Toward Robotic Autonomy in Data-Scarce and Visually Challenging Environments

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Jungseok Hong

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Advisor: Dr. Junaed Sattar

June, 2023

Acknowledgements

First of all, this endeavor would not have been possible without my advisor, Professor Junaed Sattar, who saw the potential that I did not know I had in me. I am extremely grateful for his invaluable guidance, patience, and support throughout my Ph.D. years. His valuable feedback and insightful suggestions have been priceless in shaping my research. Additionally, the field trial opportunities he provided have greatly helped define my research and improved my understanding of real-world robotic challenges.

Additionally, I am grateful to the members of my thesis committee, Professor Nikolaos Papanikolopoulos, Catherine Qi Zhao, and Maziar Hemati. I am thankful to Professor Papanikolopoulos for ingenious suggestions from his robotics expertise and for being a great mentor since I took his course during my first semester. Thanks should also go to Professor Zhao for providing insightful feedback on my research from a computer vision and machine learning perspective. I would like to extend my sincere thanks to Professor Hemati for bringing his engineering expertise to give valuable feedback on my research.

Thanks to Michael Fulton, who is a friend and has been a constant research collaborator during my Ph.D. years. Thanks for your time discussing research ideas and answering my stochastic questions, which regularly happened late at night while we were working on our projects and even during the peak of COVID-19. These discussions gave my research momentum and improved my research.

I also want to acknowledge my fellow former and current Ph.D. students, Md Jahidul Islam, Jiawei Mo, Chelsey Edge, Karin de Langis, Sadman Sakib Enan, Corey Knutson, Demetri Kutzke, and Sakshi Singh. It was a great experience for me to work with and collaborate with them. I was motivated by their enthusiasm for research, and I very much appreciated suggestions and discussions on my research. Some already started successful careers, and I believe the rest of them will be thriving in the field as well.

I gratefully acknowledge the assistance of undergraduate students who I worked with: Marc Ho, Hannah DuBois, Mark Oh, Christopher Morse, Cole Wyeth, Christopher Walaszek, Kevin Orpen, Kimberly Barthelemy, Keara Berlin, and others. Their efforts enriched my experiments and improved my research. I would like to recognize numerous volunteers who participated in underwater experiments and data collection for my research.

I would like to express my appreciation to my internship mentors, Professor Volkan Isler, Hyun Soo Park, and Dr. Dimitris Zermas. Their expertise in robotics and diverse perspectives improved the breadth of my research. I must also thank Professor Changhyun Choi for being my mentor.

I would like to express gratitude for the generous support from the National Science Foundation, Minnesota Robotics Institute, MnDrive, and the University of Minnesota.

Finally, I would like to extend my deepest gratitude to my family. I am deeply indebted to my parents, Euiin Hong and Haeran Jung, for their unconditional love, support, encouragement, and understanding throughout my entire life. I learned from them how to turn obstacles into opportunities and pursue my goal. I am extremely grateful to my brother, Jungho Hong, for his love and support. I am also thankful to my parents-in-law for their support and encouragement. Lastly, words cannot express my gratitude and love to my supportive wife, Jiyeon Hwang. Her encouragement and understanding helped me to keep pursuing and achieving my goals with strong momentum.

Dedication

To my parents.

I dedicate this thesis to my dad, Euiin Hong, who intrigued my interest in science by spending numerous nights together making rubber-powered airplanes and water rockets after his long day at work, and to my mom, Haeran Jung, who taught me the passion to pursue my dream and the joy of reading and writing.

Abstract

Recent advances in field robotics have allowed robots to be used in challenging unstructured environments (*e.g.*, space, ocean, and natural disaster scenes) instead of sending human explorers since such environments could pose a threat to human lives. However, most field robots are not fully autonomous, relying on communication with human operators. The inefficiencies and constraints of this dependency result in the limited usage of field robots. Hence, enhancing robotic autonomy for unstructured environments is necessary to enable robots to handle large-scale and time-sensitive problems without operators.

Among such environments, the underwater domain needs immediate attention since the hydrosphere has high impacts on our lives. Specifically, water covers 71% of Earth's surface and has the largest ecosystem on Earth. Unfortunately, various human activities have negatively affected the health of underwater ecosystems, which threatens our Earth's ecosystem as a whole. One such example is the underwater debris problem. Underwater debris has already had damaging effects on the environment (*e.g.*, damaging wildlife habitats), and its long-term effects are predicted to be catastrophic. Existing methods for collecting underwater debris are ineffective, and human exploration and clean-up of the debris are prohibitively risky and cannot adapt to the scale of the problem. This shows it is necessary to deploy underwater robots, often called autonomous underwater vehicles (AUVs), to clean up underwater debris, but the lack of robotic autonomy discourages AUVs from being used for the task.

To bridge this gap, this thesis presents our efforts to improve robotic autonomy to make it possible for robots to handle challenging tasks, such as marine debris cleanup, in degraded environments. Part I introduces our efforts to improve robotic underwater object detection, focused on underwater debris, by addressing data scarcity and unique circumstances underwater (*e.g.*, modified shapes of objects over time, degraded vision). In Part II, we present localization and obstacle avoidance algorithms underwater to enable robots to explore a given environment safely and localize underwater objects after detecting them. Even with the improved object detection and navigation methods covered in Part I and II, robots can still struggle with autonomous behaviors due to the extreme challenges that unstructured environments pose in perception and navigation. In cases such as these, human-robot collaborations are still necessary, but robotic capabilities to establish the collaborations are under-investigated and need to be improved. To enable such collaborations, we present algorithms for effective human-robot collaboration in Part III. Lastly, in Part IV, we introduce open-source underwater robotic systems to support robotic research, including the development and deployment process of the research presented in Part I - III.

The research introduced in this thesis has been realized and tested on board physical AUVs in various water environments. While our research has mainly focused on the problems in the underwater environment, its robust results in this domain mean that it can be applied to real-world problems in other domains that share similar challenges. Thus, the research has great potential to significantly advance robotic autonomy, bring a broader impact on human well-being, and drive robotics forward for social good.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advances in field robotics have empowered robots to survey unstructured environments, which we define as naturally formed environments (*e.g.*, ocean, cave, and space). These robots are capable of collecting information about unknown environments without risking human safety or lives. For instance, NASA's Perseverance Rover [7] has been exploring Mars since 2020, and the Woods Hole Oceanographic Institution's remotely operated vehicle (ROV) Nereus, reached a depth of $9,990$ meters in the Kermadec Trench, northeast of New Zealand [8] in 2014. However, such robots rely on communication with human operators, often along long cables, to execute missions due to challenges from unstructured environments. This dependency limits the robots' usability and poses constraints when addressing large-scale and time-sensitive problems. The challenges posed by unstructured environments and the conceivable utility of greater robot autonomy in such environments can be illustrated in a few selected examples.

Example 1: Underwater Debris Collection

In 2006, the United Nations Environment Programme (UNEP) [9] reported that the ocean had $46,000$ pieces of floating plastic per square mile. Debris accumulates in the ocean in various ways. About 80% of the debris is land-based trash (e.g., plastic bags, beverage containers, and packaging) [10]. Ocean activities contribute to the remaining 20% of the debris, which is mostly abandoned commercial fishing gear [11]. Small water bodies like rivers, lakes, and ponds have a similar problem. For example, an article [12]

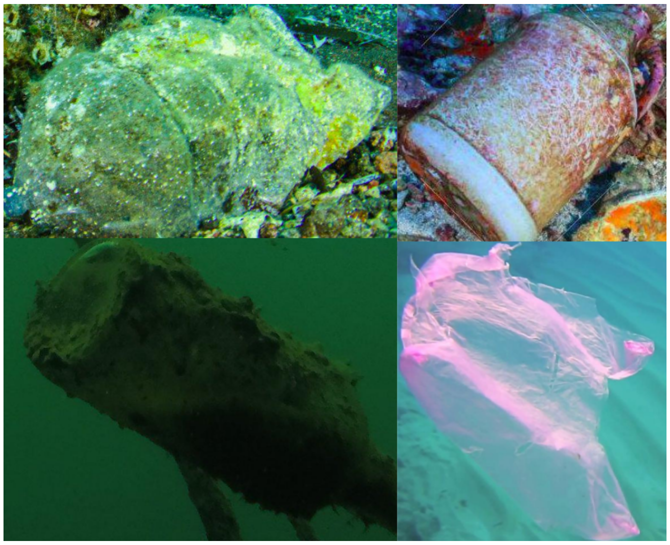


Figure 1.1: Sample images showing underwater trash made of plastic and glass at different stages of shape and color deformation. The images show challenges posed by the underwater domain, and the deformation makes object detection for underwater trash a significantly challenging problem.

reported that a single resident of Minnesota, USA collected more than $1,500$ pounds of debris falling into Lake Hiawatha from May to July 2015. Among all types of underwater debris, plastic debris is one of the most damaging (Fig. 1.1). It disintegrates into microplastics, which are too small to detect, after which it is consumed by a vast range of organisms [13]. The report from the European Union Action to Fight Environmental Crime (EFFACE) [14] shows that marine pollution causes a wide range of problems including, but not limited to, environmental, health, social, financial, and economic impacts. Hence, underwater debris is a critical issue for mankind that must be addressed. While some efforts, including recycling and waste management regulations, have been made to reduce the amount of underwater debris, there is still a large amount of debris in water bodies around the world. Some devices, such as floating seabins [15] and nets [16], passively collect floating marine debris, but these devices cannot collect submerged debris. Also, they may collect sea animals accidentally since they cannot determine whether an object is debris or not. People can both collect and identify objects, but the time and cost to address the problem in this way would be prohibitive, to say nothing of the potential threat toxic waste would pose to the people searching for and collecting the debris. Little research using autonomous robots to remove underwater debris has been conducted despite the fact that it could be safer and more scalable than the previously mentioned methods. This highly motivated me to develop the research presented in Part I - IV of this thesis to address this large-scale and time-sensitive problem with AUVs.

Example 2: Fukushima Daiichi Nuclear Disaster

The Fukashima Daiichi Nuclear incident [17] in 2011 was among the worst catastrophes in mankind's history. In 2011, a 15-meter tsunami following the Tōhoku earthquake damaged the generators which powered the reactor building. The loss of power disabled the cooling of three reactors at the Fukushima Daiichi nuclear power plant and caused a nuclear meltdown and chemical explosions. During these incidents, uranium fuel, which is highly radioactive, was lost. Highly trained operators from the Tokyo Electric Power Company (TEPCO) controlled ROVs to find missing melted uranium fuel from the reactors of the plant. Using ROVs was necessary because human exposure to the fuel would have resulted in lethal doses of radiation in a matter of seconds [18]. Even with such effort, the task of finding the fuel after the accident took six years. This delay was mainly due to the fact that robotic search missions encountered many failures because of degraded visibility caused by debris and excessive radiation, in addition to the time spent on training the operators. If the robots were able to find the fuel autonomously and collaborate with the operators instead of taking one-way commands, it could have taken less time to find the fuel and lessened radioactive contamination to the ecological system of the Earth, which can take thousands of years to decay. Similar to underwater debris in the previous example, the fuel has varying appearances, and finding enough data to predict the fuel's possible appearances can be difficult. Incidents such as this example motivated me to develop visual perception algorithms introduced in Part I of this thesis for challenging environments.

Example 3: Tham Luang Cave Rescue

In 2018, 12 members of a junior soccer team and their coach were trapped in the Tham Luang cave in northern Thailand due to the heavy rainfall that flooded part of the cave [19]. Degraded visibility and narrow routes of the cave with the risk of a seasonal monsoon made the rescue mission extremely challenging. The rescue work was expanded into international efforts soon after because of the amount of work required and its urgency. Consequently, more than $10,000$ people, including highly trained SEAL divers from across the world, were involved in the rescue work [20]. In addition to the human efforts, underwater robots and drones were deployed to aid search efforts, but the robotic operations were not successful [21, 22]. This was mainly due to the lack of robotic autonomy and the environmental challenges posed by the cave. The turbidity of its water caused degraded visibility, and this worsened vision-based robotic navigation. Also, GPS or Wi-Fi signals could not penetrate the ground, which made localizing inside the cave difficult. With only human labor for the rescue work, it took 18 days to rescue all the members, and two divers died during and after the rescue process. This shows the similar challenges that AUVs could face while they search for underwater debris in the first example. The urgency and potential benefit of addressing issues like this example inspired our research on robotic exploration and localization presented in Part II of this thesis. Furthermore, limited robotic autonomy due to the extreme challenges from unstructured environments like caves motivated our research for enabling human-robot collaboration presented in Part III of this thesis to augment robotic autonomy.

The aforementioned examples show existing challenges in unstructured environments which have visually degraded conditions and data-scarcity issues. For instance, in these environments, robots have to recognize objects (*i.e.*, the melted fuel and underwater debris) while the objects do not hold constant shapes. Appearances of the objects could change drastically after some time, and existing images obtained from the ground would be ineffective in training any object detection models. Furthermore, robotic localization would be extremely difficult since the environmental factors (*i.e.*, excessive radiation, cave, and underwater) can deteriorate communication methods (*e.g.*, Wi-Fi and GPS) generally used to locate robots.

