here being used to train the network is the combination of all three losses.

$$L = L_{GAN} + L_{PatchNCE} + L_{externalNCE} \quad (2.12)$$

We refer the readers to [99] for a detailed description of the Loss function and network. In our proposed system, the target domain is the realistic environment T_{real} .

- **Details of Training the CUT** We want the CUT network to be able to capture the synthetic oyster to real oyster translation. So, we extract 8959 images of single oysters from the synthetic data that we generated as R and we also extract 957 samples of single oysters from the real underwater images as T_{real} . In the training phase, we learn two mapping functions: $G : R \rightarrow T$ and $F : T \rightarrow R$. Once the CUT is learned, we perform inference on our synthetic images to make them look photorealistic (See Fig. 2.16) by performing domain transfer which is further used to train our oyster detection network *OysterNet*. In

this work, we train CUT for 153 epochs to obtain our generator G . In our case, we trained the CUT network for 153 epochs to get our final Generator G .

In the Testing phase, we applied Generator G onto the simulation data and get the synthetic data. Notably, individual oysters are used to train the generators G and F as shown in Fig. 2.16a and Fig. 2.16b. Instead of applying the trained network to the individual oyster, We applied the trained generator G to the larger images (Fig. 2.16c) to obtain synthetic oyster data (Fig. 2.16d) with 30 to 60 oysters per image.

2.3.3 Experiments And Results

First, we describe the datasets used in this section. To validate our generated oyster model, we then compare the results obtained by two different segmentation networks and two different datasets. Lastly, we experiment with how different values of the oyster model parameters affect the segmentation results.

2.3.3.1 Description of Datasets

Rendeblue/Synthetic Dataset: contains images obtained by randomly rendering generated 3D oyster models on the seabed using BlenderTM. Rather than just laying the oysters on the seabed, we simulate the randomization pose of the oysters in various realistic positions. This randomness in oyster pose makes the neural network more robust for recognizing the oyster in different poses. After the images are generated from BlenderTM, we used CUT, as described in Sec. 2.3.2-B, to create photo-realistic images. The rendeblue dataset contains 4800 images.

Real Dataset: contains images from Harris Creek taken by Chesapeake Bay Foundation by driving a BlueROV as shown in Fig. 2.17. We labeled the images by hand. We have 29925

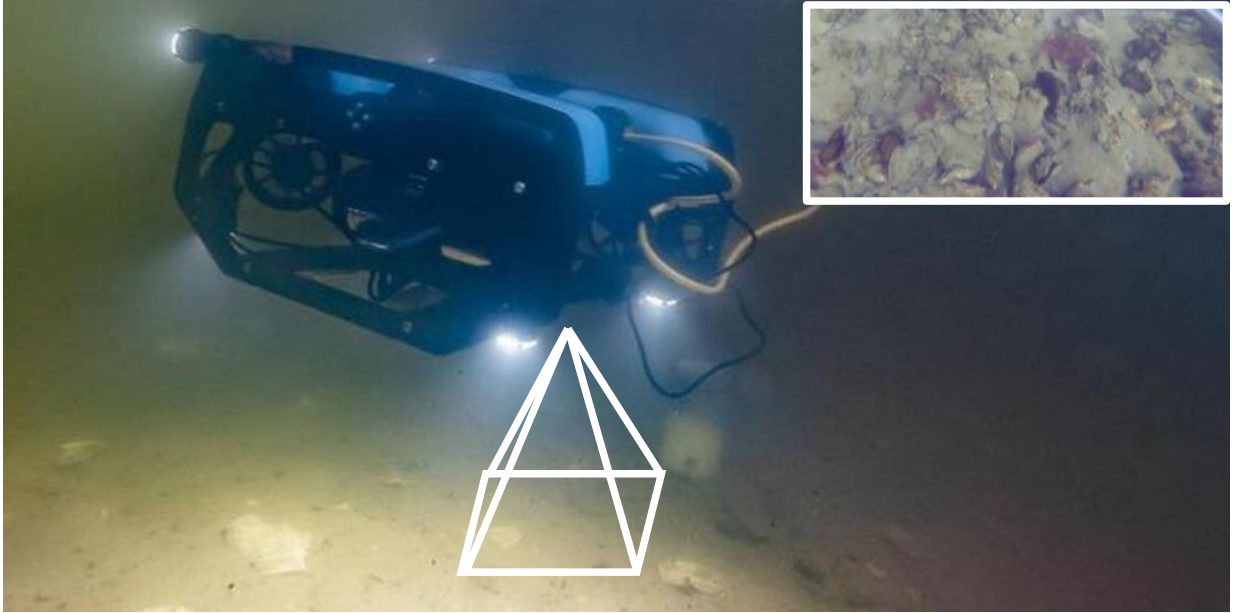


Figure 2.17: BlueROV collecting data for our real dataset. The inset shows a sample image captubblue which is a part of our real dataset.

images of size 256×256 . There are over 3900 oysters labeled in the dataset. Unlike some other datasets for underwater object detection, oysters are really hard to recognize, even for humans. So, we worked with experts at the Chesapeake Bay Foundation to obtain this dataset.

2.3.3.2 Experimental Results

OysterNet adapts UNet [88] as its backbone. We trained *OysterNet* with a learning rate of 0.001 with decay. We use Adam optimizer coupled to the Jaccard loss [90] loss function. We used a batch size is 32 and trained the network for 100 epochs.

We want to verify our hypothesis that the synthetic oyster data we generated can help improve the oyster semantic segmentation.

[14] obtained the best results for oyster detection using the FPN [89] network. This is the model we use which we denote as the DCO (Detecting and Counting Oysters) method. In our observation, UNet [88] architecture performed better using our approach which refers to it as

OysterNet (“Ours”) in Table 2.5.

We want to use the minimal amount of real data for training as they are hard to obtain, expensive, and labor intensive. To this end, we use only 25% of the real dataset we used as training which we will denote as O_{real} . We test our method on the remaining 75% of the real dataset that we call O in the Test Data. We denote all the images in our generated synthetic dataset as O_{syn} which will be used for training with/without real data. Formally, $O_{syn_and_real}$ is the combination of O_{real} and O_{syn} .

In this experiment, we perform training on O_{real} (our real dataset) and test it on the O (our held-out real dataset) with both *OysterNet* and DCO methods. The Intersection over Union (IoU) scores are 18.16% and 18.88% respectively which serves as the baseline for oyster segmentation results for our dataset. Both the methods perform similarly in this case. Then we want to observe the results when trained only on the generated synthetic dataset (O_{syn}). We train on O_{syn} and test on O with both methods. The IoU score is 7.45% and 6.47% respectively which is lower than our baseline. The network has learned to recognize oysters in the synthetic domain but the sim-to-real domain transfer is still not as desirable. To this end, we add a small amount of real data to the synthetic data and use it for training ($O_{syn_and_real}$). We achieve a state-of-the-art IoU Score of **24.54%** against the expert human labeled ground truth which is 35.1% better than just using real dataset for training and 12.7% better than DCO when trained on synthetic augmented real data.

The results are tabulated in Table 2.5. It shows that with the added synthetic dataset, we achieve a significant $\sim 35.1\%$ accuracy improvement over just using the real dataset for training. As we can see from Figs. 2.12b and 2.12e, the network predicts mostly sediments as oysters when trained using only real data. This leads to a lot of false positives and false negatives. How-

Table 2.5: Comparison of Semantic Segmentation Results with the State-of-the-art.

Method	Train Data	Test Data	IoU Score(%)
DCO [14]	O_{syn}	O	6.47
DCO [14]	O_{real}	O	18.88
DCO [14]	$O_{syn.and.real}$	O	21.76
<i>OysterNet</i> (Ours)	O_{syn}	O	7.45
<i>OysterNet</i> (Ours)	O_{real}	O	18.16
<i>OysterNet</i> (Ours)	$O_{syn.and.real}$	O	24.54

Table 2.6: Ablation Studies Of The Proposed Oyster Geometric Model (Also see Fig. 2.18).

Scenario a						Scenario b					
μ_1	50	100	150	200	250	μ_2	50	100	150	200	250
IoU(%)	21.70	23.10	24.54	23.21	17.67	IoU(%)	24.19	21.23	24.54	22.34	19.57
Scenario c						Scenario d					
σ_1	50	100	150	200	250	σ_2	15	30	60	120	240
IoU(%)	19.37	22.07	24.54	23.64	20.67	IoU(%)	24.54	21.53	23.64	22.42	22.95
Scenario e						Scenario f					
α	5-10	10-15	15-20	20-25	25-30	Scale(%)	10-15	15-20	20-25	25-30	30-35
IoU(%)	22.76	22.88	24.54	22.63	22.66	IoU(%)	18.97	23.21	24.36	24.54	21.20

ever, when the network is trained using real data augmented with our synthetic data (which we call *OysterNet*), there is a significant decrease in false pblueictions as we can see from Figs. 2.12c and 2.12f.

Since the generated dataset affects accuracy significantly, we ablate on how different parameters in our proposed geometric model affect performance in the next section.

2.3.4 Ablation Studies

We varied the parameters Θ that controlled the amount of high-frequency components and the number of layers to generate oyster models. We also varied the size of the oyster when

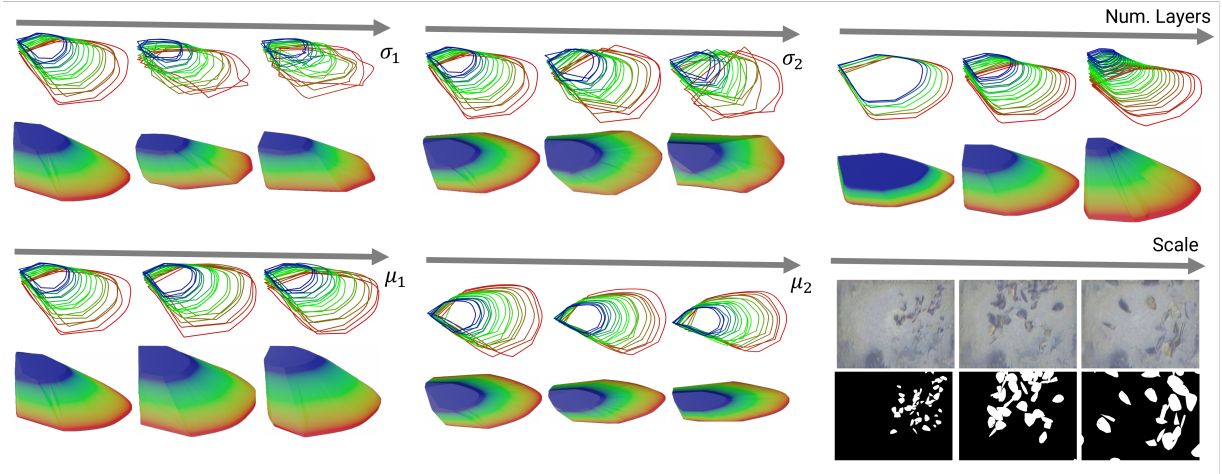


Figure 2.18: Qualitative demonstration of how different parameters of our oyster model affect the shape of the oyster. The parameters are in the same order as the experiments in Table 2.6.

rendering it in the simulation. When not specified, the parameters are set as $\mu_1 = 150$, $\mu_2 = 150$, $\sigma_1 = 150$, $\sigma_2 = 15$, $\alpha \in [15, 20]$, and $Scale \in [25, 30]$. Only one parameter is varied at a time while keeping others fixed to the values specified. No two oysters are the same with respect to size and height, therefore we chose different ranges for α and $Scale$. We see from Table 2.6a that IoU Score increases with mean of the noise and then decreases.

This is because the noise increases, and the variety of the oysters increases which would benefit the detection of the oysters. However, when the noise is too large ($\mu_1 > 150$), the oyster generated no longer looks like an oyster (See Fig. 2.18). The IoU Score drops significantly when $\mu_1 = 250$. A similar trend is observed with μ_2 and σ_1 with the IoU Score peaking around 150 (Table 2.6b and 2.6c). We also observe that with increase in standard deviation (σ_2) for X_2 (Table 2.6d), the detection accuracy drop slightly. By varying the number of layers (α) for the oysters, we notice that the change in the IoU Score is relatively minimal. Since the image that we generated is in 2D, the parameter for layers (α) does not have a significant effect (Table 2.6e) as we mostly observe oysters from far away and in mostly top view.

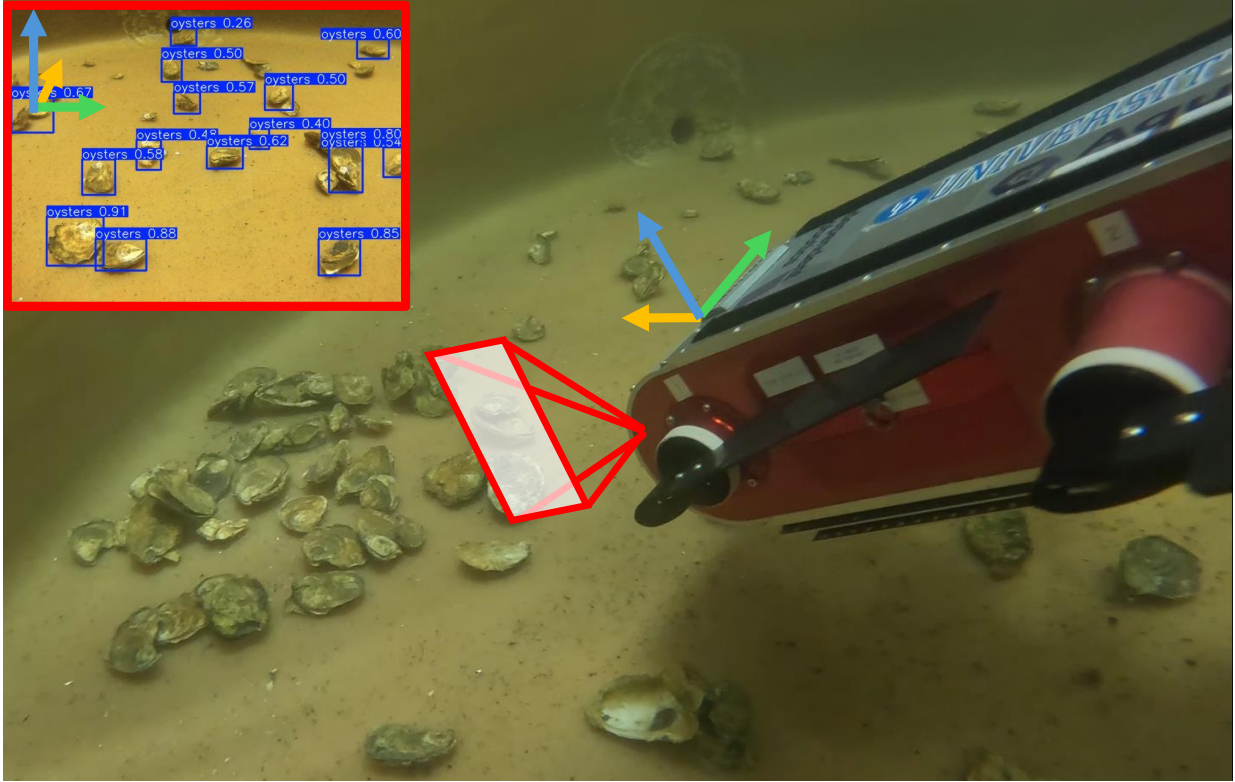


Figure 2.19: Oyster detection system deployed on an Autonomous Underwater Vehicle (AUV). The main image shows oyster detection in real-time in a shallow marine environment on the Aqua2 robot. The inset highlights the output bounding boxes around the oysters detected in situ

In Table 2.6f, we observe that the scale of the oyster has to match the size of the oyster in the image to get the optimal result. The accuracy drops when the scale of the oyster is either too large or too small. The IoU Score difference between the scale values in the ranges of 20 – 25 and 25 – 30 is only 0.18% (Tables 2.6f).

2.4 Edge-based Marine Object Detection

2.4.1 Introduction

Oyster reefs are crucial benthic environments, providing numerous ecosystem services, including water filtration, enhanced species richness, shoreline stabilization, and velocity attenuation [100–102]. Unfortunately, standing oyster stocks in areas such as the Chesapeake Bay [91] and the North Sea [92] have decreased significantly due to over-fishing, global warming, and

disease. To mitigate this devastating ecological loss, massive efforts are being made to monitor and restore oyster habitats across Europe [92], Asia [103, 104] and the United States [93–96].

Despite this effort, efficiently monitoring oyster beds for guided and enhanced restoration remains a major challenge [93, 105, 106]. Traditional monitoring methods, such as those conducted annually in Delaware Bay, New Jersey [107], and biannually in Chesapeake Bay, Maryland [108], rely heavily on destructive sampling techniques (dredging or tonging) and trained technicians to process samples. Among these reports, universal parameters such as “reef areal dimensions, reef height, oyster density, and oyster size-frequency distribution” are commonly assessed [92, 94–97]. The overall process is costly and labor-intensive, limiting the scalability of surveying efforts.

Similar challenges exist in on-bottom oyster aquaculture, where larval oysters are set on clutch and left to grow for years before harvest. Despite global growth for this economic sector [109], the industry still relies on legacy equipment [110]. Advances in autonomy, which have improved terrestrial farming with real-time detection and precision techniques [111] could help underwater farming. By using these methods, farmers could better track oyster health, abundance, and size, optimize harvesting, and avoid areas with low yields or undersized oysters [112].

In fact, by collecting videos or photographs of the beds and assessing the quantity of live and dead oysters, a census of a localized region can be produced to determine the quality of the reef and the harvestable biomass. Similar video-based methods were successfully used to assess habitat provisioning in oyster reefs and aquaculture gear to provide valuable estimates of relative abundance, species composition, and behavior [113–116]. However, these methods require highly trained individuals to manually identify and count animals over a large timespan, limiting the depth, scale, and applicability of monitoring for surveying. Autonomous systems can

detect and assess live oysters in real time using vision models, offering a promising approach to monitoring. These systems enable on-site decision-making based on live feedback, but training such models requires large datasets of oyster imagery, which are often scarce.

To address this, we leverage diffusion models to generate photorealistic synthetic images, augmenting real datasets. These combined datasets are used to train models that are then deployed in autonomous underwater vehicles (AUVs) for real-time oyster population surveys. The models trained on these combined datasets are deployed on AUVs, enabling autonomous surveys in oyster fields, and providing accurate counts of oyster populations within their biomes. A key aspect of our approach is that the detection computation is performed in real-time onboard the AUV.

The contributions of this work are as follows:

- The development and deployment of a state-of-the-art model for oyster detection with on edge implementation on the Aqua2 biomimetic AUV, to carry out tests on the oysters in real environment;
- A novel scheme for the generation of photorealistic oyster imagery;
- Open-sourcing the datasets and associated code to facilitate future research.

2.4.2 System Overview

To efficiently monitor oyster habitats in real time, we utilize the Aqua2 robotic platform (Figs 2.19, 2.21), designed for nondisruptive environmental monitoring tasks. We will first provide an overview of the Aqua2 robot, followed by a detailed explanation of the software deployed on it. To further address the challenge of limited real-world oyster imagery in reef environments,

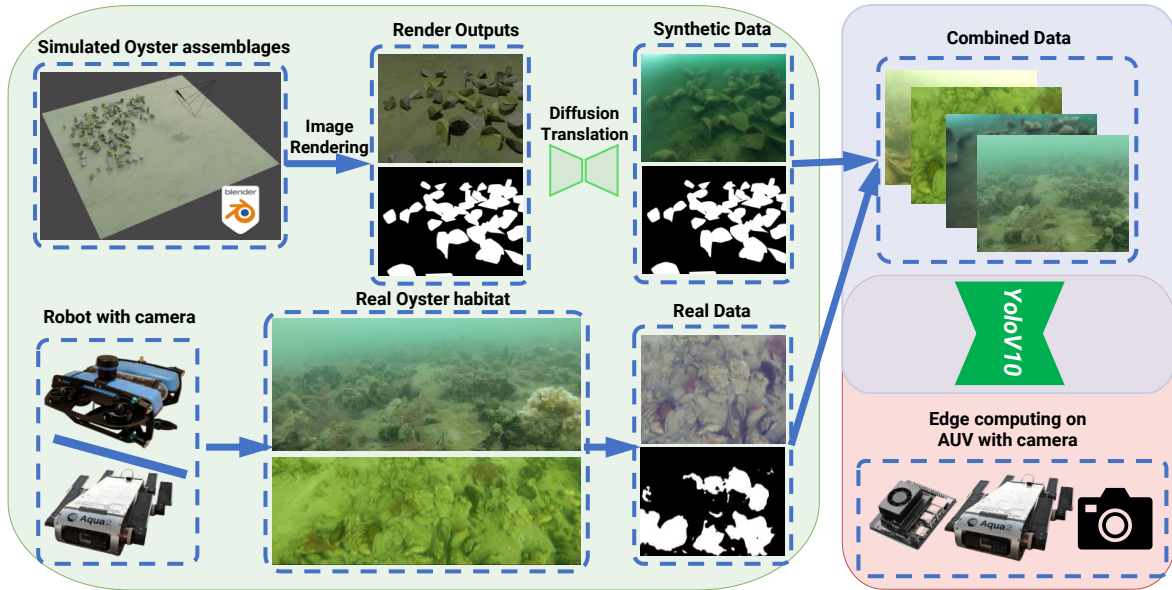


Figure 2.20: Overview of the oyster detection system. Simulated oyster assemblages are rendered using Blender and paired with ground-truth segmentation masks. Diffusion models are employed to generate synthetic oyster imagery, ensuring consistency with real oyster habitats. Real-world data are collected by the Aqua2 and BlueROV, equipped with a camera for capturing oyster images in the field. The combined dataset is used to train a YOLOv10, which is deployed on an edge computing platform on Aqua2 for real-time oyster detection.

we supplement our dataset with synthetic data, which will be discussed in the section 2.4.2.3 of this overview. The proposed system can be viewed in Fig. 2.20

2.4.2.1 Hardware Overview

Aqua2 is a biomimetic hexapedal Autonomous Underwater Vehicle (AUV), designed for minimal environmental disturbance in sensitive habitats such as coral [117, 118] and oyster reefs [119]. Unlike traditional AUVs that use thrust-based propulsion, Aqua2 employs a unique biomimetic reciprocating paddle motion with its six fins, significantly reducing sediment disruption by avoiding thrust wash. This feature is crucial to maintain water clarity in underwater environments, as sediment plumes can obscure sensors and cameras [120]. The Aqua2 vehicle is equipped with a NVIDIA Jetson Xavier NX processor, which features a 384-core NVIDIA Volta GPU with 48 Tensor Cores, delivering up to 21 TOPS (Tera Operations Per Second) of

performance at 15W of power. The processor is complemented by a 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU, providing robust processing capabilities essential for complex, real-time image processing tasks. In addition, the vehicle is equipped with three 2-megapixel cameras optimized for low-light underwater imaging and with scene-based white balancing. The two front E-CAMs capture full HD video at 65 fps, while the rear Blue Robotics camera records at 30 fps. Both have focal lengths of about 3mm, but the front cameras offer a much wider field of view and overall better performance for surveying underwater environments. On a full load, Aqua2 is capable of running for 5 hours in real world environments.

2.4.2.2 Software Overview

Aqua2’s software architecture leverages the Docker containers for easy management and deployment. NVIDIA’s Container Runtime, compatible with the Open Containers Initiative (OCI), provides GPU access as outlined in 2.4.2.1. Based on Dustynv/L4t-ml v-r36.2.0² container, we integrate Robot Operating System (ROS2), OpenCV, PyTorch v2.2.0 and Ultralytics. The system runs on ROS2 Humble middleware for real-time communication capabilities, modular design, and simplicity. OpenCV handles basic computer vision tasks and Ultralytics installs YOLOv10 and all its dependencies. We developed a custom ROS package that executes the YOLOv10 ROS node to establish the vision pipeline. During operation, Aqua2’s vision pipeline captures images from a downward-facing camera. Pre-trained YOLOv10 submodels, cached on the robot, are quickly loaded to process and compare predictions, highlighting regions of interest and publish these images on the ROS network. These results can be analyzed in real-time by the operator for decision-making or used for autonomous monitoring tasks. With the hardware and

²<https://hub.docker.com/r/dustynv/l4t-ml>

software infrastructure in place, we now turn our attention to addressing the challenge of limited real-world oyster imagery by leveraging synthetic data generation to further enhance the model’s performance.

2.4.2.3 Synthetic Image Generation

Given the challenges of acquiring real-world data, synthetic image generation has become a valuable tool for extending datasets. By using diffusion models with ControlNet, we transform a synthetic dataset, comprising segmentation masks and depth images generated in Blender, into realistic oyster imagery that closely resembles the target underwater environment, enhancing vision model performance. This method effectively bridges the gap between synthetic and real-world data, producing more representative images while retaining the flexibility of synthetic generation. The data generation pipeline in Fig. 2.22 illustrates the ControlNet inputs on the left side, consisting of target domain images, the synthetic blender images, depth images and segmentation ground truth masks rendered with blender. For further guidance a positive and negative prompt is provided. The details of the synthetic data generation pipeline are described in the following sections.

2.4.2.4 Simulation Image Rendering

To model the geometry of an oyster, we first 3D scanned ten oysters to build a mathematical model that approximates the shape in two stages: a 2D perimeter model and a stratified 3D model.

The 2D perimeter is represented using two cubic B-splines shown in (2.3) and (2.4) for the top and bottom shell halves. This model is then extended to 3D by adding depth changes across layers. High-frequency details are simplified for computational efficiency, with visual textures compensating for intricate surface variations. This approach provides a scalable way to generate



Figure 2.21: Experimental deployment of the system, surveying a wild near-shore oyster reef in Lewes, DE. Operators can be seen teleoperating the Aqua2 and analyzing the feedback.

realistic oyster models. More details of the modeling can be found in OysterNet [63]

The 3D modeled oysters are then randomly distributed within the Blender™ 3D game engine. The camera view was angled slightly to better emulate the point of view of an AUV observing the oysters in situ (Fig. 2.19), providing a more accurate visual representation of the AUV monitoring the oyster habitat.

The simulation is used to generate depth maps and the corresponding segmentation masks, which are used as the source domain in the synthetic dataset generation. Providing geometric consistency in ControlNet through the depth map indicates the locations of individual oysters within the segmentation mask.

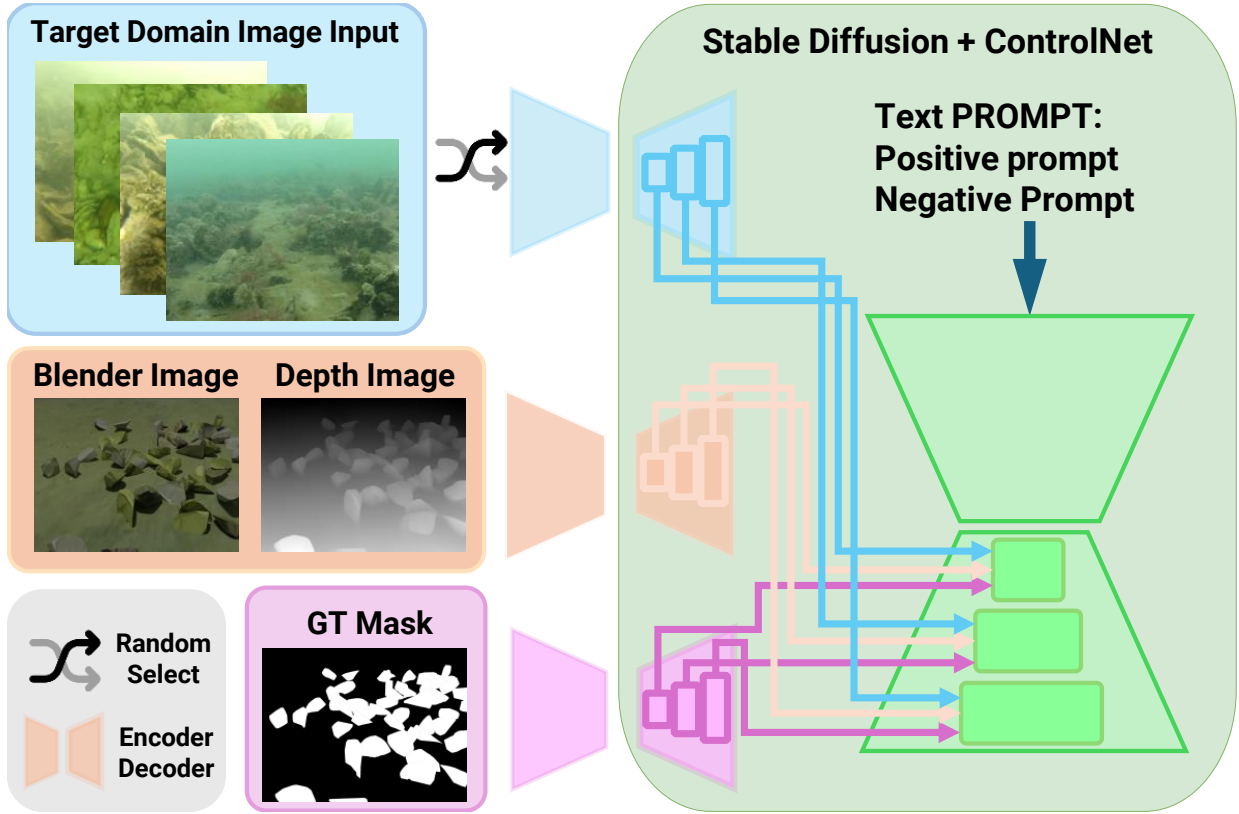


Figure 2.22: Synthetic image generation pipeline using Stable Diffusion and ControlNet. ControlNet uses randomly sampled real underwater images combined with Blender-generated images, depth images, and ground truth masks to ensure consistency. Stable Diffusion model, guided by text prompts, and ControlNet, refines the synthetic data to closely match real-world oyster environments for vision model training.

2.4.2.5 Real Data Collection

To facilitate data collection, the BlueROV platform was equipped with a GoPro camera, and the Aqua2 robot was also employed to capture video of oysters in situ at Chesapeake Bay (Neavitt, MD; 38° 44' 41.0"N, 76° 18' 22.6"W) and Delaware Bay (Lewes, DE; 38°47'20.4"N 75°09'44.6"W) respectively. During the experiments, the robot was teleoperated to decouple the detection and classification results from the challenge of navigating around oyster structures, ensuring more precise data collection. From the video, high-quality frames were extracted and manually annotated by trained experts to identify the specific locations of the oyster within each frame. This data was subsequently combined with the earlier OysterNet dataset to obtain the

final combined, fully real dataset of more than 2000 manually annotated images. Furthermore, real images are used as ControlNet inputs that represent the target domain to generate synthetic images.

2.4.2.6 Stable Diffusion and ControlNet

Images generated as detailed in Section 2.4.2.4 are provided as the source domain and iteratively selected as ControlNet inputs. Depth maps and segmentation are kept in corresponding pairs such that the segmentation and geometry complement each other. Therefore, guiding the image generation to align with the desibblue structure and spatial characteristics. Additionally, four randomized reference images from the real data collection are used as image prompts for ControlNet to fine-tune the stylistic and environmental attributes of the generated images. A carefully selected text prompt ensures that Stable Diffusion focuses on the specific target object, namely, the oysters, and it includes negative prompts to mitigate certain biases in the diffusion models. Each generated image from the diffusion model is paibblue with the corresponding segmentation mask and then used as one of the ControlNet inputs. These masks serve to help generate annotations for the synthetic oyster images, enabling them to act as the ground truth.

Combining real and synthetic data ensures a sufficiently large dataset for training. With the system reviewed, we proceed to assess the performance of the system through a series of experiments.

2.4.3 Experiments And Results

The mixed datasets obtained via the method described in 2.4.2.3 were then subsequently used to train all of the available YOLOv10 models and comparisons were run between these augmented models and the models that had only seen the real data.

2.4.3.1 Training Details

To train the various models, 4000 oyster images were synthetically generated as described in Section 2.4.2.3 with the size of 640×480 , which were then combined with the existing dataset of 2025 real images of 1920×1680 resolution. To this end, we used only 30% of the real dataset for training combined with all the synthetic dataset. We tested our method on the remaining 70% of the real dataset to report our result. We train all networks for 300 epochs with an initial learning rate of 0.01 with a cosine annealing learning rate scheduler. The Adam optimizer was used with momentum set to 0.937 and a weight decay of 0.0005. To avoid overfitting. The training batch size was 16, and the model was trained on resized images to 640×640 pixels. In terms of data augmentation, we utilized several techniques, such as flipping, scaling, and random cropping. Close mosaic augmentation was applied for the first 10 epochs, with the mosaic probability set to 1.0. The model was further fine-tuned using automated augmentations, such as *randaugment* and random erasing with a probability of 0.4. During training, the object detection model was evaluated against a validation set, with an IoU threshold of 0.7. The maximum number of detections per image was capped at 300 to ensure efficient processing of the oyster habitat scenes.

2.4.3.2 Experimental Results

Extensive testing of various YOLOv10 submodels was carried out at the Marine Operation boat basin facility of the University of Delaware in Lewes, Delaware Fig. 2.21. This test focused on assessing the performance, accuracy and adaptability of these submodels in real-world scenarios, ensuring their robustness in handling the specific environmental and operational challenges presented by the location.

Table 2.7: Quantitative Comparison of All YOLOv10 Sub-Model Performance

Model	Inference Time (ms)	Frequency (Hz)	mAP50	mAP50-95
YOLOv10-N	54.9	9.2	0.458	0.234
YOLOv10-S	71.0	8.9	0.557	0.314
YOLOv10-M	158.1	4.8	0.642	0.468
YOLOv10-B	214.8	3.8	0.645	0.416
YOLOv10-L	264.1	3.2	0.657	0.430
YOLOv10-X	375.2	2.3	0.642	0.424

Lighter models like YOLOv10-N achieved faster inference times and higher frequencies, while more complex models like YOLOv10-X were slower. For example, YOLOv10-N processed a frame in 55 ms with a frequency of 9.2Hz (Table 2.7a), while the YOLOv10-S model, although more accurate, had a frequency of 8.9Hz with a processing time of 71 ms (Table 2.7b).

The YOLOv10-M model, suitable for general detection, processed frames in 158 ms at 4.8Hz (Table 2.7c). YOLOv10-B offered a good compromise, with a processing time of 214 ms and a frequency of 3.8Hz (Table 2.7d). For more accuracy, YOLOv10-L took 264 ms per frame at 3.2Hz (Table 2.7e). Lastly, YOLOv10-X, the most computationally intensive model, was processed at 372 ms with a frequency of 2.3Hz (Table 2.7f).

The qualitative results in Fig. 2.23 from testing multiple YOLOv10 sub-models demon-

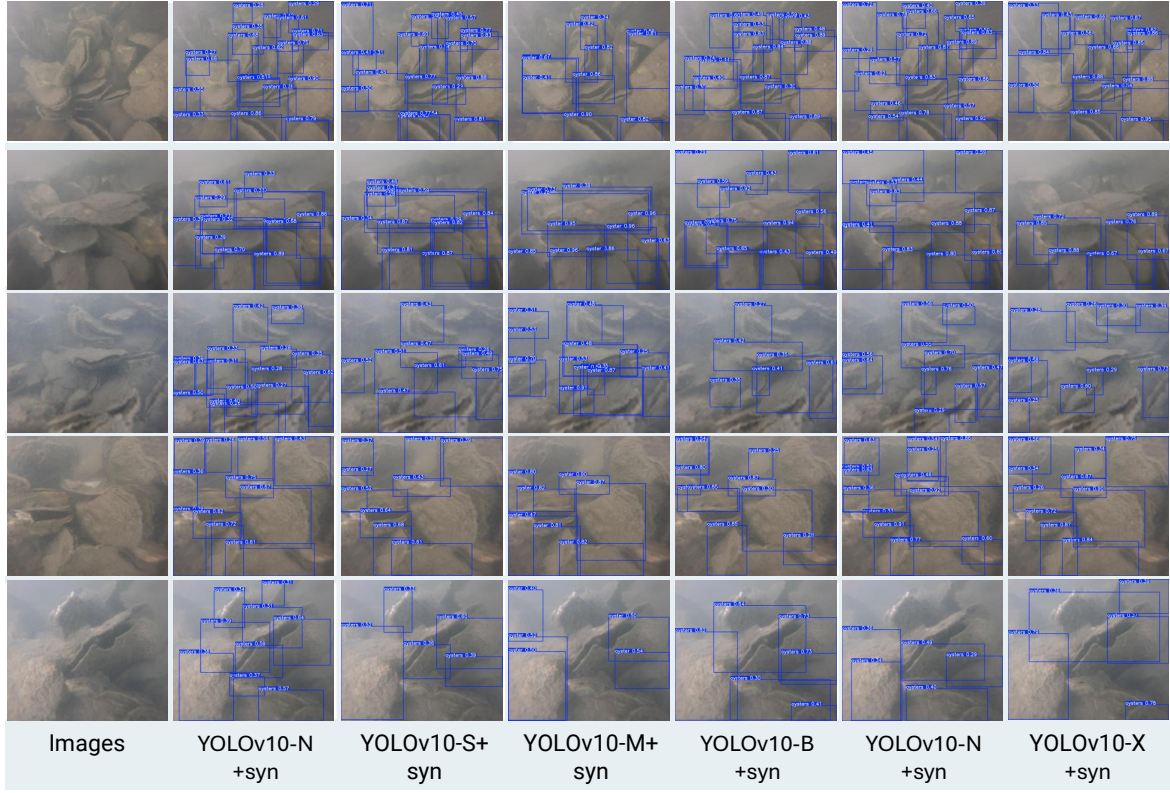


Figure 2.23: Qualitative Comparison of All YOLOv10 Sub-Model Performance. *This image is best seen in color on a computer screen at 400% zoom.*

strate a clear trend: generally, larger models achieve better detection accuracy, as indicated by higher mAP50 and mAP50-95 values. As depicted in Table 2.7, YOLOv10-L outperforms smaller models like YOLOv10-N, with mAP50 scores of 0.657 compared to 0.458. However, this trend does not hold for the largest model, YOLOv10-X, where the performance slightly drops, suggesting that the size of the dataset might not be sufficient to fully leverage the complexity of the larger network. This indicates a point of diminishing returns with increasing model size, where the added complexity does not necessarily translate into better detection performance.

When comparing models trained with and without synthetic data in Fig. 2.24, it is evident that incorporating synthetic data improves the overall detection results across the board. As depicted in Table 2.8, YOLOv10-L with synthetic data achieves an mAP50 of 0.657 compared to 0.638 without synthetic data. However, this improvement comes with a trade-off: the confidence

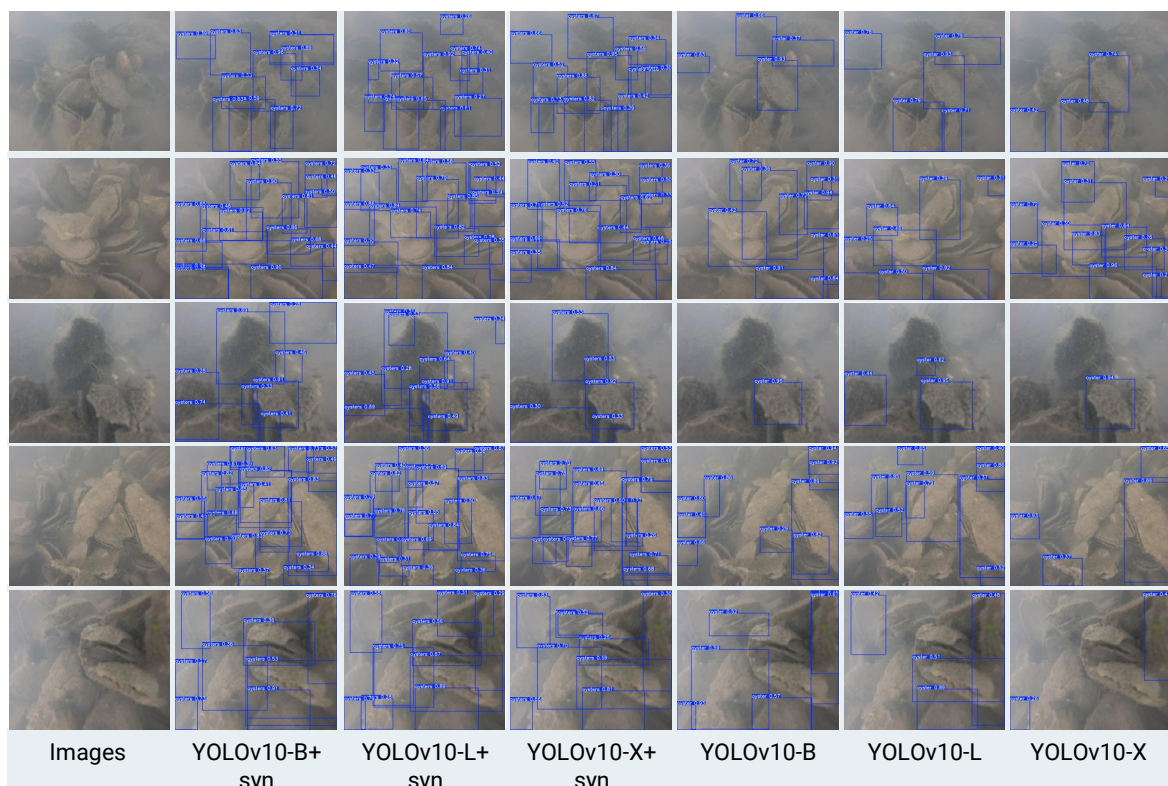


Figure 2.24: Qualitative Comparison of YOLOv10-B, YOLOv10-L, and YOLOv10-X Models with and without Synthetic Data. *This image is best seen in color on a computer screen at 400% zoom.*

in these detections, as reflected in the mAP50-95 scores, is slightly lower with synthetic data.

This suggests that while synthetic data can enhance detection capabilities, it might introduce variability or noise that affects the precision of the model's predictions.

Table 2.8: Comparison of YOLOv10-B, YOLOv10-L, and YOLOv10-X Models with(S) and without(R) Synthetic Data. (S): the model is trained on combined synthetic and real data. (R): the model is trained on real data only

Model (S)	mAP50	mAP50-95	Model (R)	mAP50	mAP50-95
YOLOv10-B	0.645	0.416	YOLOv10-B	0.639	0.472
YOLOv10-L	0.657	0.430	YOLOv10-L	0.638	0.478
YOLOv10-X	0.642	0.424	YOLOv10-X	0.630	0.480

Chapter 3: Marine Information-driven Vision-based Navigation

3.1 Introductions

Underwater navigation is a challenging task that has been the subject of numerous research studies [37]. In particular, interest in marine exploration and scientific sampling applications [121–123] has significantly increased the demand for efficient underwater navigation methods for exploration and data gathering. Many classical navigation methods developed for ground and aerial vehicles have been adapted to underwater environments [40, 124]. Navigation in an underwater environment presents unique difficulties, such as limited visibility, dynamic environments, and a lack of cost-efficient, accurate localization systems. Underwater vehicles are commonly used for data collection to monitor environmental phenomena. To improve the efficiency of data collection, several complete coverage path planning methods have been developed [125, 126] and adapted to underwater environments [41, 46, 48]. However, these approaches typically rely on partial or complete knowledge about the environment or assume the robot can localize itself.

The absence of reliable localization [127] requires new approaches for tackling underwater navigation or exploration problems. Towards this, vision-only based methods have been proposed for surveying coral reefs [49] and shipwrecks [50]. Nevertheless, these approaches are domain-dependent as they have been trained over specific domains and objects and thus require retraining navigation policy each time a new Object of Interest (OOI) is being surveyed.

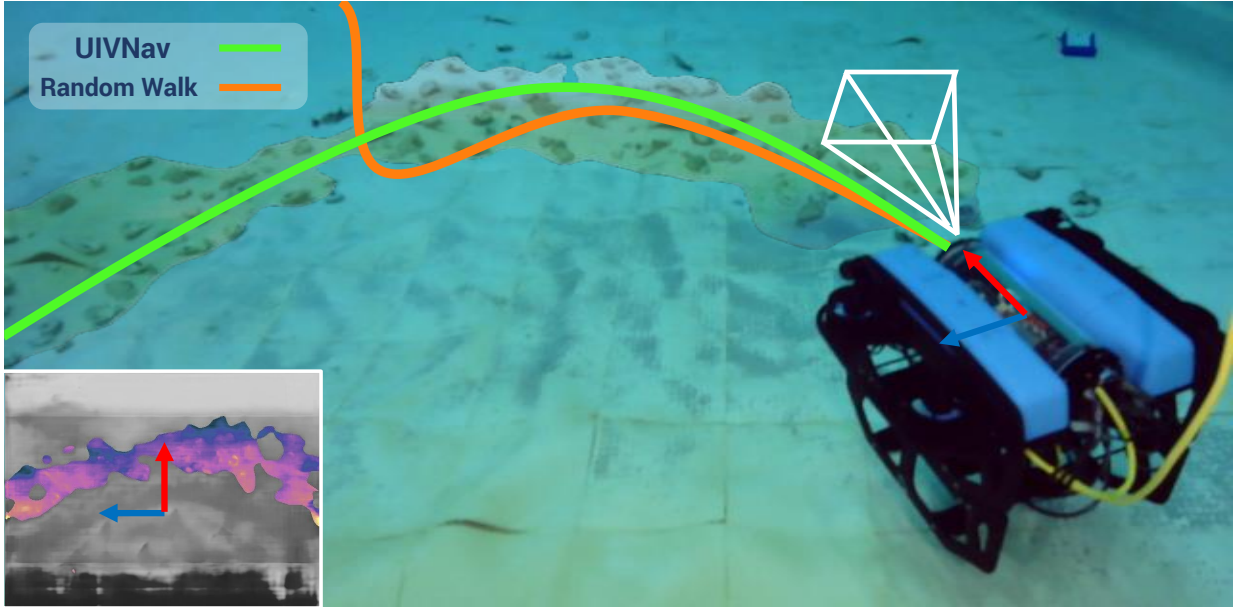


Figure 3.1: Example of two trajectories of BlueROV surveying oyster-reef in a pool. On the bottom left is the current frame in the intermediate representation that the robot observes.