Among different types of unstructured environments, the underwater environment is one environment that needs immediate attention. The underwater environment has the largest ecosystem on Earth, but it has suffered severe destruction by various human activities (*e.g.*, littering, fishing, etc.) over the years. Particularly, underwater debris has severely damaged the underwater environment as described in the first example. Such destruction has started to affect billions of human lives and must be addressed. However, the problem is too large and dangerous for humans to handle. Hence, improving robotic autonomy to tackle such a problem is a must. Additionally, addressing the underwater debris problem can be beneficial in tackling problems with similar characteristics in other unstructured environments, as described in the above examples. Underwater robots, generally referred to as autonomous underwater Vehicles (AUVs), have additional challenges due to the inhospitable nature of their environment. These include: (1) the shape of each object can be modified over time due to chemical reactions, (2) color absorption occurs as depth increases, (3) light scatters underwater causing blurred and noisy underwater images, (4) GPS and other forms of radio frequency-based communications are either completely unavailable or limited to extremely short ranges, (5) landmark-based localization using exteroceptive sensors is not reliable due to the lack of distinctive features underwater, and (6) running the state-of-the-art computer vision and AI models on robotic hardware in real-time is often not feasible due to the lack of computational power.

Besides overcoming the listed challenges above to achieve autonomy, the proper type of AUVs needs to be deployed to address the examples introduced. It has been more than 60 years since the self propelled underwater research vehicle (SPURV) [23], one of the first AUVs, was developed in 1957. Since then, many AUVs have been introduced with improved hardware and lower cost [24, 25]. AUVs can be classified into small ($< 1m$), medium ($> 1m, < 3m$), large ($> 3m, < 7m$), and extra-large ($> 7m$) categories based on their sizes. Large and extra-large size AUVs can handle heavier payloads, perform longer missions, and be used for deep-sea exploration. However, a designated mothership is required to deploy these AUVs. On the other hand, small and medium AUVs have better versatility allowing access to narrow spaces such as an underwater valley and cave. They are generally equipped with imaging sensors and have limited computing power due to their small payload capacity.

Along with the progress of robotic hardware development, efforts to understand underwater objects from images with neural networks (NNs) started in 2005 [26, 27]. However, computational requirements for NNs have been a roadblock to running such algorithms on board AUVs. Also, the inherent data scarcity of the underwater domain prevents training NNs with their maximum capacity. This degrades the performance of NNs compared to their counterparts in in-air applications. In parallel with improving the performance of NNs, utilizing human expertise via human-robot collaboration (HRC) can help robots to understand previously unseen objects during missions, which results in improved robotic autonomy. To this date, underwater HRC has not gained much attention, although it has the potential to generate shared autonomy by exploiting the strengths of both humans and robots [28]. Additionally, underwater navigation, localization, and mapping are crucial problems to be addressed to increase robotic autonomy [25]. However, they still remain challenging for small and medium AUVs since algorithms addressing these problems generally rely on expensive and power-demanding sensors (*e.g.*, multi-beam sonar, long baseline (LBL), ultra-short baseline (USBL), and short baseline (SBL) systems). Furthermore, most AUVs are proprietary and costly, and this has contributed to the limited access to underwater robotic research.

Given this gap in the research, we present our efforts to address these problems and improve robotic autonomy in unstructured environments. More specifically, in this thesis, we will focus on developing algorithms with the goal of enabling robots to autonomously complete challenging tasks underwater, such as cleaning underwater debris. Considering the required maneuverability to navigate and identify underwater debris, we will concentrate on developing algorithms for small and medium AUVs. In particular, this thesis consists of the following four parts: Part I: visual perception in data-scarce and visually challenging underwater environments, Part II: localization and exploration in unstructured environments, Part III: algorithms for effective human-robot collaboration, and Part IV: developing open-source underwater robotic systems. A more detailed description of the parts of this thesis is given below. Additionally, we worked to address perception challenges in agricultural and grasping robots, demonstrating that our research is applicable across environments and applications. This additional work will be covered in Appendices. The research described in this thesis presents excellent opportunities to tackle a significant environmental problem and advance the autonomy of field robots. This work, in turn, has the potential to benefit society greatly and improve human well-being.

Part I: Visual Perception in Data-Scarce and Visually Challenging Underwater Environments

The first part of this thesis presents the substantial perception challenges underwater and methods to overcome such challenges for detecting objects underwater, where we focus on underwater debris detection. Part I consists of four chapters. Chapter 2 introduces the first deep learning-based approach [29] to address the underwater debris problem in robotic platforms. In Chapter 3, we present a dataset, *TrashCan* [30] created to improve the detection performance further with a larger amount of data. *TrashCan* is the first publicly available underwater image dataset consisting of $7,212$ pixel-level annotated images of debris, robots, and a wide variety of undersea flora and fauna. However, building large datasets is labor-intensive, and collecting imagery of underwater debris, which is scarce, is difficult and dangerous. This makes TrashCan a significant contribution to the autonomous detection of marine debris. In Chapter 4, we introduce a generative approach [31] to overcome data scarcity, producing realistic fake debris images to augment existing datasets using a two-stage variational autoencoder. Lastly, Chapter 5 presents an image blending method to generate images and corresponding annotations simultaneously, which eliminates time-consuming dataset labeling efforts for training object detection models. Part I presents significant first steps toward overcoming perceptual problems in challenging environments.

Part II: Robotic Exploration and Localization for AUVs

Part II of this thesis provides methods to find and identify objects underwater and is composed of two chapters. Underwater robots frequently encounter unknown environments. Chapter 6 proposes an approach to fuse depth and instance segmentation information with an artificial potential field algorithm (APF), which makes it possible for robots to keep flexible distances to objects (*i.e.*, stay away from bio-organisms or approach to objects of interests) during exploration. After detecting underwater debris from exploration, accurate localization of AUVs is essential to estimate the location of detected debris. Chapter 7 presents a low-cost approach by fusing bathymetry data of underwater environments with depth and altitude data from an AUV. Part II introduces robotic exploration and localization methods for underwater, which are stepping stones for collecting underwater debris autonomously while minimizing environmental interferences and maximizing operation time.

Part III: Algorithms for Effective Human-Robot Collaboration

Robotic operations are extremely challenging when AUVs explore fully unknown environments or face previously unseen objects. Human-robot collaboration can be useful to augment robotic perception underwater, but methods that allow a robot to find divers and position itself to facilitate interactions remain unidentified. Part III presents algorithms to enable underwater human-robot collaboration and consists of two chapters. Chapter 8 presents the first underwater human-approaching algorithm that enables robots to find the diver and achieve diver-relative positioning autonomously using monocular vision and body priors. Chapter 9 introduces diver identification methods for AUVs using face and pose information to secure human-robot collaboration. Part III introduces the approaches that can be used to augment robotic autonomy with human expertise while reducing the physical load for divers, thereby increasing human safety.

Part IV: Robotic Systems Development Toward On-board Object Detection

Part IV of this thesis has one chapter. In Chapter 10, we present two robotic systems developed to support the presented research, underwater detection, AUV localization, and exploration. One of them is LoCO, a low-cost and open-source AUV. It is a general-purpose, single-person-deployable, and vision-guided AUV rated to a depth of 100 meters. LoCO has an open and modular design that can be expanded to provide additional capabilities. Thus, LoCO allows us to add extra sensors as needed for our research while lack of modularity becomes a constraint with other proprietary AUVs such as Minnebot, which we have access to. The other is a data collection device called HydroEye, which we design to reduce the overhead of robotic deployments and assist a scuba diver in collecting visual and other sensory data. The device is portable and provides access to the collected data outside the water via Wi-Fi without disassembling the device.

Appendices

This section consists of three parts. Appendix A provides additional figures for Chapter 7 to show the localization performance of our proposed algorithms on eight lakes in Minnesota. Appendix B presents present a multi-view vision system for the detection and 3D localization of corn stalks in mid-season corn rows, called *ROW SLAM* [32]. I have collaborated with a Ph.D. student Jiacheng Yuan and Professor Volkan Isler for this work, where I have extended my research in the agricultural domain. Finally, Appendix C introduces a mapping approach that unifies semantic information and shape completion inferred from RGBD images and computes confidence scores for its predictions. This work [33] has been completed during my internship at the Samsung AI Center New York, which has enabled me to develop my research into the manipulation robotics field.

Part I: Visual Perception in Data-Scarce and Visually Challenging Underwater Environments

A robot's capability to perceive its surrounding is critical to being autonomous. However, detecting objects underwater poses substantial perception challenges to robots. Underwater vision is degraded by three main phenomena: (1) color absorption occurs as depth increases, (2) light scattering blurs underwater images, and (3) light refraction causes objects to appear larger. Most underwater debris is asymmetrical and deforms over time, which worsens the problem. Collectively, the research presented in Part I presents significant first steps toward overcoming perceptual problems in challenging environments. Chapter 2 presents the first deep learning-based approach [29] to detect debris with AUVs by overcoming these challenges. To improve the detection performance further, Chapter 3 introduces TrashCan [30], which is the first publicly available underwater image dataset and consists of $7,212$ pixel-level annotated images of debris, robots, and a wide variety of undersea flora and fauna. However, building such a dataset requires a labor-intensive process, and collecting imagery of underwater debris is inherently difficult and dangerous to divers. To address these risk and labor issues related to the data collection process, we present a generative approach [31] in Chapter 4, producing realistic fake debris images to augment existing datasets using a two-stage variational autoencoder. Chapter 5 takes one step further from augmenting existing datasets by presenting an image blending algorithm that generates images and corresponding annotations simultaneously. This eliminates labor-intensive dataset labeling efforts for creating datasets to train object detection models.

Chapter 2

Object Detection for Underwater Trash

One of the core capabilities for AUVs to be autonomous is perceiving objects they encounter. In this chapter, we focus on enabling AUVs to detect underwater debris. We consider a number of deep learning-based visual object detection algorithms; build a dataset, *Trash-ICRA19*, to train and evaluate them on; and compare their performance in the debris detection task with several metrics.

2.1 Related Work

Underwater autonomous robotics has been growing rapidly in significance and number of applications. Some research has focused on developing multi-robot systems for exploration of natural marine environments, both on the surface and underwater. Learning-based detection of underwater biological events for marine surveillance has also been explored [34]. Additionally, underwater robots have been used in multi-robot systems for environmental surveillance (*e.g.*, [35]), environmental mapping (*e.g.*, [36]), marine robotics for navigation and localization (*e.g.*, [37, 38, 39, 40, 41]), and more.

There have been a number of efforts to identify the location of trash and marine litter and analyze their dispersal patterns in the open ocean, particularly in the presence of eddies and other strong ocean currents. Mace looked into at-sea detection of marine debris [42] where he discussed possible applications of strategies and technologies. He points out that one of the major challenges is the debris being small, partially submerged, and buried in the sea bed. His focus is thus narrowed down to the floating debris on water parcel boundaries and eddy lines. Howell et al. [43] make a similar assessment, with their work investigating marine debris, particularly derelict fishing gear, in the North Pacific Ocean. Researchers have looked at the removal of debris from the ocean surface after the tsunami off the Japanese coast in 2011 [44]. The work by Ge et al. [45] uses LIDAR to find and map trash on beaches. Autonomous trash detection and pickup for terrestrial environments have also been investigated; for example, by Kulkarni et al. [46]. While this particular work is applied for indoor trash and uses ultrasonic sensors, a vision-based system can also be imagined.

Recent work by Valdenegro-Toro [47] has looked into using forward-looking sonar (FLS) imagery to detect underwater debris by training a deep convolutional neural network (CNN) and has been demonstrated to work with approximately 80 *percent* accuracy. However, this work uses an in-house dataset constructed by placing objects that are commonly found with marine litter into a water tank and taking FLS captures in the tank. The evaluation was also performed on water tank data. This work demonstrates the applicability of CNNs and other deep models to the detection of small marine debris, but it is unknown whether it will be applicable to natural marine environments.

Some debris detection processes use remotely operated vehicles underwater (ROVs) to record sightings of debris and manually operate a gripper arm to remove items of interest. The FRED (Floating Robot for Eliminating Debris) vehicles have been proposed by Clear Blue Sea [48], a non-profit organization for environmental protection and cleanup; however, the FRED platform is not an UAV. Another non-profit organization, the Rozalia project, has used underwater ROVs equipped with multibeam and side-scan sonars to locate trash in ocean environments [49].

To enable visual or sensory detection of underwater trash using a deep-learned appearance model, a large annotated dataset of underwater debris is needed. For capturing the varied appearances across wide geographical regions, this dataset needs to include data collected from a large spread of diverse underwater environments. Fortunately, some of these datasets exist, although most are not annotated for deep learning purposes. The Monterey Bay Aquarium Research Institute (MBARI) has collected a dataset over 22 years to survey trash littered across the sea bed off the western seaboard of the United States of America [50], specifically plastic and metal inside and around the undersea Monterey Canyon, which serves to trap and transport the debris in the deep oceans. Another such example is the work of the Global Oceanographic Data Center, part of the Japan Agency for Marine Earth Science and Technology (JAMSTEC). JAMSTEC has made a dataset of deep- sea debris available online as part of the larger J-EDI (JAMSTEC E-Library of Deep-sea Images) dataset [51]. This dataset has images dating back to 1982 and provides type-specific debris data in the form of short video clips. The work presented in this paper has benefited from annotating this data to create deep learning-based models.