We propose a new Underwater Information-driven Vision-based Navigation via Imitation Learning (UIVNAV) method to address some of these challenges. Our method focuses on information gathering as the main objective — we want to visit regions with OOI while avoiding obstacles. UIVNAV is a type of imitation learning that uses only RGB camera information and generates an Intermediate Representation (IR) of each frame, which humans then use to label corresponding actions. IR allows us to train domain-invariant navigation policies to work across different OOI. Moreover, this method does not rely on localization but only on IR. We have compared our method with baseline navigation approaches – random walk and complete coverage. The results indicate that UIVNAV outperforms random walk and even surpasses localization-based complete coverage approach by surveying on average 36% more oysters when traveling the same distances.

The main contributions of this work are:

1. The development of a domain-invariant navigation approach. This means that the naviga-

tion can gather information regardless of the type of domain being presented, reducing the need for retraining the navigation policy for new environments. We show this by deploying UIVNAV on two different OOI oysters and large rock reefs, and in different domains the simulation and the pool. This provides a more generalized solution that can be applied in a wide range of underwater environments.

2. A robot learns how to navigate successfully while covering (“seeing”) as much of the OOI as possible — without relying on localization. This is an example of a new local navigation technique that would aid a class of global exploration and coverage algorithms. As navigation computational infrastructures (e.g., SLAM and related techniques) are reaching a high level of sophistication, the need arises to perform navigation and solve various tasks while doing so.
3. Deployment of the proposed method in real-time in a large pool for surveying an oyster shell bed — demonstrating the feasibility of real-world application of UIVNAV (Fig. 3.1).

3.2 Proposed Approach

Our method, UIVNAV, is designed to maximize data collection for OOI and avoid obstacles without relying on localization. Our proposed method is based on imitation learning and is influenced by intuition used by human divers who do not rely on localization when collecting data. The backbone of the UIVNAV is the intuition that when using only vision-based navigation to collect more data and avoid obstacles, two important cues are intrinsically used by humans: (1) recognition of the OOI — segmentation of the OOI, and (2) spatial understanding of proximity of the obstacles or OOI — depth estimation. We use this intuition to design a framework that

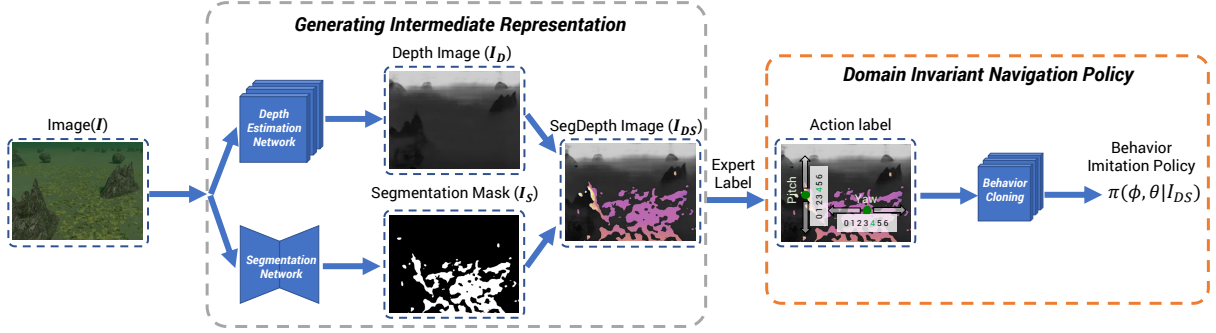


Figure 3.2: An overview of our approach. UIVNAV consists of two phases: Generating Intermediate Representation (gray box) and Data Invariant Navigation Policy (orange box). Image (I) is fed into the depth estimation network and the segmentation network. The output from the depth estimation network (I_D) and the output from the segmentation network (I_S) are used to generate the intermediate representation (I_{DS}). This is fed into an imitation learning network to train a navigation policy that predicts yaw and pitch changes. Due to training on (I_{DS}) the navigation policy is domain invariant and will not require restraining nor labeling with new OOI. achieves this objective.

An overview of the proposed approach is illustrated in Fig. 3.2. Our two-stage system consists of *Generating Intermediate Representation* and *Domain Invariant Navigation Policy*. In the first stage, the robot’s images (I) are fed into the depth estimation network and the segmentation network simultaneously. The outputs from the depth estimation network (I_D) and the outputs from the segmentation network (I_S) are used to generate the intermediate representation (I_{DS}). Creating the intermediate presentation (I_{DS}) aims to eliminate domain-specific information and allow the system to gather information for different OOI while avoiding obstacles. We assume that I_S and I_D are available or can be generated using existing methods [63, 88, 128, 129].

In the second stage, the Domain Invariant Control Policy phase, we label the intermediate presentation frames (I_{DS} from stage one) with actions (pitch and yaw changes) for obstacle avoidance and information gathering. With the labeled I_{DS} , we train a behavior cloning network that controls the robot to maximize the information gain for the OOI while avoiding obstacles. We present each stage in the following subsections.

3.2.1 Intermediate Representation

The networks that generate the intermediate representation must be altered with other state-of-the-art networks for various applications. For different domain work (air, ground and oceanic), and tasks (coral monitoring, aerial forester monitoring), the dataset used to train those networks shall also be different. We will briefly go through the networks and datasets we used and then define our intermediate representation in this subsection.

3.2.1.1 Depth Estimation Network

We need to acquire a corresponding dataset to train a depth estimation network in the underwater oyster reef environments. We utilized a convenient oyster simulation environment, OysterSim [130], to create the dataset consisting of 2000 image pairs of RGB images and the corresponding depth images(ground truth from the simulation). We adopted the monocular depth estimation network from Keras [131], with Unet [88] backbone. We used Inverse Huber Loss [132] for the depth estimation network. Sample output from the depth estimation network (I_D) can be seen in Fig. 3.2.

3.2.1.2 Segmentation Network

Similarly, we utilized OysterSim [130] to create the dataset of 2000 image pairs of RGB images of oyster reefs and the corresponding segmentation masks(ground truth from the simulation). We used the state-of-the-art oyster detection network, Oysternet [63], for segmenting the OOI. Sample output from the segmentation network (I_D) can also be seen in Fig. 3.2. We want to generate SegDepth Image (I_{DS}) for all images (I) in the Dataset ($\mathcal{D}$). Depth image (I_D) is a grayscale image with only 1 channel. We first want to save only the depth information for the

OOI (I_{D_OOI}) and only the depth information for non-OOI (I'_{D_OOI}). We then used the color mapping function (M) from `matplotlib` [133] that maps the 1-channel I_{D_OOI} to 3-channel (RGB) I_{DS_OOI} . The depth information is still preserved in the 3-channel image and its color (3-channel information) represents the region of interest. We stack (repeat) the I'_{D_OOI} three times to get the 3-channel depth information (I'_{DS_OOI}) for non-OOI. Finally, we combine these two 3-channel information to get our SegDepth Image (I_{DS}).

Algorithm 1 Generating SegDepth Image

```

1: for  $I \in \mathcal{D}$  do
2:    $I_{D\_OOI} = I_D \ \& \ I_S$ 
3:    $I'_{D\_OOI} = I_D \ \& \ \neg I_S$ 
4:    $I_{DS\_OOI} = M(I_{D\_OOI})$  ▷ Mapping function
5:    $I'_{DS\_OOI} = stack(I_{D\_OOI})$  ▷ stack 1-channel 3 times
6:    $I_{DS} = I_{DS\_OOI} + (I'_{DS\_OOI})$ 
7: end for
8: return  $I_{DS}$ 

```

3.2.2 Domain Invariant Navigation Policy

To learn our Domain invariant navigation policy, THE UIVNAV approach follows a similar pipeline as the behavioral cloning method presented in the literature [49, 50, 52, 54]. The major distinctions are the navigation policy input of the network and the labeling objectives. We use the Intermediate Representation (I_{DS}) instead of the RGB image as our input data for imitation learning. We assume the robot is positioned at least 5m above the OOI, has a front-facing camera and the environment contains obstacles that can exceed 5m height. This is a typical case for top-view data collection common in oyster, coral, or forestry monitoring applications. We generate (I_{DS}) on the raw images collected from such an environment to perform the navigation policy labeling. A human diver performs the labeling through a Graphical User Interface (GUI) that shows the current frame and possible classes for yaw and pitch changes (see Action Label in

Fig. 3.2).

The labeling objectives of the human diver that collects data over OOI are as follows: (1) prefer to move towards OOI; (2) move towards larger areas of the OOI that are further away; (3) avoid obstacles. Yaw and pitch values are each encoded as a discrete set of seven class labels similar to the related works suggested in the literature [49, 50]. A Convolutional Neural Network (CNN) with a ResNet-50 backbone is trained on 15000 labeled input images (I_{DS}) to predict these relative heading changes in yaw and pitch.

We denote the $f_\theta(I_{DS})$ and $f_\phi(I_{DS})$ for the categorical distribution prediction (vector of size 1 by 7 for the probability of each class) for yaw and pitch respectively with the input of I_{DS} from the behavior imitation policy. There are seven class prediction labels with a mean of 3 for the yaw and pitch which are described as C_θ and $C_\phi \in \{0, 1, 2, 3, 4, 5, 6\}$. C_θ corresponds to desired yaw changes of $\{3\delta_\theta, 2\delta_\theta, \delta_\theta, 0, -\delta_\theta, -2\delta_\theta, -3\delta_\theta\}$ with respect to the current image frame. Here the positive change represents a clockwise heading change and the negative change represents a counterclockwise heading change with the amount of $\{\delta_\theta, 2\delta_\theta, 3\delta_\theta\}$. Similarly C_ϕ corresponds to the desired pitch changes with respect to the current image frame. Here the positive change represents an upward-heading change.

When applying the behavior imitation policy, we choose $\delta_\theta = 5^\circ$, $\delta_\phi = 5^\circ$ as sharp changes are undesirable. As suggested by [54], we also performed label smoothing on the ground truth label. We treat $f_\theta(I_{DS})$ and $f_\phi(I_{DS})$ separately as we are training the policy while using the same loss function below. As we believe the wrong prediction for one axis shall not penalize the right prediction for the other axis. The following loss is used to train the policy $\pi(\phi, \theta|I_{DS})$:

$$\mathcal{L}_{loss}(\mathcal{D}, w) = \lambda * \mathcal{L}_{CCE}(\mathcal{D}, w) + (1 - \lambda) * \mathcal{L}_{KL}(\mathcal{D}, w) \quad (3.1)$$



Figure 3.3: Example of BlueROV surveying reef of oysters in simulation (translucent mask is applied on the oyster reef for better visualization.)

The λ here is a weight factor for the loss function. We choose $\lambda = 0.1$. $\mathcal{L}_{CCE}$ is categorical cross-entropy loss and $\mathcal{L}_{KL}(\mathcal{D}, w)$ is KL divergence from [134].

The resulting predicted 7-class discrete network values for yaw and pitch are transformed into low-level actuator commands that the robot’s motion controllers carry out during inference.

3.3 Experimental Evaluation

We use the oyster reef monitoring application as a case study for demonstrating the capabilities of UIVNAV. We performed extensive quantitative and qualitative comparisons of our method with two baseline methods — (1) Brownian Bridge (BB) [135] random walk and (2) an offline complete coverage method based on Boustrophedon Cell Decomposition (BCD) [125]. Brownian Bridge-based motions have been used to study animal trails [136] and can have the potential to replicate the distribution of natural phenomena such as oyster or coral reef growth patterns. It has also been widely used in exploration tasks [137, 138]. As our goal is to maximize the collected data of OOI, we compare our method to the complete coverage path planning method, which is the state-of-the-art approach for completely covering the area when the prior map is given. In practice, farmers or scientists will perform a lawnmower pattern — naive version of BCD pattern to navigate their vehicles and collect data or dredge oysters. For the BCD

algorithm, we only assume the boundaries of the region and obstacle locations are given but not the distribution of the OOI. Note that we do not use any prior information about the environment when running UIVNAV.

In this section, we will first conduct an evaluation in a simulated environment with oysters. Additionally, to demonstrate the generalizability of our method to navigate over different OOI, we test in a rock reef environment. Finally, we show results from a deployment of UIVNAV on a BueROV underwater robot in a pool to assess the real-world viability of our approach in real time.

3.3.1 Simulation

We conducted extensive testing and validation in simulated environments. 15,000 training samples and 5,000 validation samples composed of images, depth ground truth, and oyster segmentation ground truth are rendered in the simulation. Based on the intermediate representation we computed, we manually labeled this dataset of pitch and yaw labels for training as discussed in section 3.2. We trained Resnet-50 as our backbone with a learning rate of 0.001 coupled with Adam Optimizer. We used a weighted loss which is explained in Section 3.2. For detecting the OOI, we trained OysterNet [63] with generated images and oyster segmentation ground using a learning rate of 0.001 with decay. We use the Adam optimizer coupled with the Jaccard loss function. For the depth estimation network, we trained a Unet [88] with generated images and depth ground using a learning rate of 0.001. We used a batch size of 32 and trained the network for 100 epochs. To generate underwater oyster and rock reef environments we used an open-source oyster simulation framework based on Blender [26] game engine, OysterSim [130]. We use a model of the BlueROV robot in the simulation as shown in Fig 3.3. We have generated different

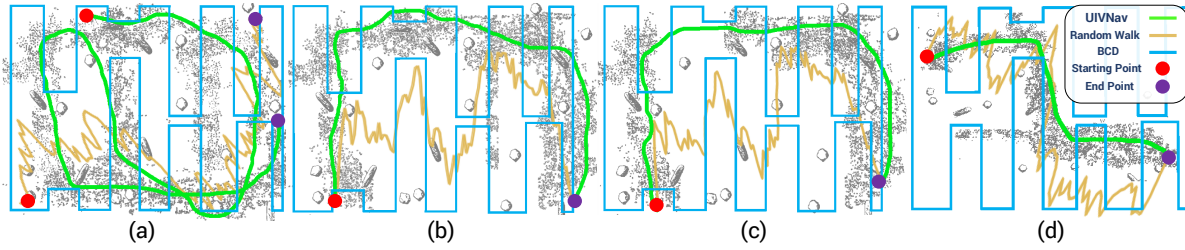


Figure 3.4: Comparing various exploration strategies across four oyster reef patterns– (a) Grid world, (b) E-shape, (c) Disconnected Paths, (d) Branching Corridor.

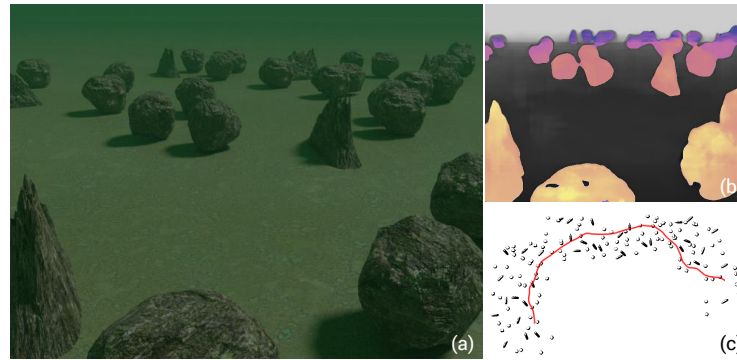


Figure 3.5: UIVNAV surveying reef of rocks OOI with no additional human-expert labeling or training. (a) The current frame of the robot field of view in OysterSim simulation. (b) Intermediate representation of the current frame that the robot uses to make navigation decisions. (c) The complete trajectory over a rock reef.

possible distributions of oyster reefs and one sample of rock reef. When designing the scenarios we took into account that underwater natural reefs – oyster or coral — usually do not grow separately but they tend to be in large groups. Nevertheless, they can have different discontinuous patches. As such, our test scenarios consist of the following instances:

1. Grid World - a reef with connected continuous patches of oysters with multiple branches (Fig. 3.4a).
2. E-shape - C-shaped wide oyster reef patch with a narrow oyster patch in the middle (Fig. 3.4b).
3. Disconnected Paths - a reef consisting of two large oyster groups that are separated by a large sandy area (Fig. 3.4c).

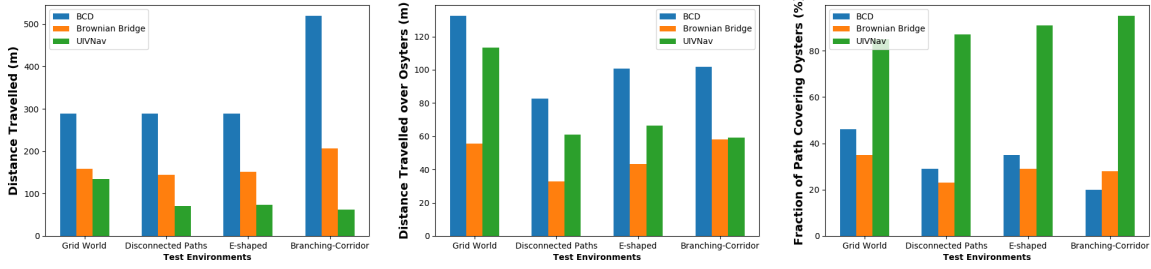


Figure 3.6: Quantitative Evaluation of Different Strategies on Different Environments. BB method travels longer distances moving on fewer areas with oysters compared to UIVNAV. Even though compared to BCD, the UIVNAV method collects on average 29% less oyster data it travels 70% less distance. This shows that UIVNAV trajectories survey on average 36% more oysters than the BCD method when traveling the same distances.

4. Branching Corridor - a reef with wide and narrow oyster patches branching from the opposite sides of the reef (Fig. 3.4d).
5. Rock Reef - a reef of large rocks with continuous C-shape (Fig. 3.5c).

3.3.2 Results

Since our objective is to maximize the data collected we have measured *Distance Traveled* with *Distance Traveled over Oysters* in meters and calculated the *Percentage of Oysters* contained within each generated path assuming the robot travels at a consistent speed. As depicted in Fig. 3.6, our results prove the hypothesis that our method UIVNAV is better than the random walk. Compared to the complete coverage method, the results indicate that a robot using UIVNAV collects on average 29% less oyster data while traveling on average 70% less distance. This shows that UIVNAV trajectories survey on average 36% more oysters than the complete coverage method when traveling the same distance.

In each of the four test environments, we show that UIVNAV makes qualitatively more efficient decisions (Fig. 3.4). In *Grid World*, UIVNAV almost completely covers the oyster reef

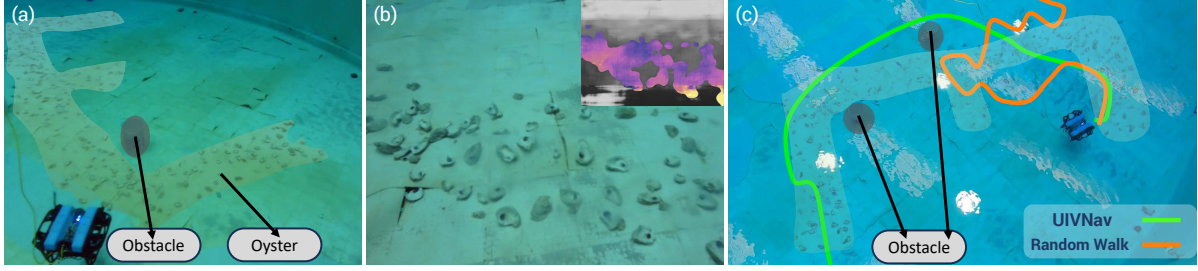


Figure 3.7: (a) BlueROV2 Deployed in a Neutral Buoyancy Pool with Yaw and Pitch control with E-shape oyster shells distributed at the bottom. (b) BlueROV’s field of view. (c) Experimental result of two trajectories of BlueROV surveying oyster shells in a pool.

without navigating over sand patches and avoids all of the obstacles (Fig. 3.4a). In *E-shape*, we can observe that UIVNAV chooses to turn on a wider region towards the endpoint rather than traveling through the middle patch — demonstrating the ability to infer “future reward” (Fig. 3.4b). *Disconnected Paths* scenario, UIVNAV is able to navigate through the gap and locate a new OOI in sight, enabling it to effectively survey over the remaining portion of the oyster reef (Fig. 3.4c). The last scenario with oyster reef — *Branching Corridor*, demonstrates UIVNAV’s ability to select wider regions with oyster surface to continue exploration (Fig. 3.4d).

We also qualitatively evaluate UIVNAV on the *Rock Reef* scenario to show that the navigation policy is domain invariant. To obtain the intermediate representation for different OOI, we re-train the segmentation network for detecting the rocks. On a newly generated IR, we directly apply a trained navigation policy. Robot successfully explores new OOI without the need for additional human-expert labeling or re-training of the policy (Fig. 3.5).

3.3.3 Lab Deployment

3.3.3.1 Hardware Setup

We conducted pool experiments with the BlueROV2 robot in Heavy Configuration [139] with 8 thrusters for motion in all 6 degrees of freedom. The BlueROV2 is equipped with the

following onboard sensors: TDK Invensense ICM-20602 Inertial Measurement Unit (IMU), Waterlinked DVL-A50 Doppler Velocity Logger (DVL), and a Bosch BMP280 pressure sensor, an onboard Low Light HD USB Camera with 1080p resolution. The robot works on the *Navigator* flight controller with ArduSub as the software suite. MAVLink protocols are used for interfacing the BlueROV2, using the `pymavlink` library. The BlueROV2 is controlled by giving Pulse-Width Modulation (PWM) to the thrusters through I2C interface PCA9685 — a 16-Channel PWM Module.

The BlueROV2 is connected through the tether to the Ground Central Station (GCS) - Dell Precision 7560. The GCS is equipped with the NVIDIA RTX A2000 GPU.

3.3.3.2 Software Setup

All drivers for processing the data and autonomous control of the robot are ROS-based software written in Python. The UIVNAV was deployed on GCS, executing navigation commands in real-time at a frequency of 0.5Hz. The commands generated from the UIVNAV network are mapped to the PWM inputs for thrusters to change the BlueROV2's pitch, yaw, surge, and heave. The network gives us a set of commands (c) as single digit numbers, which determine the desired velocity (ν) — ($\nu = c * \alpha$), where α is a constant determining factor by which the magnitude of speed changes. The input value for the PWM input is calculated as follows:

$$\text{PWM}_\eta = 1500\mu\text{s} + k_\eta \nu \quad (3.2)$$

where $1500\mu\text{s}$ keeps the thrusters at neutral position, η denotes the concerned degree of freedom and k denotes the gain for each direction which is calculated experimentally.

The main assumption in UIVNAV is the availability of segmentation and depth masks for generating the intermediate representation. Since the lab setup (Fig. 3.7) is very different

from the simulated oyster reefs, we retrained the detection network for OOI. We first manually collect 300 images of the oysters from the bottom of the pool and label their segmentation masks. These images were trained with the OysterNet [63] model for detecting OOI with a learning rate of 0.001 with decay. On the resulting intermediate representation without any additional training the UIVNAV has been executed to produce navigation policy. We also use the Adam optimizer along with the Jaccard loss function. The actions involve a set of pitch, yaw, and surge commands. Additionally, the BlueROV2 was programmed to hold its depth at 1 ft above the pool bottom using the feedback of its pressure sensors. Notice that, we did not retrain our navigation policy and depth estimation network. We only re-train the detection network for OOI here.

3.3.3.3 Experiment setup

We distributed up to 300 oyster shells at the bottom of the neutral buoyancy tank. The neutral buoyancy tank is 50 feet across and 25 feet deep. The E-shape pattern of the oyster shell patch is bounded by (Fig. 3.7a) a 40x20 feet area — which occupies half of the tank. The breadth of the oyster patch averages 1 to 2 feet, which fits nearly 3-5 shells.

The real-world deployment of the robot posed several challenges that significantly increased the task’s difficulty. Namely, the robot’s motion was influenced by the altitude at which front camera images were captured, drift from turning and surging, and external forces that affected the robot’s turning radius and positioning. Nevertheless, the robot successfully surveyed the oyster shells using the navigation policy derived from our method (see Fig. 3.7c). In comparison as a baseline, we showed a random walk in Fig. 3.7c in yellow.

In this experiment, without prior knowledge of the environment and localization, the UIVNAV moves the robot towards oyster reefs (Fig. 3.7b)— maximizing the information gain over

OOI while avoiding two obstacles (two water buckets). By using random walk, the robot also traverses through a quarter of the OOI; however, the trajectory went outside of OOI.

Chapter 4: Oceanic Real and Virtual Image Segmentation

4.1 Introduction

Advanced computer vision systems enable robots to comprehend scenes by analyzing pictorial information. Although this may work well in many cases, various real world scenarios can pose challenges due to the diverse nature of light propagation and scene complexity. The images received by human and machine visual sensors are often a complex superposition of direct imaging, reflections, refractions, scattering, and the likes. This complexity is particularly pronounced at the interfaces between different media. Consequently, the input to vision sensors contains real image regions formed by direct imaging and virtual areas typically formed by distortion, reflection, or refraction. Inherent complexities in differentiating between real and virtual regions pose challenges for both low-level and high-level computer vision tasks (see Fig. 4.1), such as depth and pose estimations, visual odometry, object recognition, semantic segmentation, etc. Therefore, real-virtual image segmentation to localize visually credible areas is crucial for marine robots, particularly in multimedia scenarios.

Novel deep learning-based methods have significantly improved the performance of numerous computer vision tasks such as object recognition [140] and detection [141], depth estimation [142] and 3D navigation [143–145]. However, our investigation indicates that real-virtual image segmentation in multimedia scenarios remains a relatively underexplored area. One primary challenge is the scarcity of publicly available datasets for real-virtual image segmentation in

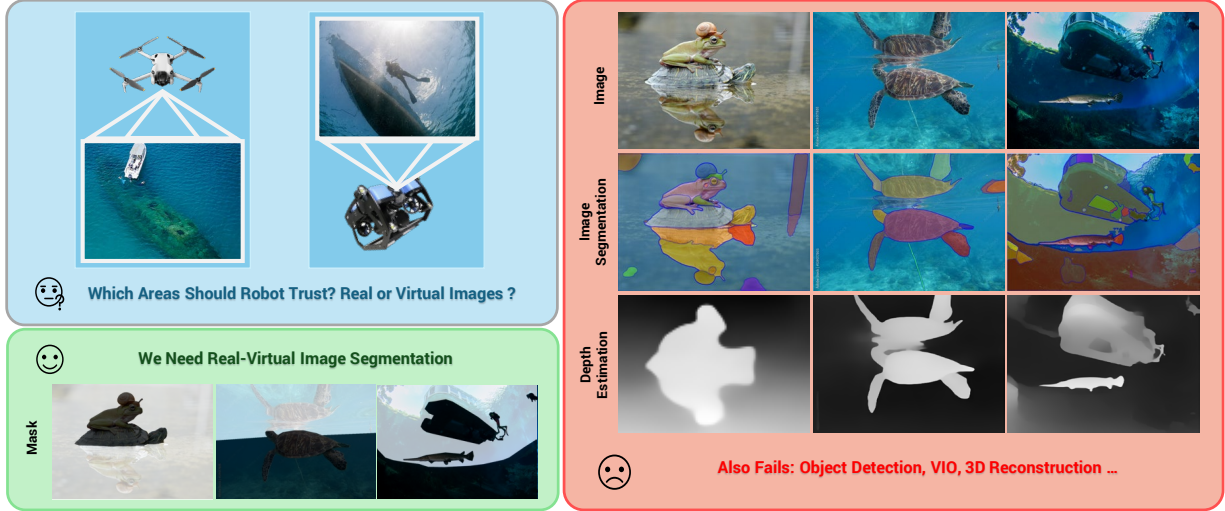


Figure 4.1: Various computer vision tasks fail to varying degrees in multi-media scenarios due to virtual images formed by the diverging reflection or refraction of light rays. The real-virtual image segmentation mask is crucial to provide the robot with credible region information in the visual sensor input.

multimedia scenarios. Additionally, existing underwater simulators often overlook the intricate modeling of water-air interface imaging, making synthetic data extremely difficult to acquire. At the same time, data-driven methods are heavily relying on large-scale datasets. Secondly, the absence of an efficient annotation pipeline for the real-virtual image segmentation task, particularly for images with visual scene ambiguity, results in manual annotation being more laborious than common computer vision tasks. Moreover, achieving this task usually involves the use of Unmanned Aerial Vehicles (UAVs), Autonomous Surface Vehicles (ASVs), Unmanned Underwater Vehicles (UUVs), or even amphibious robots, which makes the acquisition of images from multiple domains costly.

To address these challenges, we develop *AquaSim* to efficiently generate training data and simulate realistic underwater and water-air interface scenarios for related research. This comprehensive platform is tailored to train neural networks for water surface segmentation. Another key contribution is *MARVIS*, a robust and efficient pipeline for real-virtual image segmentation.

Our proposed network incorporates temporal information and the epipolar geometric constraint into its supervised learning pipeline. We innovatively propose the Local Motion Entropy (LME) kernel, which serves as a pivotal layer in the encoder of the *MARVIS* model. Based on the violation of the epipolar geometric constraint in the refractive virtual image area, we design an Epipolar Geometric Consistency (EGC) loss as weak supervision to further improve the performance of the pipeline when stereo image pairs are available. LME and EGC provide domain-invariant information, enabling robust segmentation in two domains (synthetic and real) without re-training. With comprehensive motion & geometry-aware learning on 3012 pairs of synthetic images and their corresponding masks, *MARVIS* achieves SOTA performance in both synthetic domain (IoU over 94% and a F1-Score over 96%) and real world domain (IoU over 78% and a F1-Score over 86%) despite having such a light architecture. Furthermore, *MARVIS* offers significantly faster inference rates: over 43.4 FPS on a NVIDIA™ RTX 4070 GPU and 8.0 FPS on an Intel™ Core i9-4.10GHz CPU.

4.2 Underwater Simulator including Dynamic Water-air Interface Distortion

AquaSim leverages the Blender™ game engine to simulate environments for generating synthetic datasets (see Fig. 4.2) used in training neural networks for real-virtual image segmentation. By inputting desired parameters, such as water color, wave characteristics, and reflection properties (examples in Fig. 2, *AquaSim*), users can create tailored datasets alongside corresponding ground-truth image masks. This streamlines the development of real-virtual image segmentation networks. Comparison with other simulators is detailed in Table 4.1. *AquaSim* pioneers realistic water surfaces and uniquely provides ground-truth masks for real-virtual image alignment.

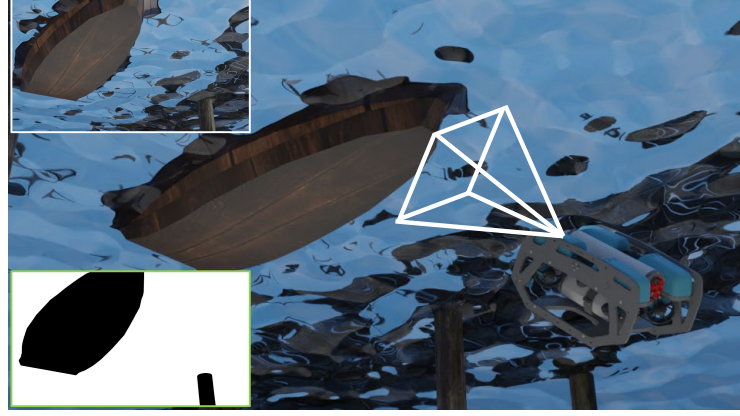


Figure 4.2: BlueROV collecting data for our synthetic dataset in *AquaSim*. The inset shows a sample image captured from the camera on a BlueROV. The top left displays the feed from a front-facing RGB camera, while the bottom left shows the real image’s corresponding ground truth masks.

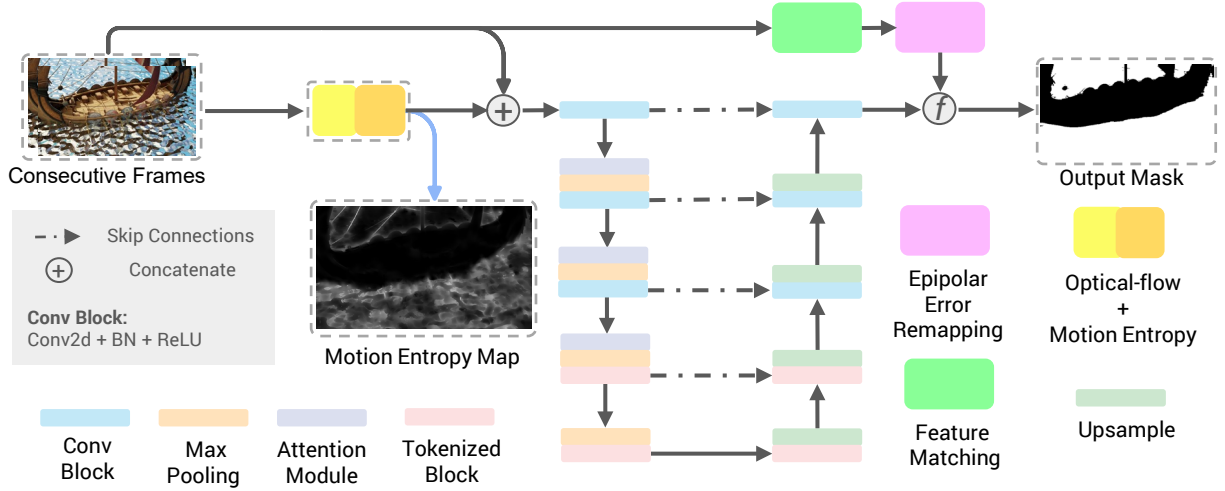


Figure 4.3: Full learning pipeline of *MARVIS*: initially, two consecutive grayscale video frames (at time $t - \delta t$ and t) are fed into optical flow estimation module [1]. This motion information is forwarded to the proposed Local Motion Entropy (LME) layer for LME feature extraction. The fused spatio-temporal feature maps are then passed to a five-stage encoder to obtain features in high-dimensional latent space. Subsequently, the five-stage decoder progressively upsamples by fusing high-resolution feature maps from the corresponding stage encoder through skip connections to generate the segmentation mask with the original resolution. The learning objective involving pixel-wise losses (EGC loss) is given by Eq. 4.3

For maritime object detection, *AquaSim* enables training neural networks to detect diverse shapes, colors, and oceanic variables like water texture, water color variations, different turbidity levels, and lighting conditions. Traditional datasets lack robustness due to limited underwater

views, making *AquaSim*'s provision of precise ground-truth image masks valuable for network training.

4.2.1 *AquaSim* Simulator

Water Modeling: *AquaSim* utilizes BlenderTM's CYCLES rendering engine, employing path-tracing techniques for realistic oceanic environment rendering. Customizable features facilitate the creation of varied underwater scenes with different water properties, including color, texture, and turbidity levels. Additionally, *AquaSim* offers real-time simulation capabilities via the EEVEE rendering engine, balancing rendering quality with interactivity.

A significant advantage of *AquaSim* is its automated data generation that requires minimal user interaction. With a local installation of Blender, you can open an underwater scene equipped with pre-configured lighting and camera setups.

AquaSim offers utilities for custom and prebuilt implementations, allowing users to fine-tune underwater environments. For example, users can adjust the water color using specific RGB value ranges, resulting in corresponding image renderings and masks. In addition, varying levels of visibility is achieved by turbidity adjustment. Water texture is a key parameter in *AquaSim*, offering customization options such as detail, dimension, scale, metallic properties, lacunarity, and strength, which influence the water surface appearance when viewed from below. Furthermore, *AquaSim* incorporates white and Gaussian noise for the characteristics of the water. Users can also adjust underwater lighting conditions by specifying varying levels of sunlight penetration. Lastly, *AquaSim* simulates various wave characteristics, such as height, tilt, sharpness, and textures, to further enhance the realism of the underwater scene.

Sensor Modeling: In addition to generating synthetic underwater environments, *AquaSim*

Table 4.1: Comparison of the different marine robotics simulators showing *AquaSim* excels in rendering near-water surface

Simulator	Quality	Rendering	Surface	GT-Mask
UUV [27]	Low	Gazebo	×	×
URSim [146]	Moderate	Unity3D	×	×
UWRS [147]	Moderate	Unity3D	×	×
HoloOcean [30]	Moderate	UE4	×	×
DAVE [36]	Low	Gazebo	×	×
MARUS [32]	Moderate	Unity3D	✓	×
UNav-Sim [33]	High	UE5	×	×
<i>AquaSim</i> (Ours)	Highest	Blender	✓	✓

*Only Blender and UE5 rendering engine supports ray-tracing with OptiX denoiser for photorealistic images;
Quality: Rendering quality; Rendering: Rendering Engine used; Surface: Realistic Water Surface Simulation;
GT-Mask: Ground truth real-virtual mask .

offers the ability to generate data for the Inertial Measurement Unit (IMU) and information on the location of objects. This allows users to simulate object motions and accurately capture their trajectories using the following model:

$$M_{sim} = M_{GT} + M_{bias} + M_{noise} \quad (4.1)$$

The simulated IMU measurement (M_{sim}) is composed of the ground truth IMU measurement (M_{GT}), the bias (M_{bias}), and the measurement noise (M_{noise}). The latter includes factors

such as angle and velocity random walks and instability correlation bias. We have defined three error models that represent varying noise levels depending on the accuracy of the selected sensor. Lastly, we consider the vibration part of the noise, which can be characterized as white Gaussian, sinusoidal, or specified by its power spectral density.

Furthermore, *AquaSim* automatically generates intrinsic calibration for both stereo cameras, ensuring an accurate representation of the simulated scene. This calibration process is essential for maintaining consistency between the virtual camera’s parameters and real world camera characteristics, enhancing the realism of the generated images, and facilitating accurate data collection and analysis.

4.2.2 Dataset Generation

Having described the *AquaSim* capabilities, we next delve into the dataset creation process utilizing the enumerated objects within simulated scenes. In this phase, *AquaSim* systematically generates synthetic datasets comprising 60 to 120 image pairs for each scene. These feature a variety of objects: kayaks, canoes, rowboats, paddleboards, rafts, surfboards, jet skis, sailboats, fishing boats, beach balls, docks, buoys, water skiers, seaplanes, divers, lifebuoys, water bikes, windsurfers, floating platforms, floating restaurants, speedboats, submarines, yachts, pedal boats, anchors, lighthouses, crab traps, oyster beds, coral reefs, seagulls, and the likes. The diverse array of objects embedded within a scene and the generation of multiple images per scene demonstrate *AquaSim*’s versatility in producing diverse datasets that encapsulate a wide range of maritime environments. This rich dataset composition facilitates robust training and evaluation of object detection algorithms, thus enhancing accuracy in classification and recognition of objects commonly encountered in Aquatic environments. Furthermore, *AquaSim* can also contribute high-

quality datasets to advance research and development in the detection of maritime objects and other similar domains.

4.3 Real and Virtual Image area Segmentation Pipeline

Our primary goal is to design a paradigm with robust and efficient performance in the real world, particularly with respect to camera and object motion. *MARVIS* aims to perform in real-time with limited computing resources on the most widely used unmanned autonomous platforms.

4.3.1 Motion Entropy Kernel

Virtual image areas generated by reflection and refraction often have static spatial features that are difficult to distinguish from the real image area. Therefore, spatial information alone does not provide sufficient visual cues to eliminate ambiguity in complex scenes. Moreover, air-water interfaces with highly variable dynamics and turbulence lead to virtual images with chaotic motions. In contrast, real objects in scenes within the same medium as the camera tend to exhibit relatively smooth motions in a sequence of consecutive frames. This suggests introducing a Local Motion Entropy (LME) measure as a key effective temporal feature for real-virtual image segmentation. This paper proposes and considers it an essential and efficient temporal feature for real-virtual image segmentation. We propose LME as the following function:

$$\begin{aligned}
 H(\mathbf{M}, \mathbf{A}) = & -\alpha \cdot \sum_{m \in \mathcal{M}} p(m) \log_2 p(m) \\
 & -\beta \cdot \sum_{a \in \mathcal{A}} p(a) \log_2 p(a),
 \end{aligned} \tag{4.2}$$

where $\mathbf{M}$ and $\mathbf{A}$ are, respectively, the patch of motion vectors' magnitudes and angles within the local receptive field, m denotes the possible values of the normalized motion vector's

magnitude, $\mathcal{M} \rightarrow [0, 1]$, and a represents the possible values of the normalized motion vector angle, $\mathcal{A} \rightarrow [0, 1]$. The probabilities $p(m)$ and $p(a)$ are calculated from the normalized histogram counts of the variables m and a separately, α and β are the weights of the magnitude and angle of the motion vector in the calculation of local motion entropy, respectively (we set them both to 0.5 here).

Furthermore, we formulate the calculation of local motion entropy as a fixed parameter layer in the encoder of our *MARVIS* pipeline and design a motion entropy kernel with an adjustable receptive field according to Eq. 4.2. Using the sliding-window algorithm, the motion entropy map of the entire image is calculated for different receptive field sizes. LME captures the relative motion entropy between consecutive frames, ensuring consistency despite camera or object motion. This property allows *MARVIS* to be used flexibly in mobile robots rather than just fixed cameras. In addition, due to the property that local motion entropy is independent of pixel information, it is highly robust and consistent in different domains (e.g., underwater and terrestrial domains or synthetic and real world domains). This allows it to provide powerful and stable guidance for network training and improve the model’s zero-shot inference in unseen domains, as demonstrated in Section 4.4.

4.3.2 Weak Supervision of Epipolar Geometric Consistency

The propagating light refracts at the interface of two media with density difference, e.g., air and water. As a result, the refracted rays form a virtual image that no longer satisfies the epipolar geometry under different camera viewpoints. Based on this fact, we impose the epipolar geometric consistency error of sparse keypoints as weak supervision to improve the prediction accuracy of the pipeline, when stereo image pairs are available. In addition to the sparsity of

keypoint matches, the number of keypoints in different images is generally inconsistent. To build a suitable epipolar error map that serves as weak supervision of the network, we remap the normalized epipolar error values of keypoints back to the coordinates of the target image and set the pixel values of other non-keypoint pixels to zero. This ensures that the network can read the error map efficiently and operate in parallel. Accordingly, we define the Epipolar Geometric Consistency (EGC) loss based on the violation of epipolar geometric constraint in the refractive virtual image area:

$$\mathcal{L}_{EGC} = \frac{\sum [-(\hat{y} - 1) \cdot \mathbf{E}_{EGC}]}{\text{Count}_{-(\hat{y}-1) \cdot \mathbf{E}_{EGC} \neq 0}}, \quad (4.3)$$

where $\hat{y}$ is the predicted binary mask, and $\mathbf{E}_{EGC}$ represents the normalized epipolar error map.