2.2 Data Source and Training Set Construction

This work evaluates four deep learning architectures for underwater trash detection. To evaluate these four networks, we construct a first of its kind dataset, *Trash-ICRA19* [52], for training and evaluation, define our data model, and annotate images for training.

2.2.1 The Trash-ICRA19 Dataset

The dataset for this work was sourced from the J-EDI dataset of marine debris which is described in detail in the previous section. The videos that comprise that dataset vary greatly in quality, depth, objects in scenes, and the cameras used. They contain images of many different types of marine debris, captured from real-world environments, giving us a variety of objects in different states of decay, occlusion, and overgrowth. Additionally, the clarity of the water and quality of the light vary significantly from video to video. This allows us to create a dataset for training which closely conforms to real-world conditions, unlike previous contributions, which mostly rely on internally generated datasets.

Our training data was drawn from videos labeled as containing debris, between the years of 2000 and 2017. From that portion of data, we further selected all videos which appeared to contain some kind of plastic. This was done in part to reduce the problem to a manageable size for our purposes, but also because plastic is an important type of marine debris [53]. At this point, every video was sampled at a rate of three frames *per*

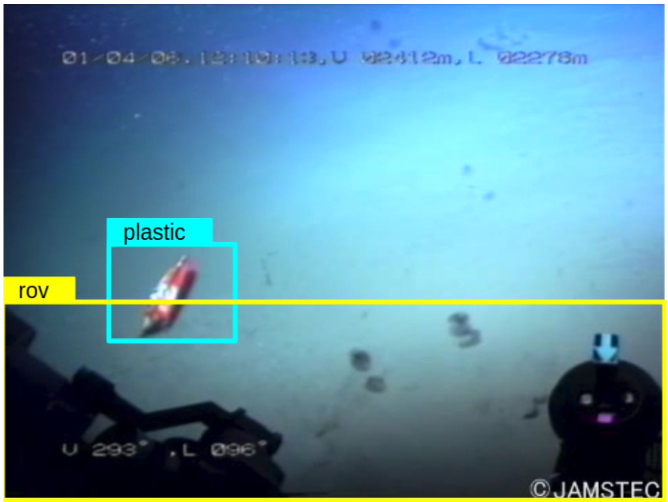


Figure 2.1: An example of a labeled training image showing two classes: the 'plastic' class in blue, and the 'ROV' class in yellow.

second to produce images which could be annotated to prepare them for use in learning models. This sampling produced over 240, 000 frames, which were searched manually for the good examples of plastic marine debris and then annotated. The annotating process was completed by a number of volunteers, who used the freely available LabelImg [54] tool. The final training dataset used in this work was composed of 5, 720 images, with dimensions of 480x320. More images have been labeled since, and a fully labeled dataset will be publicly released in the near future.

2.2.2 Data Model

Our trash detection data model has three classes, defined as follows:

- **Plastic:** Marine debris, all plastic materials.
- **ROV:** All man-made objects(*i.e.*, ROV, permanent sensors, etc), intentionally placed in the environment.
- **Bio:** All natural biological material, including fish, plants, and biological detritus.

The model is designed to find all plastic debris, separating debris from biological entities and intentionally placed man-made objects. These two distinctions are important, as accidental removal of biological entities such as plants or animals could damage the very ecosystems we hope to protect, and the destruction of intentionally placed parts such as a sensor array could set back cleanup efforts significantly.

Other versions of this data model were tested: one including a label for timestamp text on the screen (rejected due to lower accuracy) and models with multiple classes for different plastic objects (rejected for lack of data on some classes and lower detection performance overall). Both versions achieved lower accuracy than this model on all networks. Examples of labeled classes in images from this data model can be seen in Fig. 2.1.

2.2.3 Evaluation Data

To evaluate the models, test images which contained examples of every class in our model were selected for a test set. However, these images were drawn from videos of objects which had not been used to train the network, so there is no overlap between the training and test set. Different types of plastic objects were selected, such as grocery bags, plastic bottles, etc. At least three different videos and a minimum of 20 images per class were collected for each of these types of objects. In the end, 820 such images were collected and annotated for testing. These images contained examples for each class, in a variety of environments, which were intentionally selected to be challenging to the debris detectors, so as to provide a realistic evaluation of how these detectors would perform in field conditions.

2.3 Network Architectures

The four network architectures selected for this project were chosen from the most popular and successful object detection networks in use today. Each one has its own benefits and drawbacks, with varying levels of accuracy and runtime speeds.

2.3.1 YOLOv2

You Only Look Once (YOLO) v2 [55] is the improved version of YOLO, an earlier network by the same authors. Although YOLO is much faster than Fast R-CNN, Faster R-CNN, and other models, it has notable localization errors and low recall in comparison with other state-of-the-art object detection algorithms. With several techniques such as batch normalization, a higher resolution for input images, and a change in the way bounding boxes are proposed to use anchor boxes, YOLOv2 improves upon the accuracy of its predecessor. In order to process faster, YOLOv2 uses a custom network called Darknet-19 rather than VGG-16, which is used by many object detection algorithms. Darknet-19 requires only about one-sixth of the floating point operations of VGG-16 for a single pass of an input image.

2.3.2 Tiny-YOLO

Tiny YOLO [56] was introduced along with YOLO. Tiny-YOLO is based on the neural network that has a smaller number of convolutional layers and filters inside the layers compared to the network used by YOLO while sharing the same parameters with YOLO for training and testing. In this way, tiny-YOLO runs at even faster speeds than YOLO while having similar mean Average Precision (mAP) values. Tiny-YOLO is especially useful to implement object detection for applications where computational power is limited (*e.g.*, for energy consumption reasons) such as field robotics.

2.3.3 Faster RCNN with Inception v2

Faster RCNN [57] is an improvement on R-CNN [58] that introduces a Region Proposal Network (RPN) to make the whole object detection network end-to-end trainable. The RPN uses the last convolutional feature map to produce region proposals, which is then fed to the fully connected layers for the final detection. The original implementation uses VGG-16 [59] for feature extraction; we use the Inception v2 [60] as feature extractor instead because it is known to produce better object detection performance in standard datasets [61].

2.3.4 Single Shot MultiBox Detector (SSD) with MobileNet v2

Single Shot MultiBox Detector (SSD) [62] is another object detection model that performs object localization and classification in a single forward pass of the network. The basic architectural difference of SSD with YOLO is that it introduces additional convolutional layers to the end of a base network which results in improved performance. In our implementation, we use MobileNet v2 [63] as the base network.

2.4 Training

All four networks were trained according to the methods suggested by the community around the networks [61, 64], using on a Linux machine, using four Nvidia GTX 1080s. In general, this involved fine-tuning the networks with some pretrained weights which had been trained on the COCO dataset [65], a large-scale detection dataset. There were 5,720 images in our training set for each network, resized to 416x416 in the case of YOLOv2 and Tiny-YOLO. For RCNN, SSD, and Tiny-YOLO, we simply fine-tuned each network for a few thousand iterations.

However, in the case of YOLO, the network was not only fine-tuned, but also trained using a transfer learning. Transfer learning [66] is a technique used to improve learning in a new task by transferring knowledge from related tasks. The goal of transfer learning is to expedite the learning process by utilizing prior knowledge from one domain to new related domains. There are several benefits of applying transfer learning to this problem. First, the chance of the network properly converging for a new task may be better. Instead of training from a base, it uses the transferred knowledge to set initial training parameters. This may prevent a model from becoming trapped in local minima. Second, learning time to find parameters of a network may be reduced compared to learning from a base. Lastly, the final inference may have better performance than learning without transferred knowledge. In this study, we froze all but the last several layers so that only the last few layers had their weights updated during the training process. In this way, we attempt to capitalize on the fact that earlier layers' pretrained weights already encode for basic image features, and simply train the last few layers of YOLOv2. We report the results of these transfer-learned versions separately from the RCNN, SSD, and Tiny-YOLO results, to explore the possibility of using transfer learning to overcome deficiencies in a dataset.

2.5 Evaluation

2.5.1 Metrics

The four different networks used for the marine debris detector were evaluated using the two standard performance metrics below:

- mAP (Mean Average Precision) is the average of precision at different recall values. Precision is the ratio $precision = \frac{TP}{TP+FP}$, and recall is the ratio $recall = \frac{TP}{TP+FN}$, where TP, FP, and FN stand for True Positive, False Positive, and False Negative.
- IoU (Intersection Over Union) is a measure of how well predicted bounding boxes fit the location of an object, defined as $IoU = \frac{\text{area of intesection}}{\text{area of union}}$, where the intersection and union referred to are the intersection and union of the true and predicted bounding boxes.

These two metrics concisely describe the accuracy and quality of object detections.

2.5.2 Hardware

Additionally, runtime performance metrics are provided on a GPU (Nvidia 1080), embedded GPU (Nvidia Jetson TX2), and CPU(an Intel i3-6100U) in terms of the number of frames which can be processed per second (FPS). The hardware used for evaluation is only relevant to the runtime. Performance metrics, such as mAP, IoU, and recall, can be calculated identically on any device. The devices were selected to provide an indication of how these models could be expected to perform in an offline manner (GPU), in real-time on a computationally powerful robotic platform (embedded GPU), and in a lower-power robotic platform (CPU).

2.6 Results

The results below were obtained on the test dataset described in subsection 2.2.3, which was designed to be challenging to the models and to provide a true-to-life representation of how a marine debris detector would have to operate. The videos from which the test data is sourced are mostly from 10 years or more before the time of videos in the

Table 2.1: Detection metrics in mAP, IoU, and AP.

Network	mAP	Avg. IoU	plastic AP	bio AP	rov AP
YOLOv2	47.9	54.7	**82.3**	9.5	52.1
Tiny-YOLO	31.6	49.8	**70.3**	4.2	20.5
Faster R-CNN	81.0	60.6	**83.3**	73.2	71.3
SSD	67.4	53.0	**69.8**	6.2	55.9

Table 2.2: Performance metrics in frames per second.

Network	1080	TX2	CPU
YOLOv2	74	6.2	0.11
Tiny-YOLO	**205**	**20.5**	0.52
Faster R-CNN	18.75	5.66	0.97
SSD	25.2	11.25	**3.19**

Table 2.3: Detection metrics for different training methods for YOLO.

YOLO Training	mAP	Avg. IoU	plastic AP	bio AP	rov AP
Fine Tuned	47.9	54.7	**82.3**	9.5	52.1
Last 4 Layers	33.9	45.4	**71.3**	13.6	17.0
Last 3 Layers	39.5	34.1	**74.6**	19.9	23.9

training set; they differ from each other and the training set in location, depth, and the camera used to capture the scene. Because of these factors, the test set provides a good indicator of how these detectors would perform across different platforms over the course of many years and through many environmental changes. This does degrade the performance of the detector from what could have been achieved if data with similar appearance was chosen, but it is a better evaluation of what the detector's performance would be in the field. Along with evaluating the data on our test set, we also processed three additional videos, with each network, which can be seen in Fig. 2.2 as well as in the video submitted with this paper. The third video, which is shown in Figs. 2.2i - 2.2l, is a viral video of ocean trash recorded in Bali [1]. Being from an entirely different visual environment than our training and test data, it challenges all of our detectors greatly, but some promising detections can be seen over the course of the video.

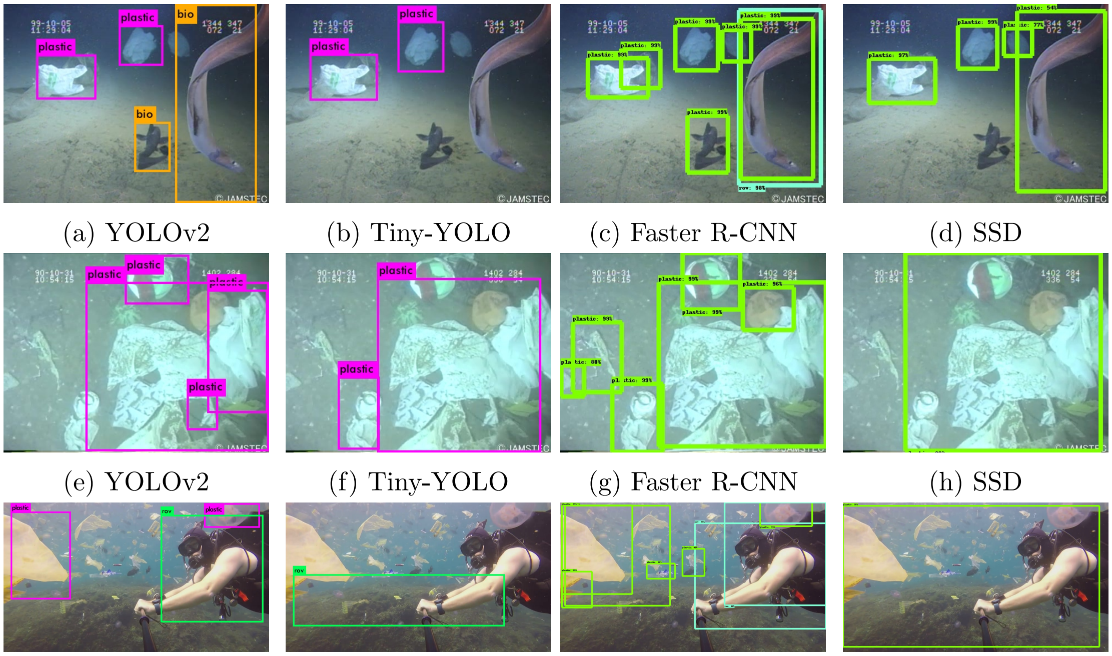
2.6.1 Quantitative Results

The results shown in Table 2.1 are encouraging. The networks tested have fairly good average precision for the plastic class (similar to the accuracy of the FLS imagery based approach in [47]). They also display some known traits of the network architectures used. Overall, YOLOv2 and Tiny-YOLO have lower mAP when compared to Faster R-CNN and SSD. Conversely, Faster R-CNN and SSD have higher processing times, as seen in Table 2.2. These traits are well known, but it is important to note that their performance remain consistent in this application. This trade-off between mAP and FPS does not affect IoU, however. All four network architectures have similar IoU values, meaning that none are the clear victor in terms of how accurate their bounding boxes are.

In terms of which method would be the ideal for underwater trash detection, Faster R-CNN is the obvious choice purely from a standpoint of accuracy, but falters in terms of inference time. YOLOV2 strikes a good balance of accuracy and speed, while SSD provides the best inference times on CPU. If performance is the primary consideration, however, Tiny-YOLO outpaces all other algorithms significantly on the TX2, the most realistic hardware for a modern AUV.

We can also see the results of including classes with too few examples, particularly in the case of the bio class, which has many fewer examples in the training set, despite being at least as varied as the plastic trash in the underwater environment. The fact that classes with few training images relative to their complexity will perform well is not surprising in any way, but highlights some of the difficulties inherent in creating a dataset for trash detection: some objects may only be encountered a handful of times but should still count as trash, or be marked as biological and therefore not be removed.

It is worth noting that in the case of the bio class, the versions of YOLO trained using transfer learning are significantly more accurate. We believe this to be related to the types of objects that the pre-trained weights were trained to detect and the preponderance of plastic objects in the dataset. Simply put, by not updating those earlier layers, we avoid skewing the basic image features towards plastic objects, reducing plastic detection accuracy slightly which increasing that of the bio class.



(a) YOLOv2 (b) Tiny-YOLO (c) Faster R-CNN (d) SSD

(e) YOLOv2 (f) Tiny-YOLO (g) Faster R-CNN (h) SSD

(i) YOLOv2 (j) Tiny-YOLO (k) Faster R-CNN (l) SSD

Figure 2.2: Example detection results. In images 2.2a-2.2d it can be seen that while all networks detect some plastic objects, only YOLOv2 (2.2a) correctly identifies the fish in the scene as bio and Faster-RCNN (2.2c) detects more individual plastic items than any other network. In images 2.2e-2.2h, the same evidence of Faster-RCNN's (2.2g) ability to detect more individual objects can be seen. Images 2.2i-2.2l are from [1], and display how the detectors work on data drawn from a visually different source. YOLOv2 (2.2i) and Faster-RCNN (2.2k) both successfully find plastic and ROV objects, but no detectors perform sufficiently.

2.6.2 Qualitative Results

The most substantial result from this evaluation is the answer to the question we set out to investigate: can we use deep-learning based visual object detection methods to detect marine debris in real-time? The answer, borne out by our detection accuracy and performance results is, yes, plausibly. There are some difficulties in constructing a representative dataset, but overall our detection results lead us to believe that detectors similar to these could be used to find trash underwater with high enough accuracy to be useful. Moreover, the observed performance, especially on the Nvidia Jetson TX2, is very encouraging for the prospect of a real-time application. The Jetson is small enough to conceivably be used in an AUV, and although it increases the electrical and thermal load of the system, we believe the increase to be manageable, with the Jeston only requiring 15 watts [67] and producing minimal heat. This, combined with its performance in tasks such as these, makes the Jetson ideal for mobile robotics, even for our specific use case. With a Jetson installed, any AUV should be able to achieve real-time trash detections with reasonable accuracy.

2.7 Conclusion

In this chapter, we evaluate a number of deep-learning algorithms performing the task of visually detecting trash in realistic underwater environments, with the eventual goal of exploration, mapping, and extraction of such debris by using AUVs. A large and publicly-available dataset of actual debris in open-water locations is annotated for training a number of convolutional neural network architectures for object detection. The trained networks are then evaluated on a set of images from other portions of that dataset, providing insight into approaches for developing the detection capabilities of an AUV for underwater trash removal. In addition, the evaluation is performed on three different platforms of varying processing power, which serves to assess these algorithms' fitness for real-time applications. While our research in this chapter makes progress toward AUVs being able to detect underwater debris, these algorithms show degraded detection accuracy. To address this limitation, a larger size dataset is needed. We introduce our approach to this problem in the next chapter.

Chapter 3

A Semantically-Segmented Dataset towards Visual Detection of Underwater Debris

Data scarcity in terms of volume and variety remains a primary issue in the tasks of visual detection of underwater debris. To address this issue, we introduce an instance-segmentation labeled dataset, *TrashCan*, in this chapter. The proposed dataset and our baseline results are presented in detail in the following sections.

3.1 Motivation, Problem Statement, and Related Work

Marine debris poses a significant threat to the aquatic ecosystem and has been an ever-increasing challenge to tackle. Different environmental and government agencies have proposed and adopted a number of approaches for the cleanup of underwater trash, though few have been broadly successful. Moreover, the difficulty of the cleanup task increases many folds because of the need to precisely detect trash deposits and locate them, and subsequently remove such trash while protecting underwater flora and fauna. Vision-equipped autonomous underwater vehicles (AUVs) could be used to address these challenges. Along with underwater visual localization algorithms [68], a robust trash detection feature will provide AUVs with the capability to both detect and locate trash, and possibly remove it themselves or make it easier for manned cleanup missions. A precise *underwater* trash detector, however, would need a significant dataset to train deep neural nets (*e.g.*, CNNs) for visually detecting debris which are often deformed and take non-rigid shapes. Unfortunately, no such datasets have been made available to the public.

This paper presents *TrashCan*, a large dataset comprised of images of underwater trash collected from a variety of sources [51], annotated both using bounding boxes and segmentation labels, for the development of robust detectors of marine debris. The dataset has two versions, TrashCan-Material and TrashCan-Instance, corresponding to different object class configurations. The eventual goal is to develop efficient and accurate trash detection methods suitable for onboard robot deployment. Along with information about the construction and sourcing of the TrashCan dataset, we present initial results of instance segmentation from Mask R-CNN [69] and object detection from Faster R-CNN [57]. These do not represent the best possible detection results but provide an initial baseline for future work in instance segmentation and object detection on the TrashCan dataset.

While the topic of removing marine debris from surface and subsea environments has been a topic of scientific inquiry for some time, autonomous detection of trash has been less studied. Researchers have studied at the removal of debris from the ocean surface [44], mapping trash on beaches using LIDAR [45], and using forward-looking sonar (FLS) imagery to detect underwater debris by training a deep convolutional neural network (CNN) [47]. More recently, we presented an initial evaluation of state-of-the-art object detection algorithms for trash detection, using a predecessor of TrashCan [70], and showed how such datasets could be effectively improved with data produced by generative models [31]. *TrashCan* is the next step for the autonomous detection and removal of trash: a larger, more detailed, and more varied dataset of marine debris for training deep neural networks for object detection.

3.2 The TrashCan Dataset

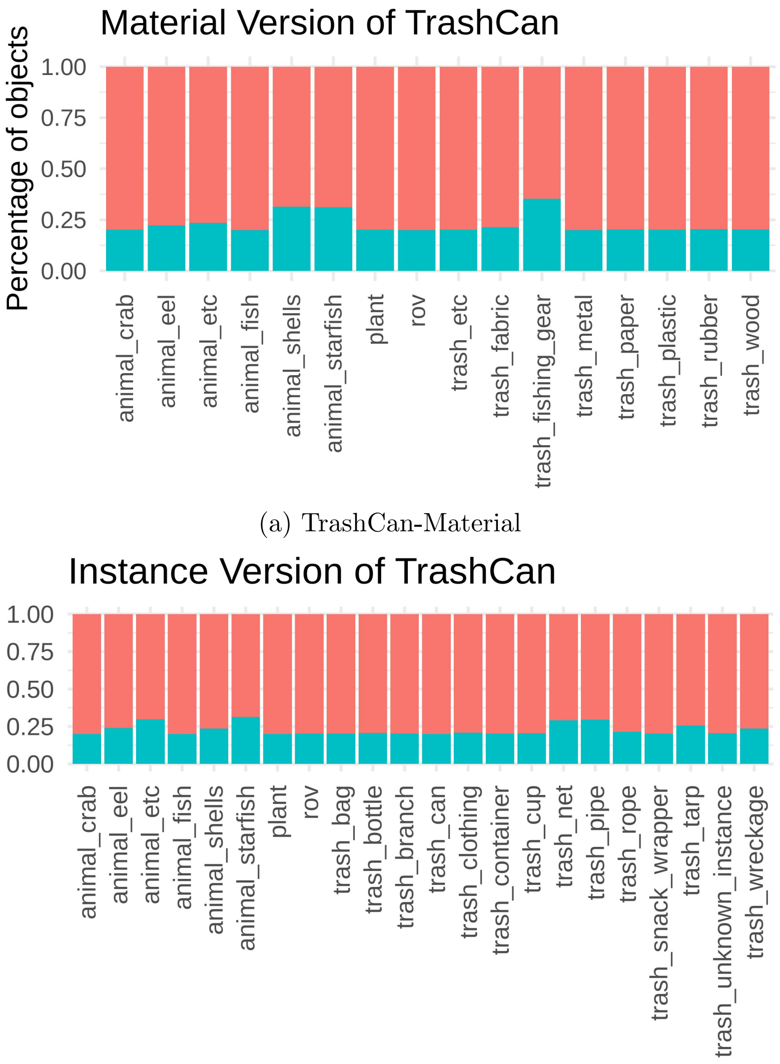
The TrashCan dataset is comprised of annotated images ($7,212$ images currently) which contain observations of trash, ROVs, and a wide variety of undersea flora and fauna.

The annotations in this dataset take the format of instance segmentation annotations: bitmaps containing a mask marking which pixels in the image contain each object. While datasets have previously been created containing bounding box level annotations of trash in marine environments, including one of our own creation [70], TrashCan is, to the best of our knowledge, the first instance-segmentation annotated dataset of underwater trash.

Dataset Source and Composition: The imagery in TrashCan is sourced from the J-EDI (JAMSTEC E-Library of Deep-sea Images) dataset [51], curated by the Japan Agency of Marine Earth Science and Technology (JAMSTEC). This dataset contains videos from ROVs operated by JAMSTEC since 1982, largely in the Sea of Japan. A small portion of these videos contain observations of marine debris, and it is from these that all of our trash data is sourced, nearly one thousand videos of varying lengths. In addition to the videos of marine debris, additional videos were selected to diversify the biological objects present in the dataset.

Annotation Process and Tools: Once the videos had been selected, they were preprocessed by extracting frames from each video at a rate of one frame per second to create a large folder of frames for each video. Once this was done, the videos were combined into similarly-sized portions and uploaded to Supervisely [71], an online image annotation tool. Once uploaded, the images were annotated by a team of 21 people, one image at a time. This took approximately $1,500$ work hours, over the course of several months. if an image was considered acceptable for labeling, the person to whom it was assigned drew a segmentation mask over it, marking it as one of four classes: *trash* (any marine debris), *rov* (any man-made item intentionally placed in the scene), *bio* (plants and animals), and *unknown* (used to mark unknown objects). Trash objects were additionally tagged by material (*e.g.*, metal, plastic), instance (*e.g.*, cup, bag, container), along with binary tags indicating overgrowth, significant decay, or crushed/broken items. Bio objects were tagged as either plant or animal, and in the case of animals, given a tag with the type (*e.g.*, crab, fish, eel). ROV and unknown class objects required no additional tags.

Object Class Versions: To prepare the dataset for use in training deep networks, it was converted from a custom JSON format to the COCO format [65]. This involved converting the bitmap masks into COCO annotations comprised of polygon vertices



(a) TrashCan-Material

(b) TrashCan-Instance

Figure 3.1: Data split between training (pink) and validation (blue) sets per object for the two versions of the dataset.

and transforming the classes and tags of Supervisely annotations into classes for COCO objects. We converted all objects into one of two dataset versions: TrashCan-Material and TrashCan-Instance (Fig. 3.1), so named for the tag data used to differentiate between different types of trash. In the material version, every trash object was given a class name following the pattern *trash_[material_name]* (*e.g.*, *trash_paper*, *trash_plastic*), as long as the given material had more than 50 objects in the dataset. Those with fewer examples were given the class *trash_etc*, the same class used by annotators when the material of the object was unknown. Similarly, for the TrashCan-Instance version, trash classes were generated using instance tags which approximated the type of object that was being annotated (*e.g.*, *trash_cup*, *trash_bag*). The same cutoff of 50 objects was used, with the catch-all class being *trash_unknown_instance*. In both versions, any object labeled with the unknown class was typically added to the catch-all trash class. The ROV class remained singular for both versions, while bio objects were either turned into *plant* or *animal_[animal_type]* (*e.g.*, *animal_starfish*, *animal_crab*) classes, based on tags applied to the object.