4.3.3 Network Architecture: *MARVIS* Model

The detailed network architecture is shown in Fig. 4.3. First, as input to the pipeline, two grayscale images from consecutive video frames (times $t - \delta t$ and t) are passed to an optical flow estimation model, using RAFT here, to obtain motion information. The proposed motion entropy (ME) layer is applied to the optical flow to compute a local motion entropy map. The result and the input grayscale image at the current time t are concatenated and sent to the five-stage encoder of the *MARVIS* model to extract features. The encoder contains three convolutional stages and two multilayer perceptron (MLP) stages. In the first convolution stage, the input that combines spatio-temporal information is passed through a convolution block with fewer filters for initial feature extraction. The resulting feature vector is fed into the two subsequent convolutional blocks with attention to further feature extraction. In the bottleneck, we use the tokenized MLP block [148] to model a good representation in the latent space. The tokenized MLP blocks project the convolutional features into an abstract token with fewer dimensions and

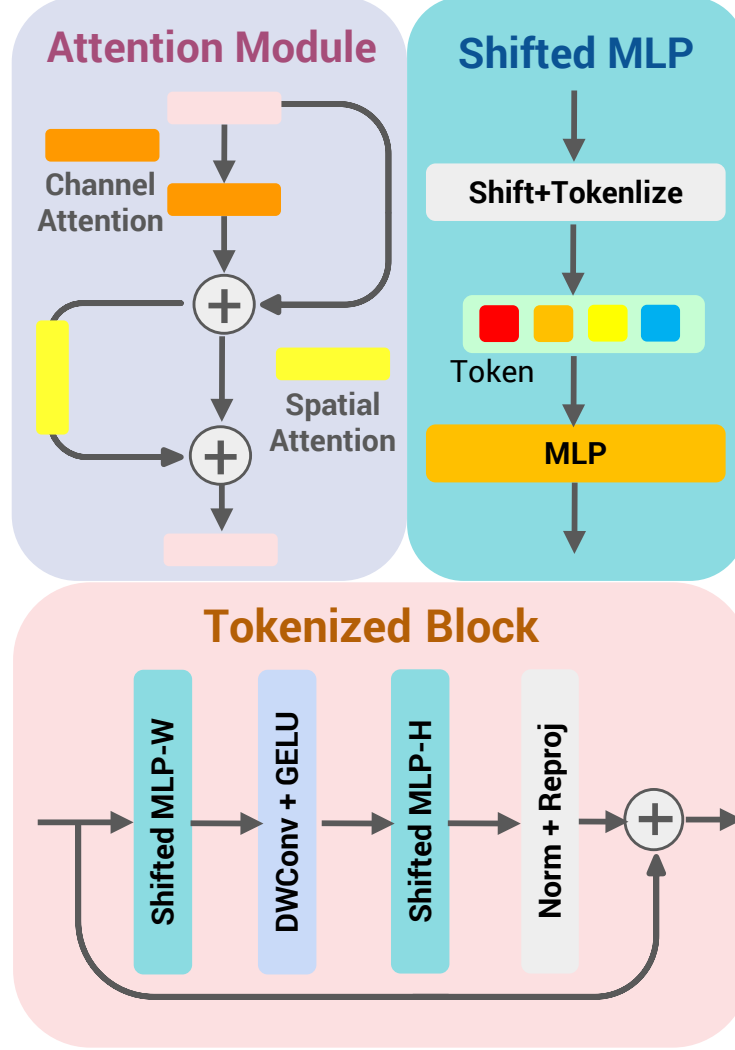


Figure 4.4: The detailed operations in the Attention Module and the Tokenized Block (also described in Eq. 4.4).

then utilize computationally lightweight MLPs to learn the feature and motion encodings for segmentation. The tokenized MLP block given in Fig. 4.4 performs the following operations:

$$\begin{aligned} \mathbf{Y} &= f(DWConv(MLP(Tokenize(Shift_W(\mathbf{X})))))) \\ \mathbf{Y} &= f(LN(\mathbf{T} + MLP(GELU(Tokenize(Shift_H(\mathbf{Y})))))), \end{aligned} \quad (4.4)$$

where $\mathbf{T}$ represents the tokens, H and W denote height and width, $DWConv$ is depth-wise convolution, $GELU$ [149] is an activation function and LN is layer normalization.

Furthermore, lightweight Convolutional Block Attention Modules (CBAM) [150] are inte-

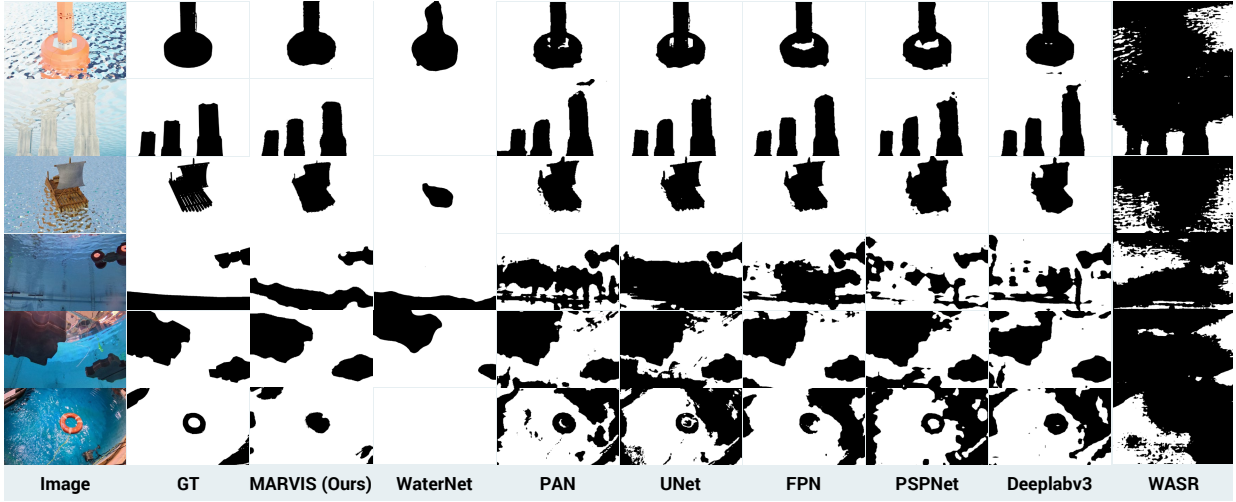


Figure 4.5: A few qualitative comparisons are shown for real-virtual image segmentation by *MARVIS* and SOTA water segmentation pipelines and widely used image segmentation models on both synthetic and real world testset. As seen, *MARVIS* infers accurate and consistent segmentations across the samples from different domains.

grated into stages 2 to 4 of the encoder, enhancing feature representation via channel and spatial attention. The *MARVIS* model’s decoder includes two tokenized MLP blocks and 3 convolutional blocks. Each encoder stage downsamples feature resolution by 2, while the decoder upsamples correspondingly. A skip connection links the encoder and decoder. Leveraging local motion entropy, *MARVIS* efficiently learns latent representations, achieving strong segmentation performance with minimal parameters and computational complexity.

4.3.4 Objective Function Formulation

The end-to-end training is driven by the binary cross-entropy loss [151], the Dice loss [152] and the proposed EGC (epipolar geometric consistency) loss in Eq. 4.3 (when stereo image pairs are available). The cross-entropy loss quantifies the dissimilarity in pixel intensity distributions between the predicted segmentation mask ($\hat{y}$) and its ground truth (y). For a total of n_p pixels, it

is calculated as:

$$\mathcal{L}_{BCE} = \frac{1}{n_p} \sum_i [-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)]. \quad (4.5)$$

To avoid class imbalance, we add the Dice loss, which balances real and virtual image classes by normalizing y and $\hat{y}$ as follows:

$$\mathcal{L}_{Dice} = 1 - \frac{2 \sum_i y_i \hat{y}_i}{\sum_i y_i^2 + \sum_i \hat{y}_i^2} \quad (4.6)$$

Finally, the end-to-end learning objective is formulated as:

$$\mathcal{L}_{MARVIS} = \lambda_B \mathcal{L}_{BCE} + \lambda_D \mathcal{L}_{Dice} + \lambda_E \mathcal{L}_{EGC}. \quad (4.7)$$

The optimal values $\lambda_B = 0.8$, $\lambda_D = 0.1$, and $\lambda_E = 0.1$. in Eq. 4.7 have been determined empirically through hyper-parameter tuning.

4.4 Experiments and Results

Next, we describe the dataset and analyze segmentation experiments, where the models are trained on synthetic data and tested with real data and synthetic data.

4.4.1 Dataset and Implementation Details

Synthetic Data: Utilizing *AquaSim* (described in Section 4.2), we created a diverse synthetic dataset focusing on water turbidity, spacing of water waves, and air-water interface reflection. With varied lighting conditions, our dataset comprises 3012 pairs of 960×540 images with corresponding real-virtual masks.

Real Data: We conducted numerous missions at the National Robotics and Fisheries Research Institute (NBRF) at the University of Maryland (UMD). Two GoPro cameras on an ASV beneath the water surface were used to capture stereo images, and we labeled each image using the appro-

priate annotation tools. The dataset comprises 450 pairs of 960×540 images with corresponding real-virtual masks.

Training Setups: *MARVIS* is supervised on our synthetic dataset created with the AquaSim simulator. AquaSim automatically generates the ground truth of real-virtual image masks under different camera viewpoints. The training, validation, and test sets were split in a 80:5:15 ratio and cross-validated during training. We used Pytorch libraries [153] to implement the learning pipeline; The receptive field of the LME kernel is set to 7×7 . AdamW [154] is the optimizer with an initial learning rate of 0.001 and exponential decay adjustment with a multiplicative factor of 0.9. The threshold for the ratio test for feature matching is set to 0.7. Our *MARVIS* pipeline has about $2.56M$ parameters: $1.0M$ for the RAFT-S [1] optical flow estimator and $1.56M$ for the encoder and the decoder.

4.4.2 Qualitative Evaluation

To our knowledge, no other pipeline has been designed to perform the same real-virtual image segmentation task. For qualitative performance comparison, WaterNet [62] and WASR [56] are the most similar and may be considered as two SOTA water segmentation pipelines. As indicated in Fig. 4.5, WASR’s performance suffers as it addresses a different task. WaterNet typically excels initially, leveraging the initial ground-truth masks provided. However, it often struggles to sustain tracking of virtual image areas over time. Although WaterNet performs well for symmetric objects like the buoy (first row in Fig. 4.5), it may lose track of non-symmetric moving objects such as the life buoy and toy car (fourth and sixth rows in Fig. 4.5). All masks generated by WaterNet are inference images obtained after processing 50 or more consecutive frames.

Since our pipeline performs binary image segmentation, five other widely used SOTA image segmentation models are also relevant: Deeplabv3 [155], PAN [156], UNet [157], FPN [89] and PSPNet [158]. We trained these models by following their recommended settings on the same pipeline and train-validation data splits as *MARVIS*. The qualitative performances of all these models for some samples are illustrated in Fig. 4.5, leading to two main findings from the qualitative experimental analysis.

First, training only on spatial pixel information is not effective for the specific task of real-virtual image segmentation, as we may conclude from Fig. 4.5. Here, the trained network mispredicts many reflections and refractions as real images when composed of similar colors and textures that are inseparable in the RGB space. Moreover, WaterNet [62] also employs motion information by frame differencing, which is no longer meaningful where both the camera and object(s) move. This leads to poor predictions in most samples captured by a moving camera. In contrast, *MARVIS* provides higher-quality segmentation results by leveraging robust and consistent local motion entropy information in test scenarios and the violation of EGC by virtual images.

Second, all pipelines except *MARVIS* give significantly worse segmentation results in real-world images than synthetic images. However, the performance of *MARVIS* remains relatively consistent in both domains. This is attributed to training with only grayscale images and robust guidance from the LME layer for real-virtual image segmentation. This design reduces the domain gap between domains in the feature space, greatly improving the generalization to unseen domains.

4.4.3 Quantitative Evaluation

We use two standard metrics for quantitative assessments: Intersection over Union (IoU) [159] and F_1 -Score [160]. The IoU measures the performance of real-virtual image segmentation using the area of overlapping regions of the predicted and ground truth labels, defined as $IoU = \frac{\text{Area of overlap}}{\text{Area of union}}$. On the other hand, F_1 -Score represents the balance of models that achieve high precision and high recall, which is defined as $F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$. The quantitative results are listed in Table 4.2.

The qualitative comparison and quantitative results are consistent and corroborate our analysis. For example, the real-world domain has scenes and lighting conditions that are different and more complex than those of the synthetic domain. According to Table 4.2, all pipelines degrade in performance by varying degrees in samples from the unseen real world. However, fully trained in the synthetic domain, the *MARVIS* learning pipeline still achieves up to 78% in IoU and 86% in F_1 -Score. This is a performance degradation of only 15.52% (from 94%) in IoU and 9.88% (from 96%) in F_1 -Score, compared to 20.34% – 40.8% in IoU and 13.75% – 28.11% in F_1 -Score by other pipelines

This confirms that the strategy to add the temporal information by the LME and EGC supervision (while ignoring color information) indeed enhances the zero-shot generalization ability of *MARVIS* on the real-virtual image segmentation task. Furthermore, *MARVIS* leverages the strong prior knowledge of LME and EGC designed for real-virtual image segmentation during the training process, enabling the lightweight network to learn a good feature representation for rapid convergence. *MARVIS* has only 2.56M parameters that can rapidly generate segmentation masks at inference rates of 43.48 FPS (8.06 FPS) on a single NVIDIATM RTX 4070 GPU

Model	Params ↓	IoU ↑		F1 ↑	
		Real	Syn	Real	Syn
WASR [56]	71M	13.24	29.10	21.78	44.37
WaterNet [62]	22M	41.08	53.86	49.63	59.45
PSPNet [158]	11.32M	61.35	87.48	74.63	90.51
Deeplabv3 [155]	5.63M	68.68	89.02	79.26	93.01
PAN [156]	4.10M	61.65	89.69	74.45	93.56
UNet [157]	31.04M	51.11	91.91	66.85	94.96
FPN [89]	13.05M	66.84	91.91	78.65	94.94
MARVIS(Ours)	2.56M	78.56	94.08	86.47	96.35

Table 4.2: Quantitative comparison for real-virtual image segmentation performance by *MARVIS* and other SOTA water segmentation pipelines and widely used image segmentation models on both synthetic and real-world test sets.

(IntelTM Core i9-4.10GHz CPU) core. Compared to other pipelines, it achieves much lower computational overhead and faster inference speeds, making it feasible to deploy on computationally constrained drones and AUVs.

Chapter 5: Other Applications

5.1 Olive Detection

5.1.1 Introductions

The olive tree, native to the Mediterranean region, has played an essential role in the advancement of humanity for millennia. The olive harvest is a lengthy process requiring almost a year-long preparation and is essential for the global economy, with a market valued at \$13.77B and is forecasted to reach \$17.99B by 2029 [161].

Yield estimation aims to forecast the output of crops. Manual inspections help farmers better understand their crops, but patrolling a grove with thousands of trees is time-consuming. One solution is to utilize Unmanned Aerial Vehicles (UAVs) to capture and analyze aerial image data to make better decisions.

Object detection via robotics has been employed for multiple types of crops [162–166]. However, olives present additional problems not encountered with other crops. Specifically, one must account for moderate variation in olive color, since the olive may have multiple shades of green. Another problem is the similarity between the olive fruit and the surrounding leaves, making it challenging to differentiate between the two. This similarity can be exacerbated even further when only the end of an olive is visible, resembling a leaf. In addition, the dense canopy of an olive tree occludes many of the olives, making them partially visible and difficult to recognize as shown in Fig. 5.1. Finally, the average size of an olive is $2\text{cm} \times 1\text{cm} \times 1\text{cm}$, magnifying all

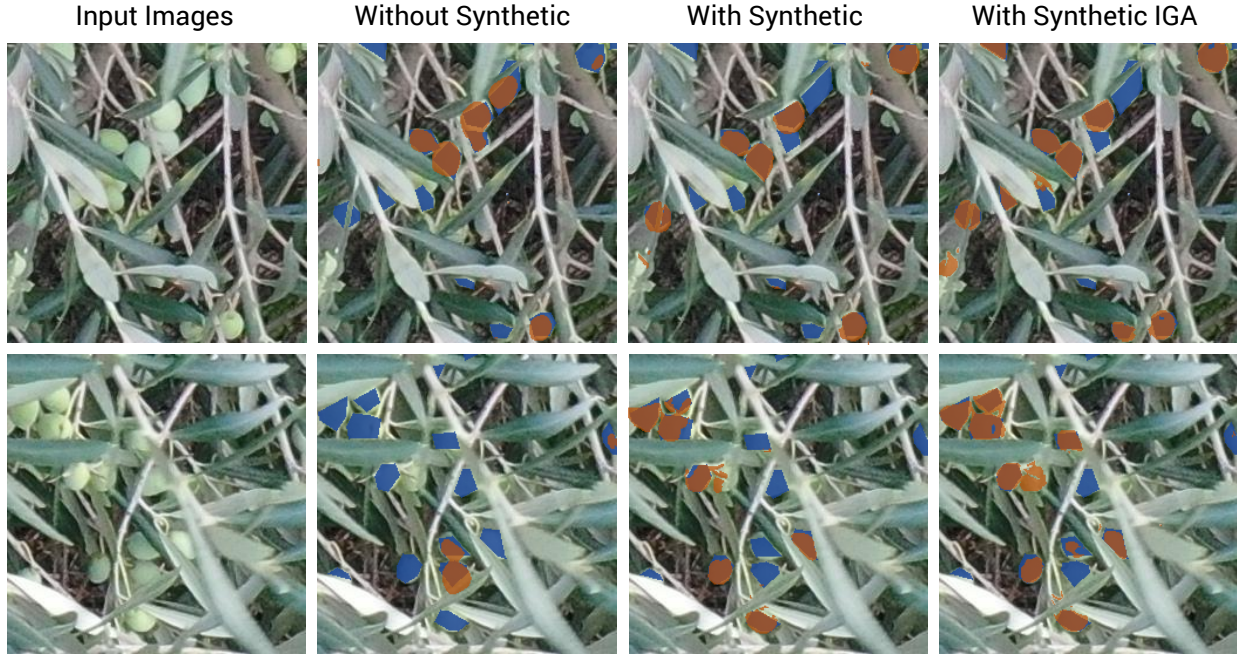


Figure 5.1: Each column left to right: Input image, pblueiction using only real data, pblueiction using real and synthetic data, pblueiction using real and synthetic data in IGA. Pblueictions are shown in **orange** and ground truths are shown in **blue**. Adding synthetic data to the training set increases the number of correct pblueictions.

the difficulties mentioned above.

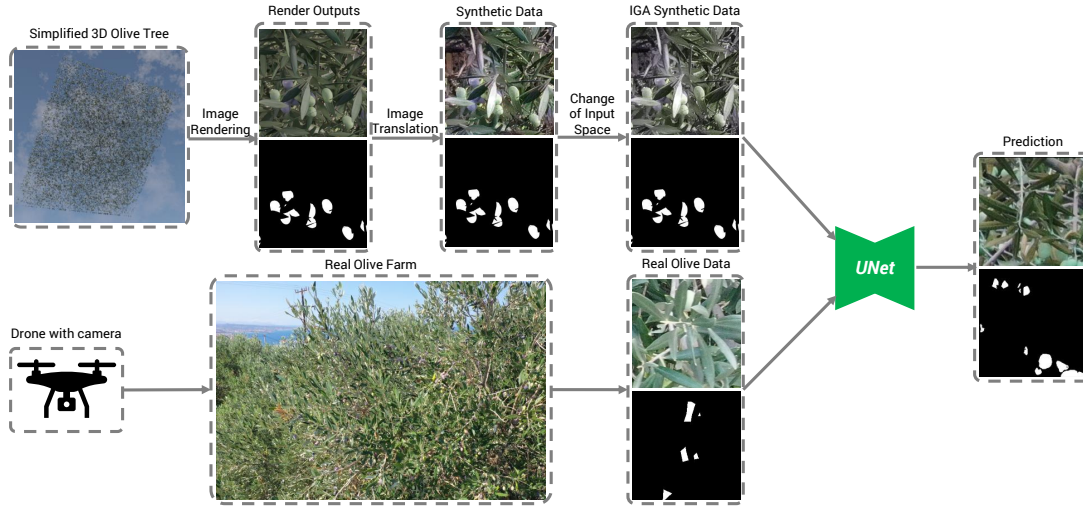


Figure 5.2: An overview of our approach: The proposed geometric model is used to generate synthetic images which are further fed into a VSAIT to enable image-to-image translation. We then combine the synthetic data with real data to train an Unet for olive detection.

However, labeling images collected from UAVs is an exhausting task, especially when

there could be up to 17,500 olives [167] in a single olive tree (up to 1000 olives in a single image from our observation). An olive grove may contain thousands of olive trees, making this very labor-intensive. *Therefore, we investigate a new approach to cover the deficiency of labeled data, namely we create and incorporate synthetic data as an aid to training models for olive segmentation.* Synthetic data is automatically labeled and can be easily scaled up to higher orders of magnitude. We create synthetically generated image-mask pairs of an olive tree, including its olives, leaves, branches, and surrounding background. We propose the use of our synthetic data to create a high-performing, automated system for detecting olives in olive groves. The contributions of this paper are as follows:

- We create the world’s first olive detection dataset consisting of mostly synthetic and some real images of olive trees
 - We generate a photorealistic 3D model of an olive tree to create an auto-labeled synthetic dataset
 - We propose and simplify a geometric model for olives on an olive tree for lightweight rendering purposes
- We create a novel color input space for our synthetic data, making it more generalizable to real-world data
- We experiment with segmentation models to assist in situations where the amount of real labeled data is very limited

5.1.2 Synthetic Olive Tree

Given the arduous nature of real-world data collection and labeling for semantic segmentation, we propose a system to rapidly streamline the process of creating a rich, diverse hybrid dataset containing real and synthetic data. The need for such a swift system is highlighted further, given that each olive tree contains around 2500 to 17500 olives, on average [167]. The resulting generated synthetic data is then combined with real data to train a segmentation model to detect olives as shown in Fig. 5.2. In Sec. 5.1.2.1, we illustrate the process of creating synthetic data by modeling the geometry of an olive tree utilizing Blender™, a 3D open-source graphics software to perform rendering [26]. We also ensure that our 3D model is light enough to be rendered under constrained computational resources (such as a modern-day laptop), further democratizing our synthetic data pipeline. In Sec. 5.1.2.2, we explain how modern image-to-image translation methods may be applied to make the rendered synthetic images appear photorealistic.

5.1.2.1 Olive Tree 3D Model

- Synthetic Olive To generate synthetic data for an olive tree, we first need to create its most fundamental component: the olive. We model the geometry of an olive, by using an ellipsoid mesh as a baseline:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \quad (5.1)$$

Here in Eq. 5.1, $a = 1.2$ cm and $b = 1.2$ cm are the width factor of the ellipsoid while $c = 2.3$ cm is the length and width of the olive as shown in Fig. 5.3a. We added some randomness in the scaling factor to make it more realistic. Then, the vertices near the end

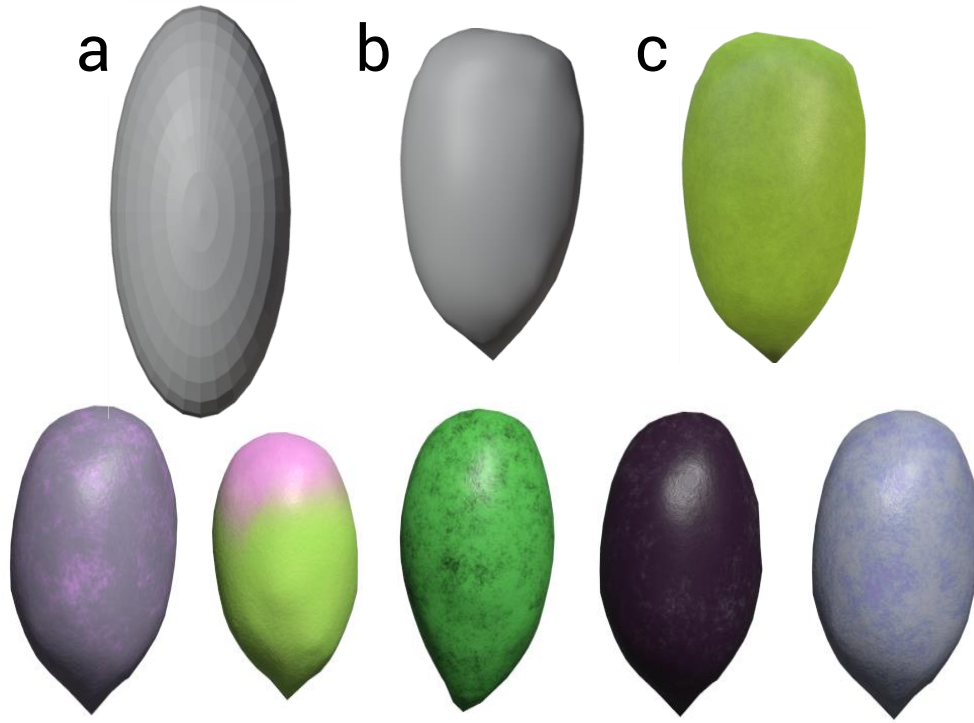


Figure 5.3: (a) the 3D modeling of our ellipsoid. (b) the 3D modeling of an olive (c) an example olive rendebblue with texture.

of the ellipsoid are extended to create a sharp point, matching the shape of a real olive.