3.3 Baseline Experiments

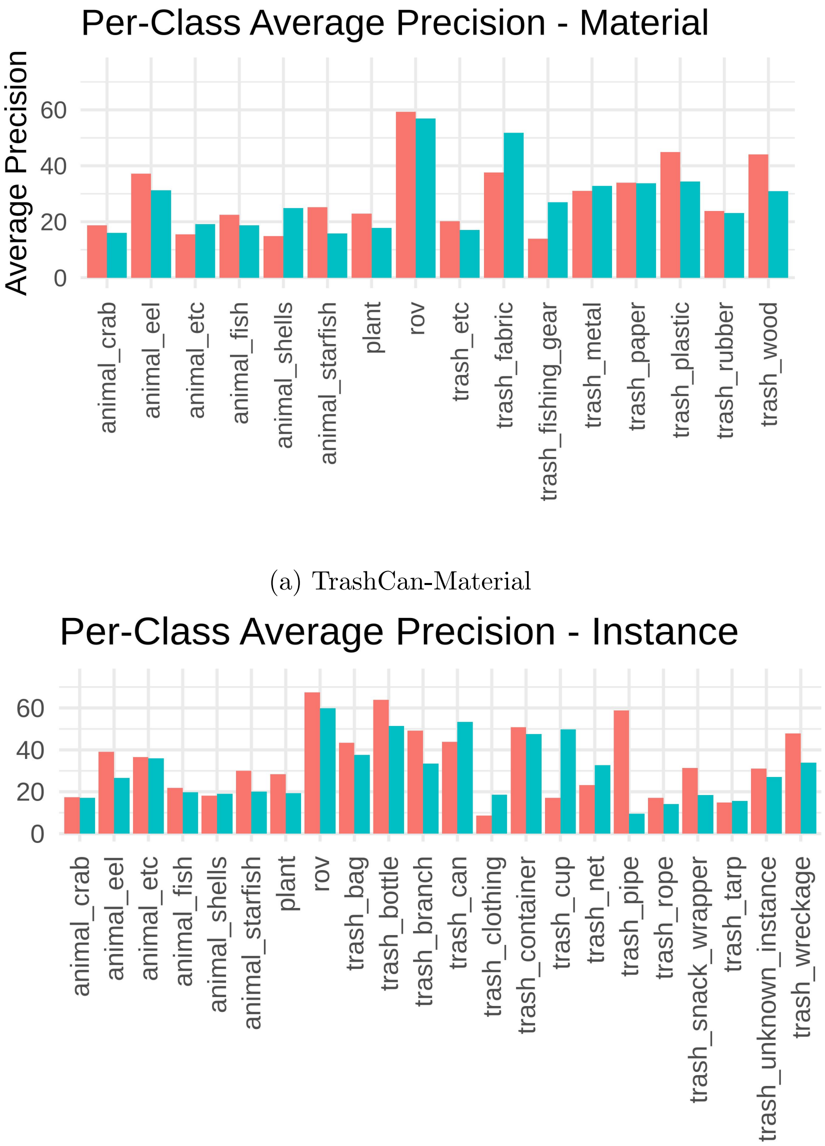
We present experiments with state-of-the-art instance segmentation and object detection models using two example datasets to provide a baseline for future model development. For the following experiments, we use the Pytorch Detectron2 [72] library and metrics introduced in COCO dataset to establish a baseline evaluation.

Detection Experiments: For object detection, we employed Faster R-CNN with a ResNeXt-101-FPN (X-101-FPN) [73] backbone. The model was trained on a pre-trained model with COCO dataset with an Nvidia Titan XP for $10,000$ iterations.

Segmentation Experiments: Mask R-CNN with X-101-FPN was chosen for instance segmentation task. The model was initialized with weights from the COCO dataset and trained for $15,000$ iterations using an Nvidia Titan V.

Table 3.1: Overall metrics for each combination of dataset and model.

Method	Dataset	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN	Instance	30.0	55.3	29.4	23.2	31.7	48.6
Mask R-CNN	Material	28.2	54.0	25.6	24.1	28.7	41.8
Faster R-CNN	Instance	34.5	55.4	38.1	27.6	36.2	51.4
Faster R-CNN	Material	29.1	51.2	27.8	28.2	30.2	40.0



(a) TrashCan-Material

(b) TrashCan-Instance

Figure 3.2: Results from Faster R-CNN (pink) and Mask R-CNN (blue) in terms of per-class average precision.

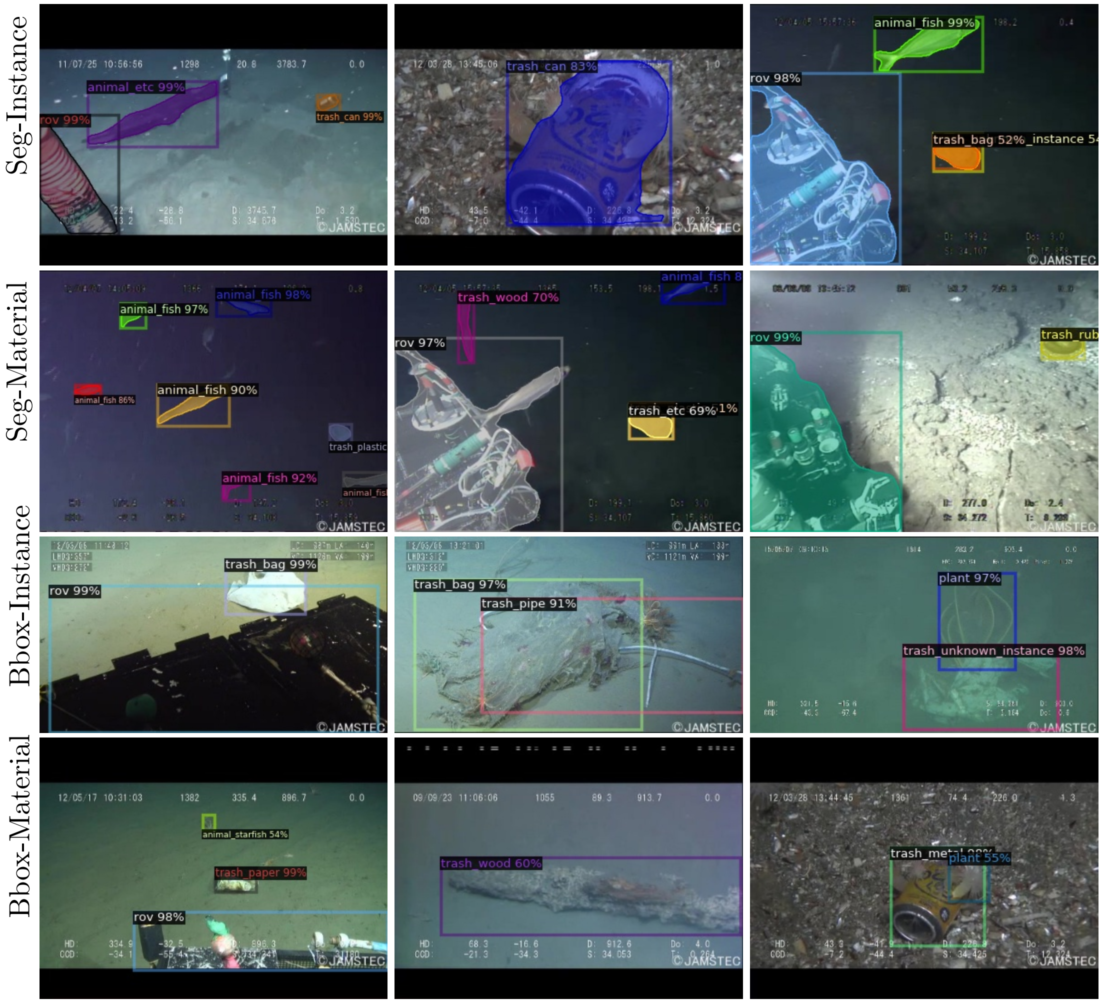


Figure 3.3: Sampled results for object detection and image segmentation for both versions of the TrashCan dataset.

3.4 Results

Fig. 3.2 and Table 3.1 show evaluation metrics of object detection and instance segmentation with each dataset. For both object detection and instance segmentation tasks, the models trained with the instance version dataset achieve higher AP in general. We believe this is because more visually similar objects are grouped into classes: while most cans look similar, not all metal objects look similar. Fig. 3.3 displays sampled results from object detection and instance segmentation models trained with both versions of the datasets. The results include a wide spectrum of object sizes and environments. Although the baseline metrics are acceptable considering the challenging nature of this dataset, there is room for improvement in future work, either by increasing the size of the dataset or by employing more advanced models.

3.5 Conclusion

In this chapter, we present our methods to overcome the data scarcity issue for underwater debris detection. To tackle this, we introduce TrashCan, which provides the first instance-segmented trash dataset in existence, along with semantic information about the material and type of the object. It is our hope that the release of TrashCan will facilitate further research on this challenging problem, bringing the marine robotics community closer to a solution for the urgent problem of trash detection and removal. In the next chapter, we present a generative approach, TrashVAE, for augmenting existing datasets with realistic images.

Chapter 4

A Generative Approach for Data Augmentation

Collecting imagery of marine debris is not just time-consuming but costly and physically demanding. To resolve this issue, we propose *TrashVAE* in this chapter, which is a method composed of a two-stage variational autoencoder (VAE) and binary classifier for producing realistic synthetic underwater debris images. The approach and evaluations are presented in detail in the following sections.

4.1 Related Work

Various generative models [74] have been proposed to generate realistic imagery, and they are used for data augmentation in the non-underwater domain [75] to improve object classification and detection tasks. Although there are generative model-based studies for the underwater domain, they mainly focus on image enhancements, such as WaterGAN [76], UGAN [77], and the synthesis of unpaired Underwater images using a Multistyle GAN (UMGAN) [78].

WaterGAN uses pairs of non-underwater images and depth maps to train its generative model. Then, it generates underwater images from above-water images. UGAN uses CycleGAN [79] to generate distorted images from images with no distortion, with the Wasserstein GAN [80] used to prevent the adversarial training process from destabilizing. UMGAN combines CycleGAN and Conditional GAN [81] to generate multistyle underwater images. Although UMGAN generates images with various turbidity and colors, output images are equivalent to the input images except for their colors and turbidity. With all these methods, it is challenging to realistically simulate color and shape distortions of various kinds of materials underwater since they have no effect on the objects which are present in the original image; these methods simply perform a broad domain-transfer technique to simulate underwater imagery.

It can be challenging to tune hyperparameters for GAN models, which aim to find the Nash equilibrium of a two player min-max problem [82, 83]. While VAE-based approaches generate blurrier images than GAN-based ones, those based on the VAE are more stable during training and less sensitive to hyperparameter choices. Among recent VAE enhancements, only the two-stage VAE [84] focuses on improving visual quality scores rather than improving log-likelihood scores. As a result, the two-stage VAE produces high-fidelity images which have visual quality scores close to the GAN enhancements while being much more stable during training.

As mentioned earlier in this section, the approaches adding various effects to the input images are limited in that they can only create images they have seen. There are currently no effective methods for solving the data scarcity problems in the underwater domain. Therefore, it is necessary to develop generative models which produce underwater images that are not dependent on input images.

4.2 Methodology

The proposed two-stage VAE model is trained with an existing dataset of underwater trash images (see Section 4.3 for details). After training, the model is used to generate synthetic underwater trash images, which are subsequently classified by hand (human observers) as “good” or “bad” based on their quality. This generated and classified data is then used to train an automated binary classifier. After training both the VAE and the binary classifier, we are able to generate and append good quality images to our dataset; the process is depicted in Fig. 4.1. Lastly, the effect of the dataset expansion is evaluated with a multi-class classifier trained to discriminate between trash and non-trash objects.

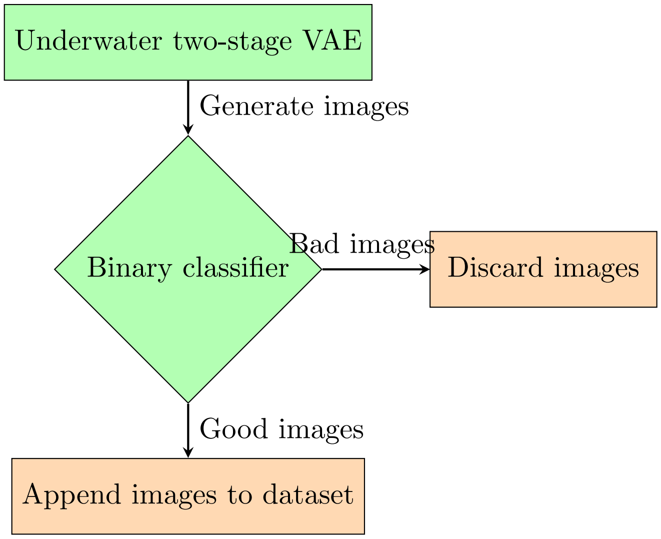


Figure 4.1: Flowchart showing image generation, filtering, and augmentation process for expanding an image dataset (green color represents trained models).

4.2.1 Data Collection

We collect and annotate images of plastic bags and plastic bottles since these are common objects which have a wide variety of shapes and frequently deform significantly over time underwater. Additionally, plastic makes up the lion's share of marine debris and has the most detrimental effect on the ecosystem. These factors make it challenging to build a truly representative dataset purely from observations. The images are obtained from the J-EDI (JAMSTEC E-Library of Deep-sea Images) dataset [51] and web

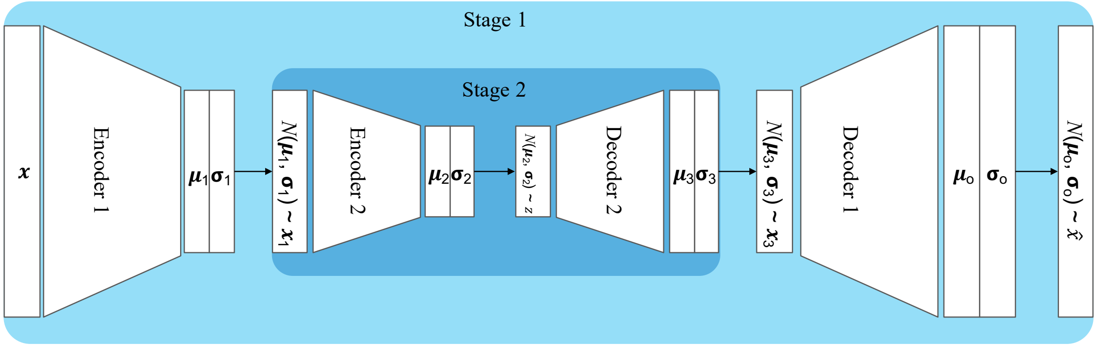

Stage 1
Stage 2
$\boldsymbol{x}$
Encoder 1
$\boldsymbol{\mu}_1$ $\boldsymbol{\sigma}_1$
$N(\boldsymbol{\mu}_1, \boldsymbol{\sigma}_1) \sim \boldsymbol{x}_1$
Encoder 2
$\boldsymbol{\mu}_2$ $\boldsymbol{\sigma}_2$
$N(\boldsymbol{\mu}_2, \boldsymbol{\sigma}_2) \sim z$
Decoder 2
$\boldsymbol{\mu}_3$ $\boldsymbol{\sigma}_3$
$N(\boldsymbol{\mu}_3, \boldsymbol{\sigma}_3) \sim \boldsymbol{x}_3$
Decoder 1
$\boldsymbol{\mu}_o$ $\boldsymbol{\sigma}_o$
$N(\boldsymbol{\mu}_o, \boldsymbol{\sigma}_o) \sim \hat{x}$

Figure 4.2: The architecture of the two-stage VAE.

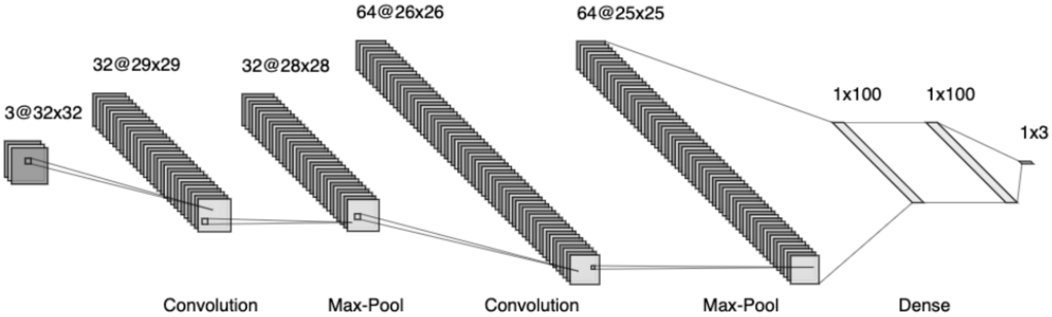


Figure 4.3: The architecture of the multi-class classifier.

scraping. These images are then cropped to have tight bounds around the contained objects as shown in Fig 4.5. The J-EDI dataset consists of images which have been collected since 1982 at various locations and depths. It is also sorted by types of trash. We collect plastic trash video clips from the J-EDI dataset, label objects in each clip, and build our own dataset, which will be released in the near future. Lastly, we add images from the web to bring more variety to our dataset.

4.2.2 Variational Autoencoder

The VAE aims to learn a generative model of input data $x \in \mathcal{X}$. The dataset $\mathcal{X}$ is d-dimensional and it has a r-dimensional manifold. The value of r is unknown. We use the term 'VAE' to denote a single-stage VAE in the rest of this paper.

Single-stage VAE (original VAE)

The VAE consists of two neural networks which are an encoder $q_\phi(z|x)$ and a decoder $p_\theta(x|z)$. The encoder outputs the parameters of the normal distribution, μ and σ. From $N(\mu, \sigma)$, the κ-dimensional latent variable z is sampled. Once the latent variable z is sampled and provided as an input to the decoder, it generates the reconstructed original input $\hat{x}$. The cost function for the VAE is shown in Eq. 4.1 where μ_{gt} is a ground-truth probability measure and $\int_{\mathcal{X}} \mu_{gt}(dx) = 1$. The function is minimized by stochastic gradient descent and the VAE jointly learns the latent variable and inference

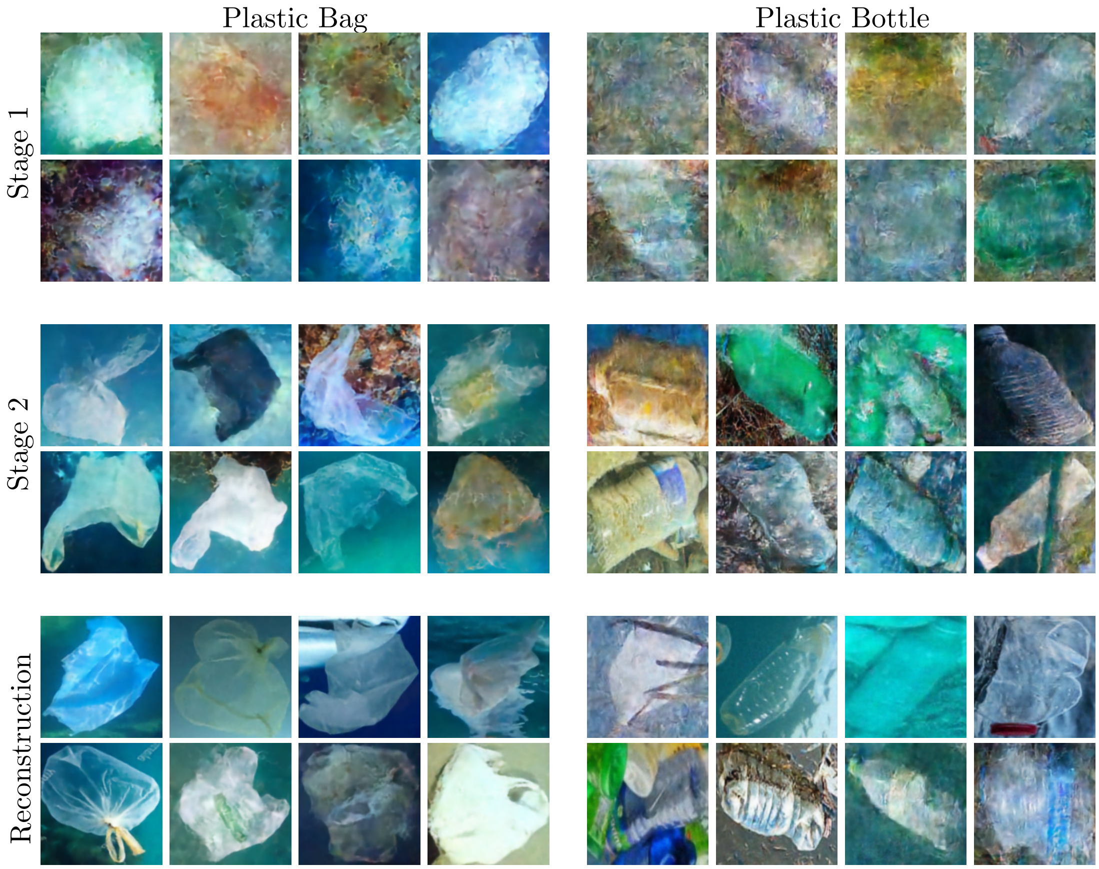


Figure 4.4: Sample generated images from the two-stage VAE for each trash class.

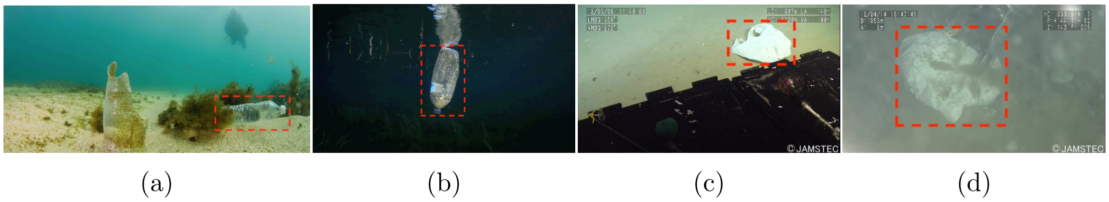
models during the training.

$$\mathcal{L}(\theta,\phi) = \int_{\mathcal{X}} \Big\{ -\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{KL}[q_\phi(z|x)||p(z)] \Big\} \mu_{gt}(dx) \tag{4.1}$$

Two-stage VAE

The VAE, compared to the GAN, (1) gives interpretable networks [85], and (2) remains stable during training [86]. However, the Gaussian assumption for the encoder and decoder of the VAE reduces the quality of generated images [87].

In [84], the authors show that there exist the parameters ϕ and θ which optimize the VAE objective with the Gaussian assumptions. Based on this finding, the authors show



(a) (b) (c) (d)

Figure 4.5: Examples of underwater trash images collected and annotated for training the proposed model. Images are collected from a variety of underwater conditions, and contain a variety of objects ranging from plastics to metals.

that the two-stage VAE finds the parameters ϕ and θ closer to the optimal solution than the parameters found by the VAE. The structure of the two-stage VAE is shown in Fig. 4.2. As explained in section 4.2.2, each encoder yields parameters for the normal distribution and the values sampled from the distribution are fed to each decoder to generate the reconstructed input.

The cost function in Eq. 4.1 is used without any modification for the two-stage VAE. Each stage of the two-stage VAE is trained sequentially from the first stage to the second stage, rather than jointly. We choose the ResNet structure [88] for the stage 1 VAE and select four fully connected (FC) layers for the stage 2 VAE. Each FC layer is 1024-dimensional in our model. The network for the stage 2 VAE is smaller than the stage 1 VAE since it is assumed that $d \gg \kappa \geq r$, where d is the dimension of the dataset $\mathcal{X}$, κ is the dimension of the latent variable z, and r is the dimension of the manifold.

We use the Fréchet Inception Distance (FID) score [89] to evaluate the quality of generated images. Lower FID scores represent higher image quality. It extracts features from generated images and real images using an intermediate layer of InceptionNet [90]. The data distribution of the features from generated images and real images are modeled separately using a multivariate Gaussian distribution. Then the FID score is calculated by Eq. 4.2 where μ is mean, and Σ is covariance.

$$\mathbf{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}}) \tag{4.2}$$

4.2.3 Binary Classifier

A binary classifier is built on ResNet-50 [88] to select good quality images from the generated images. The original input dimension of ResNet-50 is $(224, 224, 3)$, but we

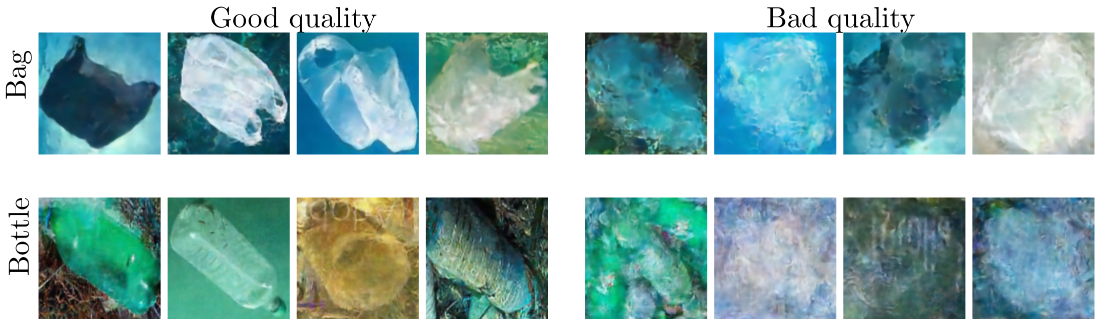


Figure 4.6: Sample outputs from the binary classifier. "Good quality" generated images of each class are used to expand the dataset whereas the "bad quality" images are rejected. Qualitatively, it is apparent that "bad quality" images are poor representations of objects of their class.

change it to $(128, 128, 3)$ to match the input dimension to the output dimension of the two-stage VAE model. The remaining structure of ResNet-50 is maintained.