Subsequently, a smoothing process is applied to the olive 3D model to eliminate any form of rigidness. The final dimensions of our 3D olive model are $2.5\text{cm} \times 1.4\text{cm} \times 1.4\text{cm}$ respectively as such are the approximate dimensions of olives when ready for harvest.

Fig. 5.3b demonstrates the process of modeling our proposed olive models. We then initiate the process of adding texture to the olive model. The appearance of an olive is impacted by the lighting, shading, and albedo of the environment. All olives begin as green fruits and gradually transform into darker shades, ending in an almost black, dark purple color as the harvest season progresses (see Fig. 5.3c and its examples underneath). Occasionally, a single olive may have two different colors simultaneously. In addition, olives usually do not have a smooth texture but may exhibit bumps and wrinkles on their surface. Given the

grove environment, it is possible for olives to accumulate dust, providing a lighter shade on certain sections of the surface. Finally, many olive groves suffer from the insect *dacus olea*, which punctures the olive causing black spot(s) to appear on its surface. We consider all the above variations when designing the texture of an olive. We model a set of eleven unique textures to be applied onto the 3D olive model, each varying in noise, roughness, distortion, bump, and color mixing. This ensures that the set of synthetic olives closely resembles real-world olives.

- **Olive tree and leaves** Next, we create the leaves and the branches of the olive tree. We scan a pair of leaves from a real olive tree to model the 3D geometry. The scanner automatically provides the texture of an olive leaf. To model the branches and subbranches, we utilize the bezier curve as a baseline, and is modeled as follows:

$$B(t) = \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i P_i \quad (5.2)$$

where t represents the position of the point on the curve and P_i represents the i^{th} control point. In Eq. 5.2 we select $n = 5$ control points per branch. We then prune the ends of the curves in Blender to make them more realistic. To match the texture of an olive tree's branches onto our 3D model's branches, we use a dark brown as our base color and apply dark noise to the branches. This process is illustrated in Fig. 5.4.

In an effort to make our 3D model more accessible to those without high-end computing devices, we decide to make a lighter version of an olive tree model without a trunk or a dense canopy. *Choosing to do so drastically blueuces the average time needed for rendering synthetic image-mask pairs.* First, we eliminate the trunk. Then, we blueuce the

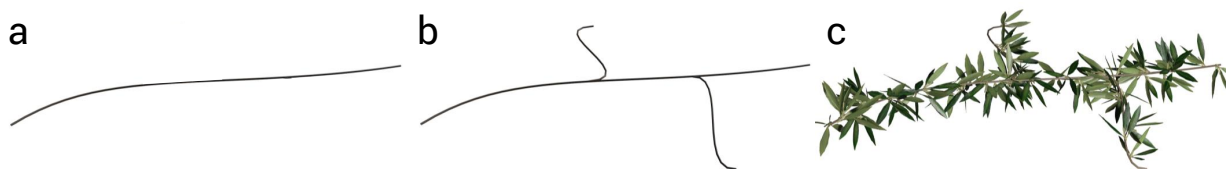


Figure 5.4: Fig. 5. (a) a single bezier curve with 2 control points. (b) the bezier curve we use as a branch. (c) the bezier curve with 3D-scanned leaves scattered onto it.

number of leaves and branches required by using an invisible 2D plane as a baseline for their scattering. Finally, to address the excessively high number of vertices on the 3D-scanned leaf, we apply a limited dissolve onto the leaf model for simplification, further democratizing (making it easily accessible for all) our 3D model.

To complete the model, we first create four separate layers of scattered leaves, two separate layers of scattered branches, and one layer of scattered olives per rendering session in the formation of 2D planes. The randomness of leaf orientation is set to no more than 9° and the size randomness is set to be no more than 10% of the original dimensions. When scattering the olives onto the leaves and branches, we place them behind one leaf layer in order to simulate the obscurity that real olive leaves provide to the olive fruit. We apply orientation randomness within 45° and size randomness within 5% of the original dimensions to all olives. In total, we scatter approximately 2,600 olives onto the leaves per rendering session. In total, we run 16 rendering sessions to produce 15,960 image-mask pairs. Example renderings are shown in Fig. 5.5c. Since the Mediterranean olive harvest occurs during the months of mid-Autumn, we emulate appropriate background and lighting conditions for our 3D model. We first simulate dry, sunny days with little to no clouds using Blender's sky, Musgrave, and gradient texture nodes. We then simulate a cloudy, overcast environment using Blender's dynamic sky. Finally, we place a plane behind the final leaf layer

with ground texture projected onto it via UV mapping to simulate a UAV flying above a canopy with the ground as the background. We also UV-map an image of scatteblue leaves onto a background plane to simulate a UAV flying at the height of the canopy.

- **Rendering** At this point, the 3D olive tree model is ready to be rendered into image-mask pairs, a very computationally demanding task. Therefore, we take the following steps to shorten the rendering process, expanding the dataset’s availability to computers with lower processing power (such as modern laptops). We utilize Intel OpenImageDenoise [168], an open-source library of denoising filters made for images rendered with ray tracing. We lower the maximum number of samples in Cycles render (number of paths to trace for each pixel in the final render) to 50, still producing a high-quality output. We eliminate all subsurface scattering in all objects

5.1.2.2 Image-to-Image Translation

Image-to-image translation is a technique that converts an image from a source domain to a target domain (e.g. convert an image of a horse to a zebra) [169] [170]. Typically, synthetic data does not possess the realistic features of the real data domain. Models trained on untranslated synthetic data perform poorly on real-world testing data. Therefore, we conduct image-to-image translation to transform synthetically rendered images in Blender of our 3D model to the real-world olive tree domain. The objective is that training with synthetic data closely resembling the real world will result in higher accuracy models. Specifically, we employ Unpaired Image Translation via Vector Symbolic Architectures (VSAIT) [171] to translate the Blender-generated images to the real-world domain. We also choose VSAIT in order to reduce occurrences of semantic flipping (when a green olive gets translated into a leaf, corrupting the translated data)

given that VSA-based methods are capable of learning high-level, abstract concepts [172].

Formally, VSAIT is trained to learn the translation between the Blender-generated synthetic olive tree images X to the real olive grove images Y . The three major components of this network are the Generator G , the Discriminator D_Y , and the Source $\leftrightarrow$ Target Mapper F . The overall loss consists of the sum of the Hypervector Adversarial Loss and the VSA-based Cyclic Loss [171].

Hypervector Adversarial Loss: This ensures the similarity between the synthetic translated image hypervectors $v_{G(X)}$ and the real target hypervectors v_Y . This is done by applying the VSA binding operation on hypervector mapping $F(v_x)$ and source vectors v_X to yield a hypervector of synthetic features mapped to the real domain. The Hypervector Adversarial Loss is calculated as:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_Y(y)} [\log D_Y(v_y)] \\ & + \mathbb{E}_{x \sim p_X(x)} [\log(1 - D_Y(v_{G(x)}))] \\ & + \mathbb{E}_{x \sim p_X(x)} [\log(1 - D_Y(v_x \otimes F(v_x)))]\end{aligned}\quad (5.3)$$

VSA-based Cyclic Loss: This cyclic loss minimizes occurrences of semantic flipping by constraining G so that similar hypervectors may be returned when mapping translated vectors back to the synthetic domain from the real domain. Meaning, $v_X \approx v_G(X) \rightarrow X$. The VSA-based Cyclic Loss is calculated as:

$$\mathcal{L}_{VSA}(G, X) = \mathbb{E}_{x \sim p_X(x)} \left[\frac{1}{n} \sum_{i=1}^n \text{dist}(v_x^i, v_{G(x) \rightarrow x}^i) \right] \quad (5.4)$$

where the *dist* is the cosine distance between the source hypervectors and inverted translated hypervectors.

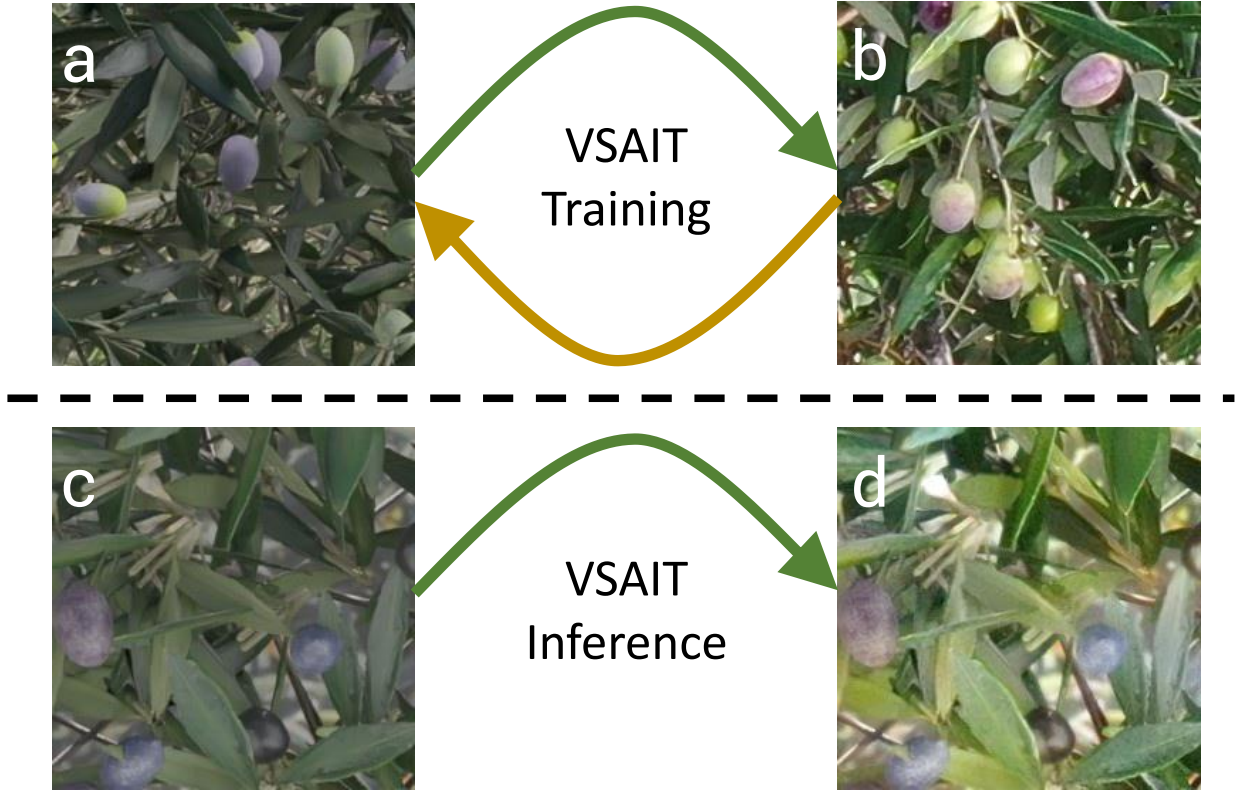


Figure 5.5: Image-to-image translation (a) A single sample of the simulated olive, (b) A single sample of the real olive, (c) Simulated olive image, (d) Synthetic olive image translated into real world for photorealism.

Therefore, the overall loss is as follows:

$$\mathcal{L}(G, D_Y, X, Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{VSA}(G, X) \quad (5.5)$$

To create X , the source domain, we render 12,768 images of our 3D olive tree model for training and 3,192 images as validation. To create Y , the target domain, we use 12,452 UAV-captubblue images for training and 3,113 for validation. Overall, we train VSAIT for 98 epochs using its default settings (see Fig. 5.5a, 5.5b). Then, we perform inference to apply VSAIT to rendeblue images (Fig. 5.5c), yielding a photorealistic synthetic output (Fig. 5.5d).

We ensure that the images used in the Y domain for image-to-image translation do not overlap with any of the real images that will later be used in testing segmentation models. When

X gets translated to Y , it learns the features of Y . Training a segmentation model using synthetic images translated to the same Y as the real-world test set is just like training with testing data as input (and must be avoided).

5.1.2.3 Synthetic Data Color Input Space

After the synthetic data undergoes image-to-image translation, the olives in the synthetic data still stand out brightly, unlike in real olive groves. Furthermore, the synthetic data translated by VSAIT can easily still be distinguished from images of the real domain, if looked at closely. Therefore, in an attempt to make the olives in the VSAIT output less distinguishable (just like in the real-world), we create a new input space, with the goal of making the olives more concealable. We draw inspiration from [173] who propose a novel input space from $(RGB \rightarrow R, M=\max(G,B), I)$ where I is the overall intensity. However, Yu *et al.* [173] apply this technique to the underwater domain where there is a deficiency of blue reflections to extract a clearer image for depth estimation. We propose a color input space for the opposite purpose: to obscure the olives further into their backgrounds. *This new input space may be applied to all synthetic agricultural domains with the purpose of blending any small fruit into its respective backgrounds* and is as follows (called IGA):

$$RGB \rightarrow (I, G, \text{Avg}(R, B))$$

Converting the VSAIT-translated synthetic data to this input space yields a slight improvement as shown in Table 5.1 and is labeled as $O_{IGA.and.real}$.

5.1.3 Experiments and Results

We first describe our datasets. Then we analyze our segmentation experiments on models trained on images with and without synthetically generated data.

5.1.3.1 Dataset Overview

Synthetic Data: As described in Section 5.1.2, we initiate multiple rendering processes of our Blender scene, each reproducing parts of an olive tree with a blue sky, leaf-only, or ground texture background. In addition, we utilize multiple lighting conditions to augment the variety of our synthetic data. Afterward, we initiate the image-to-image translation process [171]. The result of these two processes is the speedy generation of 15,960 synthetic image-mask pairs.

Real Data: We conducted several UAV flights over the island of Crete, Greece. Our UAV is the DJI Mavic 2 Pro with a Hasselblad L1D-20C camera. To make the dataset as diverse as possible, the flights were staggered to record differences in cloud cover and albedo, and to capture images of olives during different temporal stages of the olive harvest, approximately ranging from one to two months. We also chose to only set our ISO to 100 for daytime capture and 400 for evening capture. We then labeled our UAV-captured images by hand using LabelMe [174]. In total, we have 3,113 real-world image-mask pairs in patches of size $256 \times 256 \times 3$ and $256 \times 256 \times 1$ and we labeled approximately 2,600 olives. The maximum number of olives per unpatched image is around 600.

5.1.3.2 Experiments

We construct two training sets: one with 15,960 synthetic and 100 real image-mask pairs and another with only 100 real image-mask pairs. This was decided to reflect the reality that it

is very difficult to manually create a large enough labeled olive dataset. We select UNet [175] as our model architecture and train with several backbones [176, 177] as shown in Table 5.1. To standardize experiments across our two data sets, we only employ the Adam optimizer [178] with an initial learning rate of 0.001 and use the Jaccard index [179] as our loss function. All models are trained for a maximum of 100 epochs using a batch size of 32. We use minimal real-world data as it is arduous to obtain and label, simulating situations where one may only have limited access to real data, which are more realistic cases.

Our testing set consists of more than 3000 real-world image-mask pairs that are hand-labeled [174]. Our metric of choice is the Intersection over Union, as there is only one class of objects to segment. We opt out of using mean Average-Precision as an evaluation metric given that models trained on mostly synthetic data perform well only with low-confidence score thresholding, as is in our case.

5.1.3.3 Results

Based on our experiments, as shown in Table. 5.1, we observe that the models with identical architectures and backbones trained on both synthetic and real images outperform the baseline models, which are trained without synthetic data. In our *Training Data* column, O_{real} represents the set of just real-world training data, $O_{syn_and_real}$ represents the set of real-world and VSAIT-translated synthetic data, and $O_{IGA_and_real}$ represents the VSAIT-translated synthetic data converted to the IGA input space. Our biggest marked improvement, when compared to the baseline, was observed from 24.22% to 40.22% IoU, corresponding to a percent change of 66% when using ResNet101 [176] as our backbone. We also observe that when using EfficientNetB5 [177], ResNet101, and ResNet152, the highest-performing models are trained on VSAIT-translated im-

Table 5.1: IoU using real only, synthetic/real, and synthetic IGA/real

Backbone	O_{real}	$O_{syn_and_real}$	$O_{IGA_and_real}$
EfficientNetB5	41.41%	53.06%	54.05%
ResNet101	24.22%	36.28%	40.22%
ResNet152	28.99%	40.54%	45.33%

ages converted to the IGA input space. Fig. 5.1 shows the difference in segmentation performance when using models trained with and without synthetic data. When using an EfficientNetB5 backbone, we observe a 12.64% jump in IoU over the baseline when the model is supplemented with synthetic data in the IGA input space. Similarly, when we use Resnet152 as a backbone, we observe an increase of 16.34% compared to the baseline when supplemented with unedited VSAIT-generated data. Detecting olives in an olive grove is a problem in the low-training data regime domain. Under this setting, it is not convenient to label vast amounts of olive tree images and hence synthetic data offers a reasonable alternative. In this paper, we simulate such a situation and have demonstrated that supplementing a low amount of real data with easy-to-generate synthetic images improves the segmentation results by 66% as shown in Table 5.1.

5.2 Automated Inventory Counting for Warehouse Management

5.2.1 Introduction

In modern warehouses, efficient inventory management is critical to maintaining operational flow and reducing costs. Traditionally, counting large quantities of small objects, such as parts or components, is a labor-intensive and time-consuming process that is prone to human er-

ror. These challenges are compounded by the need for accuracy in large-scale operations, where even minor discrepancies can lead to significant logistical and financial issues.

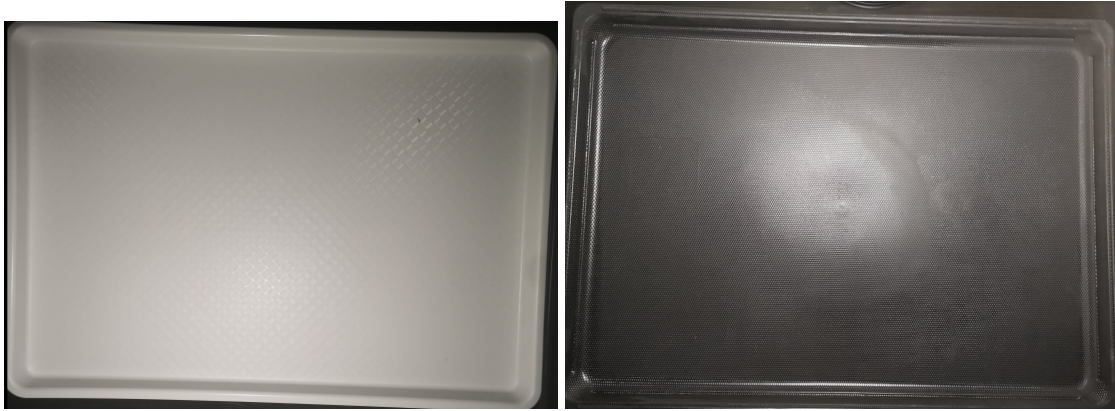
To address these inefficiencies, recent advancements in computer vision and machine learning offer promising solutions for automating the counting process. By applying object detection, segmentation, and machine learning algorithms, we can develop systems capable of accurately counting objects in real-time, regardless of their orientation or arrangement on a flat surface. This automated approach not only reduces human labor and error but also speeds up the inventory process, enabling real-time tracking and improved decision-making in warehouse environments.

This section explores how these technologies, initially developed for oceanic tasks, have been adapted for warehouse management to streamline inventory counting. The proposed system leverages machine learning models trained to recognize and count small objects, even in challenging conditions such as overlapping items or cluttered arrangements. The solution represents a significant step towards the full automation of warehouse inventory management, benefiting industries that rely on fast, accurate, and scalable solutions.

5.2.2 Proposed Approach

In this section, we present our approach to automating the inventory counting process using machine learning and computer vision techniques. The objective is to accurately count small objects placed on flat surfaces with minimal human intervention, addressing challenges such as overlapping objects, varying orientations, and clutter. Our solution leverages synthetic data, object detection, instance segmentation, and image inpainting algorithms to provide accurate real-time counts.

- All training images were synthetically generated using Unity, a powerful simulation en-



(a) Tray 1 setup

(b) Tray 2 setup

Figure 5.6: Trays for the background and the real experiment.

engine that allows for the controlled creation of diverse datasets. We utilized the two tray images (as depicted in Fig. 5.6) as the background in the simulated environment. These same trays were also used in the real-world testing environment. By generating synthetic scenes in Unity, we were able to create images of small objects arranged in random configurations, with varying levels of overlap, orientation, and clutter. Each synthetic image was accompanied by corresponding object masks, significantly reducing the time needed for data preparation, as manual labeling was not required.

Unity’s [180] flexibility allowed us to simulate a variety of warehouse scenarios, ensuring that the machine learning models trained on this data would generalize well to real-world situations. These synthetic datasets represent a wide range of object quantities (5-50 parts per image), and include common challenges encountered in warehouse environments, such as objects touching or occluding one another.

- Our initial object detection and segmentation model was based on MaskRCNN [181], a popular deep learning framework that excels at instance segmentation. MaskRCNN was trained using the Unity-generated synthetic dataset, allowing it to learn how to detect and

delineate individual objects in an image. MaskRCNN's ability to generate precise masks for each object made it an ideal choice for handling situations where objects are partially occluded or overlapping.

The training process focused on optimizing the model's performance for real-world conditions, including variation in object orientation, scale, and position. MaskRCNN performed well in many cases, successfully segmenting objects even in complex scenes. However, it struggled in situations where there were high levels of object overlap or when objects were densely packed, which led us to explore additional solutions to improve detection in these scenarios.