Table 4.1: Generation and classification evaluation

(a) FID scores for the two-stage VAE

	Stage 1	Stage 2	Reconstruction
Bag	98	196	175
Bottle	223	301	240

(b) Accuracy for the binary classifier

	Training Acc.	Validation Acc.	Test Acc.
Bag	0.89	0.88	0.86
Bottle	0.88	0.83	0.82

4.2.4 Multi-class Classifier

The network for a multi-class classifier is designed to quantify the improvements to object classification tasks using generated images. The input is resized to $(32, 32, 3)$. It has two convolutional layers followed by one dense layer and one dropout layer. A softmax function is used as the activation function for the output layer and classifies three classes. One class is for our generated images, and the other two are for underwater background and fish. The architecture of the network is shown in Fig. 4.3.

We use three metrics to evaluate the performance of the multi-class classifier: precision, recall, and F1 score. Precision is useful if the cost of false positives is high. Recall is important when the cost of false negatives is high. Lastly, F1 score provides the balanced information between precision and recall [91]. In our case of classifying trash, the cost of false positives is relatively higher than the cost of false negatives. This is because the consequences of accidentally removing something from the environment that is not trash (*i.e.*, fish, coral, etc.) are much worse than the consequences of missing a single piece of trash.

4.3 Experiments and Results

We train our models on an NVIDIA Titan-class GPU with the Tensorflow library. One epoch takes 1-2 seconds for the stage 2 VAE, the binary classifier, and multi-class classifier. It takes 17-18 seconds for each epoch of the stage 1 VAE model.

4.3.1 Data Collection

From web scraping and the J-EDI dataset, we collect 775 images of plastic bags and 283 images of plastic bottles. We augment each class of images by flipping horizontally, vertically, and rotating them by 90 degrees. As a result, we have a total of $3,000$ images of plastic bags and $1,000$ images of plastic bottles. For training and testing the multi-class classifier, we use the QUT fish dataset [92] which has $4,405$ fish images and randomly select $3,000$ images. We also collect 271 images of underwater scenes without objects and augment them using the methods described above, resulting in $3,000$ images.

4.3.2 Image Generation

Two two-stage VAE models are trained separately, one for each class of trash. We use different parameters to train each model due to the significant difference in the size of the collected datasets. During the training process, we use the mean absolute error (MAE) [93] in Eq. 4.3 as a stopping criterion for each stage of the VAE in addition to the loss functions from the original VAE. The MAE is used since it is robust to outliers and helps to evaluate the training progress.

Table 4.2: Evaluation of object classification tasks: 3 multi-class classifiers are trained separately for plastic bag and bottle cases. In both cases, addition of generated data improves classifier performance in precision, recall, and F1 score (shown in bold entries) for the instance classes.

Plastic Bag												
	Real				Generated				Mixed			
	Precision	Recall	F1 score	Support	Precision	Recall	F1 score	Support	Precision	Recall	F1 score	Support
bag	0.96	0.62	0.76	300	**0.96**	**0.80**	**0.87**	300	0.95	0.78	0.86	300
fish	0.95	0.95	0.95	300	0.94	0.97	0.95	300	0.95	0.96	0.95	300
empty	0.74	0.99	0.84	300	0.87	0.99	0.93	300	0.85	0.99	0.92	300
avg/tot	0.88	0.86	0.85	900	0.92	0.92	0.92	900	0.92	0.91	0.91	900

Plastic Bottle												
	Real				Generated				Mixed			
	Precision	Recall	F1 score	Support	Precision	Recall	F1 score	Support	Precision	Recall	F1 score	Support
bottle	0.93	0.78	0.85	300	0.93	0.78	0.85	300	**0.97**	**0.80**	**0.87**	300
fish	0.89	0.94	0.91	300	0.83	0.96	0.89	300	0.91	0.96	0.93	300
empty	0.89	0.98	0.94	300	0.96	0.97	0.96	300	0.89	1.00	0.94	300
avg/tot	0.90	0.90	0.90	900	0.91	0.90	0.90	900	0.92	0.92	0.92	900

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \tag{4.3}$$

1. Plastic Bag: For the stage 1 VAE, we use ResNet with a base dimension of 16 and a kernel size of 3, training it for $3,000$ epochs. For the stage 2 VAE, four 1024-dimensional dense layers are used. The dimension of the latent variable is 12 and the batch size is 16. We train for $6,000$ epochs for the stage 2 VAE.

2. Plastic Bottle: For the stage 1 VAE, we use the same network and parameters as the model for the plastic bag class, also training for $3,000$ epochs. For the stage 2 VAE, four layers of 1024-dimensional dense layers are used. Both the dimension of the latent variable and batch size is 8. We train for $6,000$ epochs for the stage 2 VAE.

The samples of generated images are shown in Fig. 4.4, and Table 4.1a presents the FID scores for images generated from stage 1, stage 2, and reconstructed images. It is clear that the two-stage VAE improves the visual quality of generated images for both plastic bag and bottle classes. The images from stage 1 are nearly indistinguishable between classes. On the other hand, the images from stage 2 are significantly sharper in general. Although the two-stage VAE for the plastic bottle class outputs decent results, the generated plastic bag images are more crisp. This is also observed from the FID scores.

4.3.3 Binary Classification

The binary classifier employs the ResNet-50 architecture. We first generate $10,000$ images of both plastic bags and bottles. Then, we label images as either “good” or “bad” using the visual quality of each image. We then build two separate training datasets using $1,200$ good quality images and $1,200$ bad quality images, one dataset for plastic bags and one for plastic bottles. The batch size is set to 16 and the epochs for training the classifier was 50. The training results are shown in Table 4.1b. Similar to the outputs of the two-stage VAE, the binary classifier for the plastic bag class performs better overall. Lastly, the images classified as “good quality” are added to the existing dataset for each class.

4.3.4 Multi-class Classification

To quantify the impact of the generated images, three different types of datasets are created as follows. Each dataset contains $3,000$ images so only the compositions of the datasets are different among them.

1. Real dataset: only includes the real images from the original data collection.

2. Generated dataset: consists of the images which are generated from the two-stage VAE and filtered by the binary classifier.

3. Mixed dataset: consists of 50% images from the real dataset and 50% from the generated dataset.

For each class, 3 multi-class classifiers are trained separately with each dataset. The batch size is 100, and the epoch is 30 for all training processes. Test images are 300 real images for each dataset, and the images are not shown to the classifiers during the training processes. In total, 6 multi-classifiers are trained, and the results are shown in Table 4.2.

In both plastic bag and bottle classes, the classifiers trained with the generated dataset and the mixed dataset generally outperform the ones trained with the real dataset. Recall and F1 score are improved up to 18% and 11%, respectively, in the plastic bag case. Precision becomes 4% more accurate in the plastic bottle case. For the underwater trash detection problem, therefore, the results point to the viability of using the proposed approach to augment an existing yet small dataset with generated imagery using a two-stage VAE.

4.4 Conclusion

We present an approach to address data scarcity problems in underwater image datasets for visual detection of marine debris. The proposed approach relies on a two-stage VAE and a binary classifier to evaluate the generated imagery for quality and realism. Our results show that the generated data can be used to augment real datasets. This approach will not only be valid for the underwater debris problem presented in this paper, but it will also be useful for any data-dependent task for which collecting more images is infeasible or challenging at best. In the next chapter, we take one step further and introduce our method for generating synthetic images and corresponding annotations simultaneously.

Chapter 5

Image Blending for Underwater Robotic Detection

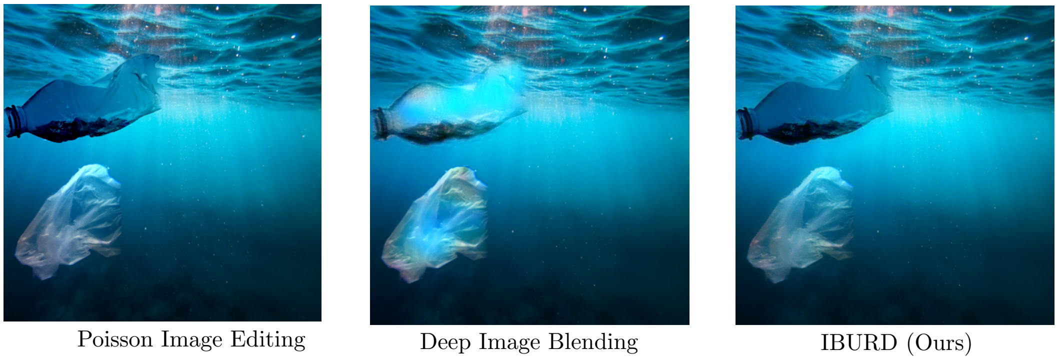
Although recent breakthroughs in machine learning and computer vision have advanced robotic perception in degraded environments, the detection of underwater debris remains an open problem because of its visually challenging nature and scarcity of data to train sufficiently accurate deep detection models. This chapter presents a novel image blending pipeline, *IBURD*, that creates realistic synthetic images and their pixel-level annotations using target background images, source objects images, and the objects' annotations, to provide training data for deep learning detection models. We train instance segmentation models only using synthetic datasets and measure their performances on a robotic platform in pool and ocean environments to evaluate our proposed method, which is introduced in detail in the following sections.

5.1 Related Work

Recent advances [94, 95] in deep learning have vastly improved object detection and instance segmentation results in the terrestrial domain. Such progress has been achieved by developing effective designs of models and training them with large datasets [65, 96] containing millions of images and corresponding labels. Even with such advances, detecting underwater debris still remains challenging. While [29] presented the first deep learning-based approach to detect underwater debris and outperformed previous non-deep learning approaches, the accuracy was worse than general object detection tasks due to their small-size dataset. To increase the debris detection accuracy, [30] proposed a larger dataset, TrashCan, which has both bounding box and pixel-level annotations for object detection and instance segmentation along with baseline results using Mask R-CNN [69] and Faster R-CNN [57]. However, increasing the dataset size to improve debris detection accuracy further is not scalable due to debris data scarcity and labeling costs. To overcome the data scarcity issue, [31] proposes a generative method, augmenting the existing dataset with synthetic underwater debris images. While the method can create realistic synthetic images, it still requires additional labeling efforts to be used for training detectors.

Style transfer [97, 98] is an approach for changing the appearance of one image based on the visual style of another. [99, 100] use this to improve detection on images taken from various domains (*e.g.*, different light conditions and image clarity). They aim to account for low-level texture changes in images by updating them to have the same style throughout the data. [101] also attempts to improve detection using style transfer, by having the detector learn high-level features (*e.g.*, object shape) instead of low-level features (*e.g.*, the texture of paintings). [102] uses style transfer to simulate various types of noise that may be present in real-world data. [103, 104] use style transfer to imitate varying light conditions. Style transfer has been applied beyond RGB images; *e.g.*,[105] converts RGB images from COCO dataset [65] to thermal images and uses them to train a thermal image detector. However, style transfer only produces data consisting of annotations with the same scale, locations, and backgrounds as source images, since it merely changes the appearance of existing images. As a result, this makes detectors trained using the images from style transfer approaches prone to overfitting with respect to object scales, locations, and backgrounds.

Unlike style transfer, image blending-based methods allow placing of source image patches anywhere on target background images. [2] introduces Poisson editing using Laplacian information to smooth the boundary between the image patches and target images. Recently, [106] uses a GAN-based approach for image blending, producing realistic images; however, it requires image pairs of empty backgrounds and objects placed in the backgrounds to train, limiting its use when the source data is limited. [107] modifies [2] to find spaces within a given image plane to blend an object. However,



Poisson Image Editing

Deep Image Blending

IBURD (Ours)

Figure 5.1: Comparison of generated images using three approaches: Poisson image editing [2], Deep image blending [3] and our method, IBURD. In our approach, we can successfully prevent over-stylization of the blended objects.

detectors trained with their synthetic data show degraded performance on real data due to the style discrepancy between the blended objects and backgrounds in the dataset. [108] uses a harmonization blending approach to create new data for aerial search and rescue, but it does not blend the boundary of target objects. [3] presents a two-stage deep network-based approach to blend the image, generating better-blended images. They also claim to not need additional training data unlike [106]. The proposed method is mainly used for artistic purposes and and struggles with blending transparent source images to background images as seen in Fig. 5.1. The method is only tested with 20 images and takes approximately four minutes for blending one object in a 512×512 sized image.

Our proposed approach, IBURD, allows us to place source images at various locations and scales in target background images with relevant bounding box and pixel-level annotations within 50 seconds, which is five times faster than [3]. Our method can tackle the task of blending transparent objects by using Poisson editing, which previous methods fail to address. Additionally, it deals with object distortion in style transfer using Fast Fourier Transform (FFT) [109] based weight adjustment for loss.