- To further enhance the system's detection capabilities, we introduced SOLO (Segmenting Objects by Locations) [182] . While MaskRCNN provides high-quality instance segmentation, SOLO divides the image into grids and predicts object presence within each grid cell, offering a more structured approach to object detection. This method is particularly effective in cases where MaskRCNN struggles, such as when multiple objects are closely packed or touching.

By combining SOLO with the previously trained MaskRCNN model, we were able to improve the system's ability to detect and count objects in challenging configurations. SOLO's grid-based approach allowed for more consistent detection, even when objects were clustered or partially occluded. This two-step detection process, using both MaskRCNN and SOLO, ensures that the model can accurately handle a wide variety of object arrangements.

- In cases where significant object overlap occurs, we incorporated inpainting algorithms to

fill in the missing parts of occluded objects. This technique allows the system to "reconstruct" objects that are partially hidden from view, ensuring that they are counted accurately even in highly cluttered scenes. The inpainting model was trained to predict the likely shape and size of occluded objects based on the visible portions and surrounding context, enabling more reliable counting in complex environments.

The inpainting process is particularly useful in warehouse settings where objects may frequently overlap due to space constraints. By combining inpainting with object detection, the system can achieve accurate counts even when object occlusion is a significant factor.

5.2.3 Result

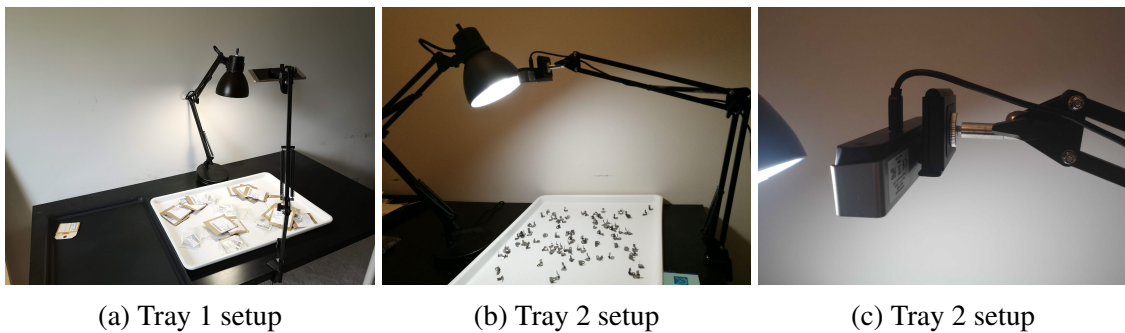


Figure 5.7: Real Experimental setup

- The real-world environment setup consists of a lamp, a downward-facing camera, and a tray as the workspace, similar to the configuration used in the simulated environment. This setup ensures consistency in both data collection and model performance across synthetic and real-world scenarios. The downward-facing camera, as shown in Fig. 5.7, captures images of the objects placed on the tray. These images are then processed by a custom-designed app, which handles the data transfer by uploading the captured images to the

cloud. In the cloud, the detection and counting models are executed, and the precise object count is returned to the app in real-time.

- Once both MaskRCNN and SOLO models were trained on the synthetic Unity datasets, they were integrated into a unified system capable of real-time object counting. The use of pre-labeled synthetic data from Unity not only accelerated the training process but also enabled us to simulate a broad range of warehouse conditions, ensuring the models are robust across different scenarios. The trained models were then deployed on mobile devices, allowing warehouse personnel to capture images of objects and receive accurate counts in real-time.

The system is designed to process images containing up to 50 objects, providing near-instantaneous results with an error rate of less than 3%. Real-time processing is enabled by the model's lightweight architecture, which ensures that the counting process can be performed quickly without sacrificing accuracy.

- To evaluate the system's performance, we used F1 score and accuracy metrics across multiple test images, each containing between 5 and 50 parts. The evaluation used synthetic images representing typical warehouse conditions, ensuring the results reflect real-world applicability. As shown in Table. 5.2, the model achieves an average F1 score of 99% and an accuracy mean of 98.92%. These results demonstrate the system's reliability and effectiveness in accurately counting objects, even in challenging conditions such as overlapping or cluttered arrangements.
- The proposed approach is designed to be highly scalable, making it suitable for a wide range of warehouse environments, regardless of size or inventory type. By using synthetic

data generated in Unity, the system can be adapted to recognize and count different types of objects, depending on the specific needs of the warehouse. This flexibility ensures that the system can be easily integrated into existing inventory management workflows, reducing the need for manual counting and improving overall efficiency.

Additionally, the system can be refined further by incorporating real-world data as warehouse operations evolve. This continuous learning capability allows the models to improve over time, ensuring they remain accurate as warehouse inventories change or new object types are introduced.

Metric	B	C	D	E	F	G	H	I	J	K	L	M	O	P	Q	R	S	AVG
F1 Score (%)	100	100	98.7	98.3	99.6	99.5	97.9	98.2	98.4	98.7	99.1	99.4	99.2	99	97.3	99.9	99.8	99
Accuracy Mean (%)	100	100	98.3	98.3	99.4	99.2	98.2	97.9	98.8	98.4	98.7	99.6	99	99.1	97.1	99.9	99.8	98.92

Table 5.2: F1 Score and Accuracy Mean (using images with 5-50 parts)

Chapter 6: Conclusion and future work

6.1 Conclusion

In this thesis, we explored the intersection of simulation, machine learning, and robotics to advance precision maritime and marine applications. We utilized game engines and synthetic data generation to create benchmark simulations for UAV-based maritime image detection, focusing on objects like the BlueROV [183] and whales [184]. The SeaDroneSim framework demonstrated that synthetic datasets can achieve competitive results in object detection (71 mAP for BlueROV), highlighting the potential for such approaches in data-critical applications where real-world data collection is challenging.

We also introduced simulation-based methods for oyster detection [63, 64], showing that synthetic data can enhance the performance of object detection models in marine environments. By generating realistic oyster models and leveraging image-to-image transformations, we achieved a 35.1% improvement in segmentation IoU over real data alone.

Additionally, we developed a comprehensive system for underwater oyster monitoring using the Aqua2 [64] platform, combining on-edge detection with stable diffusion-based synthetic data. This system improved underwater detection performance and set the foundation for broader applications in marine monitoring.

Furthermore, the MARVIS [185] system for real-virtual image segmentation near water surfaces showed that fundamental features such as optical-flow and geometric constraints can

help improve segmentation results, opening new possibilities for 3D reconstruction and object recognition tasks in maritime applications.

Finally, our UIVNav [65] framework demonstrated the efficacy of using imitation learning and intermediate representations for underwater navigation, improving data collection over oyster and rock reefs without requiring retraining when domain or object changes occur. This approach provides a generalizable navigation solution for marine exploration.

The applications of this pipeline extend beyond marine and aquatic environments. The same approach can be applied to other domains, such as agriculture [186], where automated systems can detect and monitor olives for the estimation of the yield and the assessment of the health. Additionally, this pipeline is well-suited for industrial applications, particularly in counting small groups of mechanical parts in warehouse management, improving efficiency and accuracy in inventory tracking.

6.2 Future Work

Looking ahead, several avenues of research can build on this foundation to further improve the robustness and efficiency of oceanic tasks:

- **Multimodal Sensor Fusion:** A promising direction is to integrate multiple sensor modalities for more accurate perception and navigation in underwater environments. For example, combining forward-looking sonar with RGB cameras or side-scan sonar could significantly enhance object detection, mapping, and obstacle avoidance, especially in environments with poor visibility. Fusing these sensor inputs would allow for a more comprehensive understanding of the surroundings, improving both navigation and detection capabilities in complex underwater scenarios.

- **Advanced Sensor Integration:** In addition to multimodal fusion, further improvements to the Aqua2 platform can be achieved by integrating advanced sensor modalities such as multi-beam and single-beam ultrasound. These sensors would provide better depth estimation and improve performance in murky or cluttered water environments. This enhanced sensor suite would allow the system to operate effectively across a broader range of conditions and improve both navigation and object detection accuracy.
- **Collaborative Multi-Robot Systems:** Expanding the UAV-ROV collaboration to include a fleet of unmanned systems, such as Remotely Operated Vehicles (ROVs) working alongside Autonomous Surface Vehicles (ASVs), could improve the efficiency and scalability of marine surveys. For instance, ROVs could perform detailed underwater exploration while ASVs monitor and coordinate surface activities. This multi-robot approach would enable larger-scale environmental monitoring and more precise geo-location of underwater objects. Swarms of UAVs, ROVs, and ASVs working in tandem could cover broader areas, allowing for faster and more comprehensive data collection.
- **Integration with SLAM and Mapping Techniques:** The current UIVNav system could be further improved by integrating it with SLAM (Simultaneous Localization and Mapping) algorithms or occupancy maps to allow for more sophisticated navigation in unknown environments. SLAM would enable the robot to build a map of its surroundings in real-time, improving localization and enabling it to avoid previously visited areas, reducing redundant coverage. This integration would be especially useful for long-term, autonomous exploration tasks where state estimation is uncertain or unavailable.
- **Vision-Language Models for Enhanced Navigation:** Another exciting direction is incorpo-

rating Vision-Language Models (VLMs) to aid in navigation and exploration tasks. These models could help robots interpret high-level instructions and generate detailed navigation plans in response to complex commands, such as "navigate to the reef and avoid large rocks." VLMs could also improve the robot's ability to understand and classify objects based on visual data and natural language cues, making exploration and data collection more efficient and adaptable to new environments.

Appendix A: Permissions and IEEE Copyright Notice

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of University of Maryland's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to IEEE <https://www.ieee.org/publications/rights/rights-link.html> to learn how to obtain a license from RightsLink.

Bibliography

- [1] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow, 2020.
- [2] Joel E Cohen, Christopher Small, Andrew Mellinger, John Gallup, and Jeffrey Sachs. Estimates of coastal populations. *Science*, 278(5341):1209–1213, 1997.
- [3] Saakshi Kapoor, Mukesh Kumar, and Manisha Kaushal. Deep learning based whale detection from satellite imagery. *Sustainable Computing: Informatics and Systems*, 38:100858, 2023.
- [4] Bento Collares Gonçalves, Bradley Spitzbart, and Heather J Lynch. Sealnet: A fully-automated pack-ice seal detection pipeline for sub-meter satellite imagery. *Remote Sensing of Environment*, 239:111617, 2020.
- [5] Antonio-Javier Gallego, Antonio Pertusa, and Pablo Gil. Automatic ship classification from optical aerial images with convolutional neural networks. *Remote Sensing*, 10(4):511, 2018.
- [6] Qingpeng Li, Lichao Mou, Qingjie Liu, Yunhong Wang, and Xiao Xiang Zhu. Hsf-net: Multiscale deep feature embedding for ship detection in optical remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 56(12):7147–7161, 2018.
- [7] Eleftherios Lygouras, Nicholas Santavas, Anastasios Taitzoglou, Konstantinos Tarchanidis, Athanasios Mitropoulos, and Antonios Gasteratos. Unsupervised human detection with an embedded vision system on a fully autonomous uav for search and rescue operations. *Sensors*, 19(16):3542, 2019.
- [8] Emilio Guirado, Siham Tabik, Marga L Rivas, Domingo Alcaraz-Segura, and Francisco Herrera. Whale counting in satellite and aerial images with deep learning. *Scientific reports*, 9(1):14259, 2019.
- [9] Katherine M Green, Mala K Virdee, Hannah C Cubaynes, Angelica I Aviles-Rivero, Peter T Fretwell, Patrick C Gray, David W Johnston, Carola-Bibiane Schönlieb, Leigh G Torres, and Jennifer A Jackson. Gray whale detection in satellite imagery using deep learning. *Remote Sensing in Ecology and Conservation*, 2023.

- [10] Alex Borowicz, Hieu Le, Grant Humphries, Georg Nehls, Caroline Höschle, Vladislav Kosarev, and Heather J Lynch. Aerial-trained deep learning networks for surveying cetaceans from satellite imagery. *PloS one*, 14(10):e0212532, 2019.
- [11] Robert Bogucki, Marek Cygan, Christin Brangwynne Khan, Maciej Klimek, Jan Kanty Milczek, and Marcin Mucha. Applying deep learning to right whale photo identification. *Conservation Biology*, 33(3):676–684, 2019.
- [12] Benjamin Kiefer, David Ott, and Andreas Zell. Leveraging synthetic data in object detection on unmanned aerial vehicles. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 3564–3571. IEEE, 2022.
- [13] Leon Amadeus Varga, Benjamin Kiefer, Martin Messmer, and Andreas Zell. Seadronessee: A maritime benchmark for detecting humans in open water. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2260–2270, 2022.
- [14] Behzad Sadrifaridpour, Yiannis Aloimonos, Miao Yu, Yang Tao, and Donald Webster. Detecting and counting oysters. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2156–2162. IEEE, 2021.
- [15] Jie Li, Katherine A Skinner, Ryan M Eustice, and Matthew Johnson-Roberson. Watergan: Unsupervised generative network to enable real-time color correction of monocular underwater images. *IEEE Robotics and Automation letters*, 3(1):387–394, 2017.
- [16] Bharat Joshi, Md Modasshir, Travis Manderson, Hunter Damron, Marios Xanthidis, Alberto Quattrini Li, Ioannis Rekleitis, and Gregory Dudek. Deepurl: Deep pose estimation framework for underwater relative localization. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1777–1784. IEEE, 2020.
- [17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- [18] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
- [19] Justine Boulent, Bertrand Charry, Malcolm McHugh Kennedy, Emily Tissier, Raina Fan, Marianne Marcoux, Cortney A Watt, and Antoine Gagné-Turcotte. Scaling whale monitoring using deep learning: A human-in-the-loop solution for analyzing aerial datasets. *Frontiers in Marine Science*, 10:1099479, 2023.
- [20] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes: Synthetic data takes flight. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 207–217, 2021.

- [21] Ian Cox, Jeremy Howitt, and John Duncan. The use of simulation to de-risk maritime uav operations. Technical report, SYSTEM ENGINEERING AND ASSESSMENT LTD BRISTOL (UNITED KINGDOM), 2007.
- [22] Omar Velasco, João Valente, Pablo J Alhama Blanco, and Mohammed Abderrahim. An open simulation strategy for rapid control design in aerial and maritime drone teams: A comprehensive tutorial. *Drones*, 4(3):37, 2020.
- [23] Shadi Abujoub, Johanna Mcphee, Cassidy Westin, and Rishad A Irani. Unmanned aerial vehicle landing on maritime vessels using signal prediction of the ship motion. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–9. IEEE, 2018.
- [24] Inc. The MathWorks. *UAV Toolbox*. Natick, Massachusetts, United State, 2019.
- [25] Peder Georg Olofsson Zwiłgmeyer. Creating a synthetic underwater dataset for egomotion estimation and 3d reconstruction. Master’s thesis, NTNU, 2021.
- [26] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.
- [27] Musa Morena Marcusso Manhães, Sebastian A Scherer, Martin Voss, Luiz Ricardo Douat, and Thomas Rauschenbach. Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–8. IEEE, 2016.
- [28] Javier Pérez, Jorge Sales, Mario Prats, José V Martí, David Fornas, Raúl Marín, and Pedro J Sanz. The underwater simulator uwsim-benchmarking capabilities on autonomous grasping. pages 369–376, 2013.
- [29] Rui Wang and Xuelel Qian. *Openscenegraph 3.0: Beginner’s guide*. 2010.
- [30] Easton Potokar, Spencer Ashford, Michael Kaess, and Joshua G Mangelson. Holoocean: An underwater robotics simulator. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3040–3046. IEEE, 2022.
- [31] Joshua Greaves, Max Robinson, Nick Walton, Mitchell Mortensen, Robert Pottorff, Connor Christopherson, Derek Hancock, Jayden Milne, and David Wingate. *Holodeck: A high fidelity simulator*, 2018.
- [32] Ivan Lončar, Juraj Obradović, Natko Kraševac, Luka Mandić, Igor Kvasić, Fausto Ferreira, Vladimir Slošić, Đula Nađ, and Nikola Mišković. Marus-a marine robotics simulator. In *OCEANS 2022, Hampton Roads*, pages 1–7. IEEE, 2022.
- [33] Abdelhakim Amer, Olaya Álvarez-Tuñón, Halil İbrahim Uğurlu, Jonas Le Fevre Sejersen, Yury Brodskiy, and Erdal Kayacan. Unav-sim: A visually realistic underwater robotics simulator and synthetic data-generation framework. In *2023 21st International Conference on Advanced Robotics (ICAR)*, pages 570–576. IEEE, 2023.

- [34] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*, pages 270–282. PMLR, 2018.
- [35] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018.
- [36] Mabel M Zhang, Woen-Sug Choi, Jessica Herman, Duane Davis, Carson Vogt, Michael McCarrin, Yadunund Vijay, Dharini Dutia, William Lew, Steven Peters, et al. Dave aquatic virtual environment: Toward a general underwater robotics simulator. In *2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*, pages 1–8. IEEE, 2022.
- [37] Leif Christensen, José de Gea Fernández, Marc Hildebrandt, Christian Ernst Siegfried Koch, and Bilal Wehbe. Recent advances in ai for navigation and control of underwater robots. *Current Robotics Reports*, pages 1–11, 2022.
- [38] Boris Braginsky and Hugo Guterman. Obstacle avoidance approaches for autonomous underwater vehicle: Simulation and experimental results. *IEEE Journal of oceanic engineering*, 41(4):882–892, 2016.
- [39] Niankai Yang, Dongsik Chang, Matthew Johnson-Roberson, and Jing Sun. Energy-optimal path planning with active flow perception for autonomous underwater vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9928–9934. IEEE, 2021.
- [40] Marios Xanthidis, Nare Karapetyan, Hunter Damron, Sharmin Rahman, James Johnson, Allison O’Connell, Jason M O’Kane, and Ioannis Rekleitis. Navigation in the presence of obstacles for an agile autonomous underwater vehicle. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 892–899. IEEE, 2020.
- [41] Marios Xanthidis, Michail Kalaitzakis, Nare Karapetyan, James Johnson, Nikolaos Vitzilaios, Jason M O’Kane, and Ioannis Rekleitis. Aquavis: A perception-aware autonomous navigation framework for underwater vehicles. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5410–5417. IEEE, 2021.
- [42] Ricardo Pérez-Alcocer, L Abril Torres-Méndez, Ernesto Olguín-Díaz, and A Alejandro Maldonado-Ramírez. Vision-based autonomous underwater vehicle navigation in poor visibility conditions using a model-free robust control. *Journal of Sensors*, 2016, 2016.
- [43] Adrian Manzanilla, Sergio Reyes, Miguel Garcia, Diego Mercado, and Rogelio Lozano. Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision. *IEEE Robotics and Automation Letters*, 4(2):1351–1356, 2019.
- [44] Valery A Bobkov, Vladimir Yu Mashentsev, Anton Yu Tolstonogov, and Alexander Ph Scherbatyuk. Adaptive method for auv navigation using stereo vision. In *The 26th International Ocean and Polar Engineering Conference*. OnePetro, 2016.

- [45] Enric Galceran and Marc Carreras. Efficient seabed coverage path planning for asvs and auvs. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 88–93. IEEE, 2012.
- [46] Enric Galceran, Ricard Campos, Narcís Palomeras, David Ribas, Marc Carreras, and Pere Ridao. Coverage path planning with real-time replanning and surface reconstruction for inspection of three-dimensional underwater structures using autonomous underwater vehicles. *Journal of Field Robotics*, 32(7):952–983, 2015.
- [47] Eduard Vidal, Juan David Hernández, Klemen Istenič, and Marc Carreras. Online view planning for inspecting unexplored underwater structures. *IEEE Robotics and Automation Letters*, 2(3):1436–1443, 2017.
- [48] Eduard Vidal, Narcís Palomeras, Klemen Istenič, Juan David Hernández, and Marc Carreras. Two-dimensional frontier-based viewpoint generation for exploring and mapping underwater environments. *Sensors*, 19(6):1460, 2019.
- [49] Travis Manderson, Juan Camilo Gamboa Higuera, Ran Cheng, and Gregory Dudek. Vision-based autonomous underwater swimming in dense coral for combined collision avoidance and target selection. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1885–1891. IEEE, 2018.
- [50] Nare Karapetyan, James V Johnson, and Ioannis Rekleitis. Human diver-inspired visual navigation: Towards coverage path planning of shipwrecks. *Marine Technology Society Journal*, 55(4):24–32, 2021.
- [51] Travis Manderson, Juan Camilo Gamboa Higuera, Stefan Wapnick, Jean-François Tremblay, Florian Shkurti, Dave Meger, and Gregory Dudek. Vision-based goal-conditioned policies for underwater navigation in the presence of obstacles. *Robotics: Science and Systems XVI*, Jul 2020.
- [52] Tianchen Liu, Miao Yu, and Nikhil Chopra. Learning-based autonomous underwater vehicle navigation following human actions in confined environment. In *OCEANS 2022, Hampton Roads*, pages 1–8. IEEE, 2022.
- [53] Travis Manderson, Juan Camilo Gamboa Higuera, Stefan Wapnick, Jean-François Tremblay, Florian Shkurti, David Meger, and Gregory Dudek. Vision-based goal-conditioned policies for underwater navigation in the presence of obstacles. *arXiv preprint arXiv:2006.16235*, 2020.
- [54] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4241–4247. IEEE, 2017.
- [55] Xiang Chen, Yuanchang Liu, and Kamalasudhan Achuthan. Wodis: Water obstacle detection network based on image segmentation for autonomous surface vehicles in maritime environments. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2021.

- [56] Borja Bovcon and Matej Kristan. Wasr—a water segmentation and refinement maritime obstacle detection network. *IEEE Transactions on Cybernetics*, 52(12):12661–12674, 2021.
- [57] Lorenzo Steccanella, Domenico Daniele Bloisi, Alberto Castellini, and Alessandro Farinelli. Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring. *Robotics and Autonomous Systems*, 124:103346, 2020.
- [58] Rundong Zhou, Yulong Gao, Yang Wang, Xingxiang Xie, and Xiongwei Zhao. A real-time scene parsing network for autonomous maritime transportation. *IEEE Transactions on Instrumentation and Measurement*, 72:1–14, 2022.
- [59] Remy Vandaele, Sarah L Dance, and Varun Ojha. Deep learning for automated river-level monitoring through river-camera images: an approach based on water segmentation and transfer learning. *Hydrology and Earth System Sciences*, 25(8):4435–4453, 2021.
- [60] Nur Atirah Muhadi, Ahmad Fikri Abdullah, Siti Khairunniza Bejo, Muhammad Razif Mahadi, and Ana Mijic. Deep learning semantic segmentation for water level estimation using surveillance camera. *Applied Sciences*, 11(20):9691, 2021.
- [61] Franz Wagner, Anette Eltner, and Hans-Gerd Maas. River water segmentation in surveillance camera images: A comparative study of offline and online augmentation using 32 cnns. *International Journal of Applied Earth Observation and Geoinformation*, 119:103305, 2023.
- [62] Yongqing Liang, Navid Jafari, Xing Luo, Qin Chen, Yanpeng Cao, and Xin Li. Waternet: An adaptive matching pipeline for segmenting water with volatile appearance. *Computational Visual Media*, 6:65–78, 2020.
- [63] Xiaomin Lin, Nitin J Sanket, Nare Karapetyan, and Yiannis Aloimonos. Oysternet: Enhanced oyster detection using simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5170–5176. IEEE, 2023. © 2023 IEEE. Reprinted, with permission, from Lin, Xiaomin et al., ”Oysternet: Enhanced oyster detection using simulation,” 2023 IEEE International Conference on Robotics and Automation (ICRA), July 2023.
- [64] Xiaomin Lin, Vivek Mange, Arjun Suresh, Bernhard Neuberger, Aadi Palnitkar, Brendan Campbell, Alan Williams, Kleio Baxevani, Jeremy Mallette, Alhim Vera, et al. Odyssee: Oyster detection yielded by sensor systems on edge electronics. *arXiv preprint arXiv:2409.07003*, 2024.
- [65] Xiaomin Lin, Nare Karapetyan, Kaustubh Joshi, Tianchen Liu, Nikhil Chopra, Miao Yu, Pratap Tokekar, and Yiannis Aloimonos. Uivnav: Underwater information-driven vision-based navigation via imitation learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5250–5256. IEEE, 2024.
- [66] Kaustubh Joshi, Tianchen Liu, Alan Williams, Matthew Gray, Xiaomin Lin, and Nikhil Chopra. 3d water quality mapping using invariant extended kalman filtering for underwater

- robot localization. *IEEE IROS workshop on Autonomous Robotic Systems in Aquaculture: Research Challenges and Industry Needs*, 2024.
- [67] Pei-Chun Chen, Yen-Cheng Chiang, and Pei-Yi Weng. Imaging using unmanned aerial vehicles for agriculture land use classification. *Agriculture*, 10(9):416, 2020.
 - [68] Özlem Akar. Mapping land use with using rotation forest algorithm from uav images. *European Journal of Remote Sensing*, 50(1):269–279, 2017.
 - [69] Rahul Raj, Soumyashree Kar, Rohit Nandan, and Adinarayana Jagarlapudi. Precision agriculture and unmanned aerial vehicles (uavs). In *Unmanned Aerial Vehicle: Applications in Agriculture and Environment*, pages 7–23. Springer, 2020.
 - [70] Dimosthenis C Tsouros, Stamatia Bibi, and Panagiotis G Sarigiannidis. A review on uav-based applications for precision agriculture. *Information*, 10(11):349, 2019.
 - [71] Richard Adade, Abiodun Musa Aibinu, Bernard Ekumah, and Jerry Asaana. Unmanned aerial vehicle (uav) applications in coastal zone management—a review. *Environmental Monitoring and Assessment*, 193(3):1–12, 2021.
 - [72] Rodrigo Saar De Moraes and Edison Pignaton De Freitas. Multi-uav based crowd monitoring system. *IEEE Transactions on Aerospace and Electronic Systems*, 56(2):1332–1345, 2019.
 - [73] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. Uav-based iot platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017.
 - [74] Antonio Albanese, Vincenzo Sciancalepore, and Xavier Costa-Pérez. Sardo: An automated search-and-rescue drone-based solution for victims localization. *IEEE Transactions on Mobile Computing*, 21(9):3312–3325, 2021.
 - [75] Jonathan Cacace, Alberto Finzi, and Vincenzo Lippiello. Multimodal interaction with multiple co-located drones in search and rescue missions. *arXiv preprint arXiv:1605.07316*, 2016.
 - [76] Antonio-Javier Gallego, Antonio Pertusa, Pablo Gil, and Robert B Fisher. Detection of bodies in maritime rescue operations using unmanned aerial vehicles with multispectral cameras. *Journal of Field Robotics*, 36(4):782–796, 2019.
 - [77] Yunus Karaca, Mustafa Cicek, Ozgur Tatli, Aynur Sahin, Sinan Pasli, Muhammed Fatih Beser, and Suleyman Turedi. The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations. *The American journal of emergency medicine*, 36(4):583–588, 2018.
 - [78] Balmukund Mishra, Deepak Garg, Pratik Narang, and Vipul Mishra. Drone-surveillance for search and rescue in natural disaster. *Computer Communications*, 156:1–10, 2020.
 - [79] SP Yeong, LM King, and SS Dol. A review on marine search and rescue operations using unmanned aerial vehicles. *International Journal of Marine and Environmental Sciences*, 9(2):396–399, 2015.

- [80] Patrick José Pereira. bluerov ros playground, 2018.
- [81] Zi-Hao Wang, Te Zhang, Kai-Yu Qin, and Bo Zhu. A vision-aided navigation system by ground-aerial vehicle cooperation for uav in gnss-denied environments. In *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, pages 1–6. IEEE, 2018.
- [82] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [83] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019.
- [84] Hannah C Cubaynes, Peter T Fretwell, Connor Bamford, Laura Gerrish, and Jennifer A Jackson. Whales from space: four mysticete species described using new vhr satellite imagery. *Marine Mammal Science*, 35(2):466–491, 2019.
- [85] Nare Karapetyan. *Robot Area Coverage Path Planning in Aquatic Environments*. PhD thesis, University of South Carolina, 2021.
- [86] Hannah C Cubaynes and Peter T Fretwell. Whales from space dataset, an annotated satellite image dataset of whales for training machine learning models. *Scientific Data*, 9(1):245, 2022.
- [87] P.J. Mumby, E.P. Green, A.J. Edwards, and C.D. Clark. The cost-effectiveness of remote sensing for tropical coastal resources assessment and management. *Journal of Environmental Management*, 55(3):157–166, March 1999.
- [88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [89] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.
- [90] Jeroen Bertels, Tom Eelbode, Maxim Berman, Dirk Vandermeulen, Frederik Maes, Raf Bisschops, and Matthew B Blaschko. Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II* 22, pages 92–100. Springer, 2019.
- [91] Roger IE Newell. Ecological changes in chesapeake bay: are they the result of overharvesting the american oyster, *crassostrea virginica*. *Understanding the estuary: advances in Chesapeake Bay research*, 129:536–546, 1988.
- [92] Bernadette Pogoda. Current status of european oyster decline and restoration in germany. *Humanities*, 8(1):9, 2019.

- [93] Lesley P Baggett, Sean P Powers, Robert D Brumbaugh, Loren D Coen, Bryan M DeAngelis, Jennifer K Greene, Boze T Hancock, Summer M Morlock, Brian L Allen, Denise L Breitburg, et al. Guidelines for evaluating performance of oyster habitat restoration. *Restoration Ecology*, 23(6):737–745, 2015.
- [94] Brittany N Blomberg, Terence A Palmer, Paul A Montagna, and Jennifer Beseres Pollack. Habitat assessment of a restored oyster reef in south texas. *Ecological Engineering*, 122:48–61, 2018.
- [95] Katherine McFarland and Matthew P Hare. Restoring oysters to urban estuaries: Redefining habitat quality for eastern oyster performance near new york city. *PloS one*, 13(11):e0207368, 2018.
- [96] Seth J Theuerkauf, David B Eggleston, and Brandon J Puckett. Integrating ecosystem services considerations within a gis-based habitat suitability index for oyster restoration. *PloS one*, 14(1):e0210936, 2019.
- [97] Michael W Beck, Robert D Brumbaugh, Laura Airoidi, Alvar Carranza, Loren D Coen, Christine Crawford, Omar Defeo, Graham J Edgar, Boze Hancock, Matthew C Kay, et al. Oyster reefs at risk and recommendations for conservation, restoration, and management. *Bioscience*, 61(2):107–116, 2011.
- [98] C Sullivan and Alexander Kaszynski. Pyvista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (vtk). *Journal of Open Source Software*, 4(37):1450, 2019.
- [99] Taesung Park, Alexei A Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. In *European Conference on Computer Vision*, pages 319–345. Springer, 2020.
- [100] Loren D Coen, Robert D Brumbaugh, David Bushek, Ray Grizzle, Mark W Luckenbach, Martin H Posey, Sean P Powers, and S Gregory Tolley. Ecosystem services related to oyster restoration. *Marine Ecology Progress Series*, 341:303–307, 2007.
- [101] Jonathan H Grabowski and Charles H Peterson. Restoring oyster reefs to recover ecosystem services. *Ecosystem engineers: plants to protists*, 4:281–298, 2007.
- [102] Brendan Campbell, Cassie Gurdysz, Jeremy Testa, Edward Hale, and Matthew W Gray. Assessing the spatial variability of cage movement and velocity attenuation of an off-bottom oyster farm and its influence on eastern oyster (*crassostrea virginica*) growth performance. *Available at SSRN 4858563*, 2024.
- [103] Weimin Quan, Ruiliang Fan, Yunlong Wang, and Austin T Humphries. Long-term oyster recruitment and growth are not influenced by substrate type in china: implications for sustainable oyster reef restoration. *Journal of Shellfish Research*, 36(1):79–86, 2017.
- [104] Melanie J Bishop, Brendan S Lanham, Juan R Esquivel-Muelbert, Victoria J Cole, Kristine M Faelnar, Charlotte Jenkins, Jillian Keating, Francisco Martínez-Baena, and

- Wayne A O'Connor. Oyster reef restoration-aquaculture interactions: maximizing positive synergies. *Frontiers in Marine Science*, 10:1162487, 2023.
- [105] WE Pine III, FA Johnson, PC Frederick, and LG Coggins. Adaptive management in practice and the problem of application at multiple scales—insights from oyster reef restoration on florida’s gulf coast. *Marine and Coastal Fisheries*, 14(1):e10192, 2022.
- [106] James A Fitzsimons, Simon Branigan, Chris L Gillies, Robert D Brumbaugh, Jun Cheng, Bryan M DeAngelis, Laura Geselbracht, Boze Hancock, Andrew Jeffs, Tein McDonald, et al. Restoring shellfish reefs: Global guidelines for practitioners and scientists. *Conservation Science and Practice*, 2(6):e198, 2020.
- [107] David Bushek, Jennifer Gius, and Jason Morson. 2024 stock assessment workshop report, 2024. Final Report.
- [108] Jeannie Haddaway-Riccio. 2020 maryland oyster stock assessment update summary, July 2020.
- [109] Robert Botta, Frank Asche, J Scott Borsum, and Edward V Camp. A review of global oyster aquaculture production and consumption. *Marine Policy*, 117:103952, 2020.
- [110] Ganesh Kumar, Carole Engle, and Craig Tucker. Factors driving aquaculture technology adoption. *Journal of the world aquaculture society*, 49(3):447–476, 2018.
- [111] Yang Yang, Yang Zhou, Xuan Yue, Gang Zhang, Xing Wen, Biao Ma, Liangyuan Xu, and Liqing Chen. Real-time detection of crop rows in maize fields based on autonomous extraction of roi. *Expert Systems with Applications*, 213:118826, 2023.
- [112] Kevin DE Stokesbury, Edward P Baker, Bradley P Harris, and Robert B Rheault. Environmental impacts related to mechanical harvest of cultured shellfish. *Shellfish aquaculture and the environment*, pages 319–338, 2011.
- [113] Renee Mercaldo-Allen, Peter J Auster, Paul Clark, Mark S Dixon, Erick Estela, Yuan Liu, Lisa Milke, Gillian Phillips, Dylan Redman, Barry C Smith, et al. Oyster aquaculture cages provide fish habitat similar to natural structure with minimal differences based on farm location. *Frontiers in Marine Science*, 10:1058709, 2023.
- [114] Karl B Veggerby, Mark D Scheuerell, Beth L Sanderson, Peter M Kiffney, and Bridget E Ferriss. Shellfish aquaculture farms as foraging habitat for nearshore fishes and crabs. *Marine and Coastal Fisheries*, 16(2):e10282, 2024.
- [115] RM Connolly, C Herrera, J Rasmussen, CA Buelow, M Sievers, KI Jinks, CJ Brown, S Lopez-Marcano, CDH Sherman, F Martínez-Baena, et al. Estimating enhanced fish production on restored shellfish reefs using automated data collection from underwater videos. *Journal of Applied Ecology*, 61(4):633–646, 2024.
- [116] Adam D Armbruster, Renee Mercaldo-Allen, Julie M Rose, Kristen Seda, Paul Clark, Gillian Phillips, Dylan Redman, and Christian W Conroy. Territorial and occupancy behavior of black sea bass on oyster aquaculture gear and boulder habitat. *Frontiers in Marine Science*, 11:1380484, 2024.

- [117] Md Modasshir and Ioannis Rekleitis. Augmenting coral reef monitoring with an enhanced detection system. In *IEEE International Conference on Robotics and Automation*, pages 1874–1880, Paris, France, 2020.
- [118] Md Modasshir, Sharmin Rahman, and Ioannis Rekleitis. Autonomous 3d semantic mapping of coral reefs. In *Results of the 12th International Conference of Field and Service Robotics*, pages 365–379. Springer, 2021.
- [119] Gregory Dudek, Michael Jenkin, Chris Prahacs, Andrew Hogue, Junaed Sattar, Philippe Giguere, Andrew German, Hui Liu, Shane Saunderson, Arlene Ripsman, et al. A visually guided swimming robot. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3604–3609. IEEE, 2005.
- [120] Bharat Joshi, Marios Xanthidis, Monika Roznere, Nathaniel J Burgdorfer, Philippos Mordohai, Alberto Quattrini Li, and Ioannis Rekleitis. Underwater exploration and mapping. In *2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*, pages 1–7. IEEE, 2022.
- [121] Johanna Hansen, Sandeep Manjanna, Alberto Quattrini Li, Ioannis Rekleitis, and Gregory Dudek. Autonomous marine sampling enhanced by strategically deployed drifters in marine flow fields. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–7. IEEE, 2018.
- [122] Nare Karapetyan, Jason Moulton, and Ioannis Rekleitis. Meander-based river coverage by an autonomous surface vehicle. In *Field and Service Robotics*, pages 353–364. Springer, 2021.
- [123] Sandeep Manjanna, Nikhil Kakodkar, Malika Meghjani, and Gregory Dudek. Efficient terrain driven coral coverage using gaussian processes for mosaic synthesis. In *2016 13th Conference on Computer and Robot Vision (CRV)*, pages 448–455. IEEE, 2016.
- [124] Fahad Jalal and Faizan Nasir. Underwater navigation, localization and path planning for autonomous vehicles: A review. In *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*, pages 817–828. IEEE, 2021.
- [125] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis. Efficient complete coverage of a known arbitrary environment with applications to aerial operations. *Autonomous Robots*, 36:365–381, 2014.
- [126] Nare Karapetyan, Jason Moulton, Jeremy S. Lewis, Alberto Quattrini Li, Jason M. O’Kane, and Ioannis M. Rekleitis. Multi-robot dubins coverage with autonomous surface vehicles. In *icra*, 2018.
- [127] Bharat Joshi, Sharmin Rahman, Michail Kalaitzakis, Brennan Cain, James Johnson, Marios Xanthidis, Nare Karapetyan, Alan Hernandez, Alberto Quattrini Li, Nikolaos Vitzilaios, et al. Experimental comparison of open source visual-inertial-based state estimation algorithms in the underwater domain. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7227–7233. IEEE, 2019.

- [128] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.
- [129] Boxiao Yu, Jiayi Wu, and Md Jahidul Islam. Udepth: Fast monocular depth estimation for visually-guided underwater robots. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3116–3123. IEEE, 2023.
- [130] Xiaomin Lin, Nitesh Jha, Mayank Joshi, Nare Karapetyan, Yiannis Aloimonos, and Miao Yu. Oystersim: Underwater simulation for enhancing oyster reef monitoring. In *OCEANS 2022, Hampton Roads*, pages 1–6. IEEE, 2022.
- [131] Francois Chollet et al. Keras, 2015.
- [132] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.
- [133] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [134] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.
- [135] Winston C Chow. Brownian bridge. *Wiley interdisciplinary reviews: computational statistics*, 1(3):325–332, 2009.
- [136] Jon S Horne, Edward O Garton, Stephen M Krone, and Jesse S Lewis. Analyzing animal movements using brownian bridges. *Ecology*, 88(9):2354–2363, 2007.
- [137] Bao Pang, Yong Song, Chengjin Zhang, Hongling Wang, and Runtao Yang. A swarm robotic exploration strategy based on an improved random walk method. *Journal of Robotics*, 2019:1–9, 2019.
- [138] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. Robotic exploration, brownian motion and electrical resistance. In *Randomization and Approximation Techniques in Computer Science: Second International Workshop, RANDOM’98 Barcelona, Spain, October 8–10, 1998 Proceedings 2*, pages 116–130. Springer, 1998.
- [139] Blue Robotics. Bluerov2: The world’s most affordable high-performance rov. *BlueROV2 Datasheet; Blue Robotics: Torrance, CA, USA*, 2016.
- [140] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [141] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2016.

- [142] Boxiao Yu, Jiayi Wu, and Md Jahidul Islam. Udepth: Fast monocular depth estimation for visually-guided underwater robots. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3116–3123, 2023.
- [143] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [144] Jiayi Wu. *Low-Cost Depth Estimation and 3D Reconstruction in Scattering Medium*. PhD thesis, University of Florida, 2023.
- [145] Jiayi Wu, Boxiao Yu, and Md Jahidul Islam. 3d reconstruction of underwater scenes using nonlinear domain projection. In *2023 IEEE Conference on Artificial Intelligence (CAI)*, pages 359–361, 2023.
- [146] Pushkal Katara, Mukul Khanna, Harshit Nagar, and Annapurani Panaiyappan. Open source simulator for unmanned underwater vehicles using ros and unity3d. In *2019 IEEE Underwater Technology (UT)*, pages 1–7. IEEE, 2019.
- [147] Akash Chaudhary, Rajat Mishra, Bharath Kalyan, and Mandar Chitre. Development of an underwater simulator using unity3d and robot operating system. In *OCEANS 2021 MTS/IEEE*. IEEE, Sep 2021.
- [148] Jeya Maria Jose Valanarasu and Vishal M. Patel. Unext: Mlp-based rapid medical image segmentation network, 2022.
- [149] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.
- [150] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018.
- [151] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018.
- [152] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. *Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations*, page 240–248. Springer International Publishing, 2017.
- [153] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.
- [154] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [155] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

- [156] Hanchao Li, Pengfei Xiong, Jie An, and Lingxue Wang. Pyramid attention network for semantic segmentation, 2018.
- [157] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [158] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network, 2017.
- [159] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey, 2020.
- [160] Peter Christen, David J. Hand, and Nishadi Kirielle. A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM Comput. Surv.*, 56(3), oct 2023.
- [161] Fortune Business Insights. Olive oil market size, trends amp; growth: Global forecast [2029], Sep 2022.
- [162] Steven W Chen, Shreyas S Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J Taylor, and Vijay Kumar. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017.
- [163] Mohamad Haniff Junos, Anis Salwa Mohd Khairuddin, Subbiah Thannirmalai, and Mahidzal Dahari. An optimized yolo-based object detection model for crop harvesting system. *IET Image Processing*, 15(9):2112–2125, 2021.
- [164] Muhammad Taufiq Pratama, Sangwook Kim, Seiichi Ozawa, Takenao Ohkawa, Yuya Chona, Hiroyuki Tsuji, and Noriyuki Murakami. Deep learning-based object detection for crop monitoring in soybean fields. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.
- [165] Xi Chen, Zuoxin Li, Ye Yuan, Gang Yu, Jianxin Shen, and Donglian Qi. State-aware tracker for real-time video object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9384–9393, 2020.
- [166] Tianchen Liu, Nikhil Chopra, and Jayesh Samtani. Information system for detecting strawberry fruit locations and ripeness conditions in a farm. In *Biology and Life Sciences Forum*, volume 16, page 22. MDPI, 2022.
- [167] Petar Dz. How many olives are harvested on an average olive tree? (detailed answer), Jan 2022.
- [168] Intel. Intel® open image denoise.
- [169] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

- [170] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.
- [171] Justin Theiss, Jay Leverett, Daeil Kim, and Aayush Prakash. Unpaired image translation via vector symbolic architectures. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI*, pages 17–32. Springer, 2022.
- [172] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, USA, 1988.
- [173] Boxiao Yu, Jiayi Wu, and Md Jahidul Islam. Udepth: Fast monocular depth estimation for visually-guided underwater robots. *arXiv preprint arXiv:2209.12358*, 2022.
- [174] Kentaro Wada. Labelme: Image Polygonal Annotation with Python.
- [175] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III* 18, pages 234–241. Springer, 2015.
- [176] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [177] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [178] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [179] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.
- [180] Unity Technologies. Unity. <https://unity.com/>. Accessed: [date of access].
- [181] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
- [182] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII* 16, pages 649–665. Springer, 2020.
- [183] Xiaomin Lin, Cheng Liu, Allen Pattillo, Miao Yu, and Yiannis Aloimonous. Seadronesim: Simulation of aerial images for detection of objects above water. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 216–223, 2023. © 2023 IEEE. Reprinted, with permission, from Lin, Xiaomin et al., ”SeaDroneSim: Simulation of Aerial Images for Detection of Objects Above Water,” 2023 IEEE/CVF Winter Conference on Applications of Computer Vision, February 2023.

- [184] Akshaj Gaur, Cheng Liu, Xiaomin Lin, Nare Karapetyan, and Yiannis Aloimonos. Whale detection enhancement through synthetic satellite images. In *OCEANS 2023-MTS/IEEE US Gulf Coast*, pages 1–7. IEEE, 2023. © 2023 IEEE. Reprinted, with permission, from Lin, Xiaomin et al., "Whale Detection Enhancement through Synthetic Satellite Images," *OCEANS 2023-MTS/IEEE US Gulf Coast*, December 2023.
- [185] Jiayi Wu, Xiaomin Lin, Shahriar Negahdaripour, Cornelia Fermüller, and Yiannis Aloimonos. Marvis: Motion & geometry aware real and virtual image segmentation. 2024. © 2024 IEEE. Reprinted, with permission, from Lin, Xiaomin et al., "MARVIS: Motion Geometry Aware Real and Virtual Image Segmentation," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), September 2024.
- [186] Yianni Karabatis, Xiaomin Lin, Nitin J Sanket, Michail G Lagoudakis, and Yiannis Aloimonos. Detecting olives with synthetic or real data? olive the above. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4242–4249. IEEE, 2023. © 2023 IEEE. Reprinted, with permission, from Lin, Xiaomin et al., "Detecting Olives with Synthetic or Real Data? Olive the Above," 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), September 2023.