5.2 Methodology

IBURD (Fig. 5.2) uses a two-pass process similar to [3], where the two passes are Poisson editing and style transfer. Poisson editing blends an object onto a background, and style

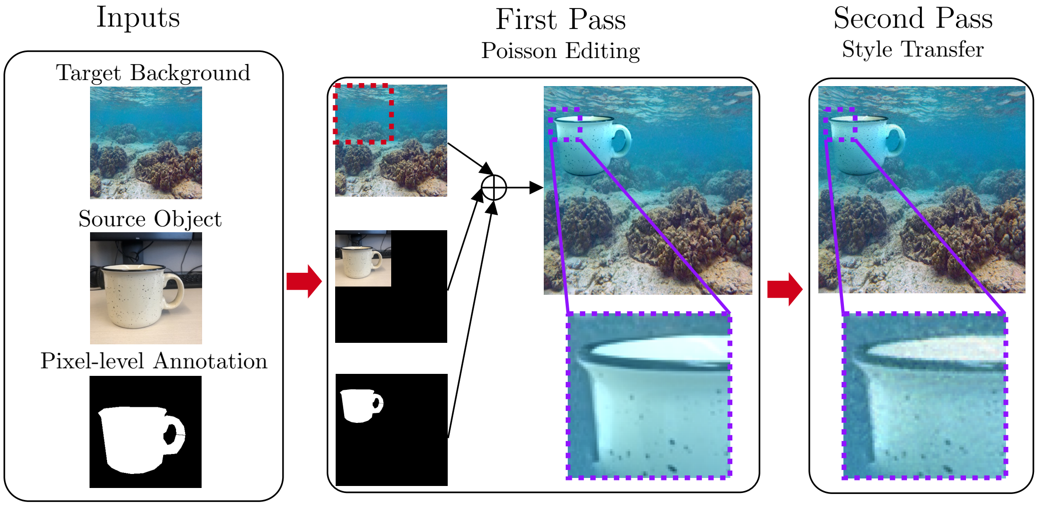


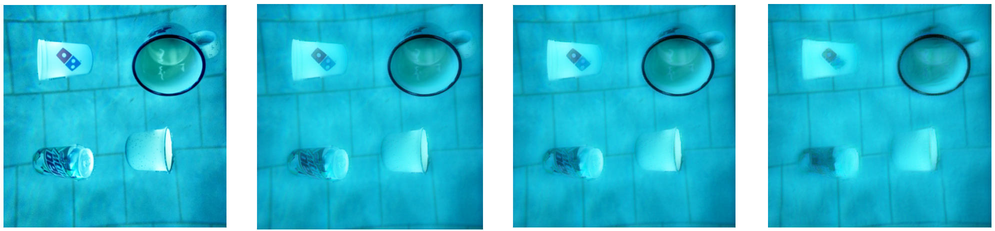
Figure 5.2: Illustration of the IBURD pipeline: The source image with its pixel-level annotation and background image are fed as inputs to the first pass. The first pass resizes and rotates the source object and selects a location for blending it. Poisson editing smooths the boundary between the blended object and the background. The second pass changes the style to produce the final image. In the zoomed in region of the object the style difference between appearance of first and second pass image is visible.

transfer changes the appearance of the object. Our pipeline takes the background image, the object image, and the object's pixel-level annotation as inputs.

In the first pass, we randomly resize and rotate the object image along with its annotation. To place the object image in the background, we split the given background image into a grid and place the object image in a randomly selected cell within the grid. We then use Poisson image editing [2] for blending, eliminating drastic boundary gradient changes between the object and the background.

Next, we feed the output from the first pass and the background image to the second pass, which reconstructs the first pass output to reflect the background style (*i.e.*, underwater appearance). For this, we use the total loss (Eq. 5.4) consisting of three different loss functions: style loss (Eq. 5.1 [110, 3]), content loss (Eq. 5.2 [110, 3]), and total variation loss (Eq. 5.3 [111, 3]). Style loss compares the background and the reconstructed image. The content loss reflects the differences between the blended image from the first pass and the reconstructed image.

In Eq. 5.1 and 5.2, I_r is the reconstructed image after optimization, I_b is the original



First Pass

Style loss weight = 800

Style loss weight = 1500

Style loss weight = 15000

Figure 5.3: Generated image samples before and after the second pass: With the increase in weight, the distortion of object boundary increases. Generated images using various weights are compared with the output of first pass, which does not have any distortion due to style transfer.

background image, I_{fp} is the output from the first pass, L is the number of convolution layers, N_l is the number of channels in activation, M_l is the number of flattened activation values used by each channel, F_l is an activation matrix computed from deep network F at layer l, $G_l = F_l F_l^T$ is the gram matrix, α_l and β_l are the respective weights.

$$Loss_{style} = \sum_{l=1}^{L} \frac{\beta_l}{2N_l^2} \sum_{i=1}^{N_l} \sum_{k=1}^{M_l} (G_l[I_r] - G_l[I_b])_{ik}^2 \tag{5.1}$$

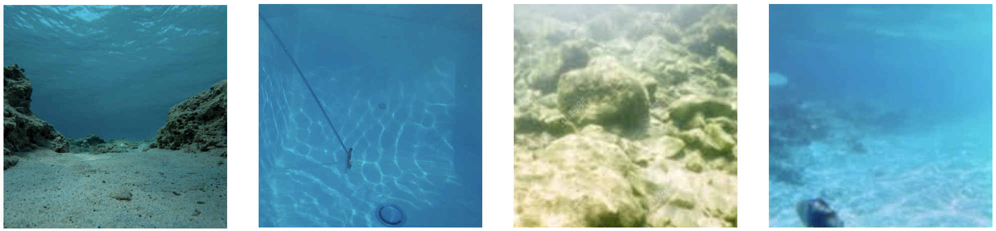
$$Loss_{content} = \frac{\alpha}{2N_L M_L} \sum_{i=1}^{N_L} \sum_{k=1}^{M_L} (F_L[I_r] - F_L[I_{fp}])_{ik}^2 \tag{5.2}$$

Using style and content loss together prevents the object from being over-stylized. The style and content loss are computed using VGG-16 [112] network's layers. Specifically, style loss uses layers relu1_2, relu2_2, relu3_3, and relu4_3 ($L = 4$). Content loss uses layer relu2_2 ($L = 1$). In addition, total variation loss preserves the low-level features in an image. In Eq. 5.3, I is the reconstructed image. $I_{m,n}$ denotes the entry on m^{th} row and n^{th} column.

$$Loss_{tv} = \sum_{m=1}^{H} \sum_{n=1}^{W} |I_{m+1,n} - I_{m,n}| + |I_{m,n+1} - I_{m,n}| \tag{5.3}$$

$$Loss_{total} = \lambda Loss_{style} + \mu Loss_{content} + \nu Loss_{tv} \tag{5.4}$$

In Eq. 5.4, λ represents the style loss weight, μ represents content loss weight, and



Blurriness Score = 0 Blurriness Score = 3 Blurriness Score = 5 Blurriness Score = 8

Figure 5.4: Sample images used in the survey to quantify blurriness. The scores obtained by the survey are shown under respective images. A lower score signifies clear image and a higher score corresponds to increased blurriness.

ν represents total variation loss weight.

With typical style transfer approaches [3, 106] developed for artistic purposes, the total loss (Eq. 5.4) distorts the object boundaries when higher style loss weights (*e.g.*, $\lambda = 15000$ in Fig. 5.3) are used on a blurry background image. However, images for training detectors need to maintain the object's content while reflecting the blurriness of the background. In other words, the style loss weight must be regulated to account for background quality while not losing the object shape completely. To achieve this, we use FFT [109] to use spatial frequency as a measurement of image blurriness. After applying FFT to the background image, its mean value from the newly transformed image is computed, which acts as an indicator of blur – the higher the value, the clearer the image.

Since the blurriness of an image is a subjective measure among individuals, it is difficult to map the blurriness to a style loss weight (λ) directly. We use a survey to validate the correlation between a FFT measure and the perceived blurriness in an image. We conduct the survey with 10 participants who were asked to rate 10 images (Fig. 5.4) on a scale of 0 to 10 (0 being clear and 10 being extremely blurry). We verify the correlation between the survey and FFT mean scores, and map the ranges of FFT mean values to four style loss weight values via empirical evaluations (Table 5.1).

Along with the blending process, the annotation information on the newly reconstructed image is generated by translating and transforming the original annotation coordinates based on the blending location. This information is collected in COCO format, suitable for training common object detectors.

Table 5.1: Blurriness-based style loss weights

FFT mean value	Avg. survey score	Style loss weight λ
mean>= 40	0	30000
40 >mean>= 10	1.4	15000
10 >mean>= 0	4.5	1500
0 >mean	7.5	800

Table 5.2: Data distribution of source object images

Class name	Number of source images
Bag	7
Bottle	6
Glass Bottle	5
Cup	8
Mug	10
Can	9
Starfish	2

5.3 Experiments

5.3.1 Data Collection

To evaluate IBURD's efficacy, we start by collecting source images of objects on the ground. We use seven different classes of commonly found objects in marine debris, giving us a total of 47 source images, and their distribution across classes is shown in Table 5.2. We then manually annotate the images, providing class labels, bounding box and segmentation information. We use two types of background images for blending, one from the open sea and other from swimming pools. We collect seven images taken at different locations from our experimental pool, and add three pool images from online sources to add more variety. In the case of the sea background, we use 10 background images sourced from the Internet.

5.3.2 Image Blending and Data Generation

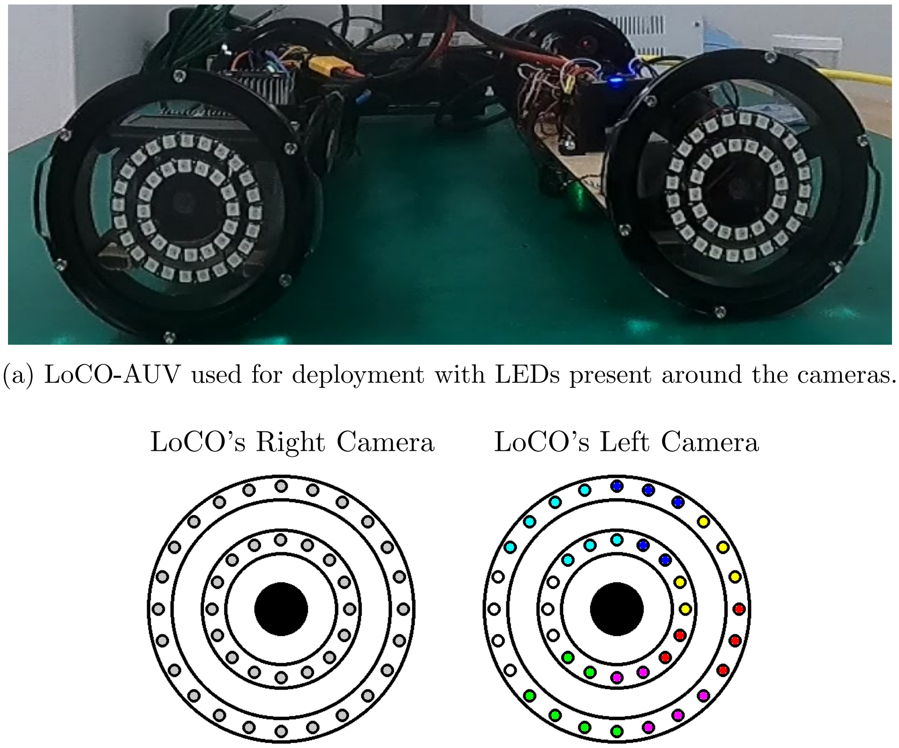
To generate a diverse dataset, we use four different rotations (0°, 90°, 180°, 270°) and four different sizes (96×96, 128×128, 192×192, 256×256) for the 47 source object images. The background image and the final blended image are of size 512×512. We determine the object location by randomly selecting one section within the grid on the background image plane. For multi-object blending, we divide the background (size 512×512) into 2×2 grid considering the maximum size of source images (*i.e.*, 256×256). For the single object case, the image plane is divided based on the current source image size (*i.e.*, 4×4 grid for image size 96×96 and 128×128, 2×2 grid for image size 192×192 and 256×256). By using various locations, rotations and scales, we create images with up to four objects blended in the same background.

For the sea backgrounds, we create $1,880$ images with one object, $2,209$ images with two objects, $3,008$ images with three objects and $4,096$ images with four objects making a total of $11,193$ images for training. Similarly, in the case of pool background, we create $1,880$ images with one object, $2,209$ images with two objects, $2,396$ images with three objects and $1,731$ images with four objects, giving a total of $8,216$ images for training. We generate a smaller dataset for pool images, since it is a controlled environment and has limited variety in the kind of backgrounds that can occur in real-world images.

For the style transfer, we use 100 iterations and $L - BFGS$ solver for optimizing the total loss. The content loss weight is, $\mu = 1$, and the total variation loss weight is, $\nu = 10^{-6}$. The style loss weight λ is based on image blurriness (Table 5.1).

5.3.3 Object Detection and Instance Segmentation

To evaluate the efficacy of our framework, we first select YOLACT [113] as an instance segmentation model based on its inference speed and performance with pre-trained weights. We choose ResNet50-FPN as a backbone of YOLACT to obtain a reasonable inference time on the low-power mobile GPU on LoCO-AUV [4]. We train the model with three different sets of data: synthetic pool data, synthetic ocean data, and TrashCan data (containing actual underwater debris imagery).



(a) LoCO-AUV used for deployment with LEDs present around the cameras.

(b) The schematic of the LEDs installed on LoCO: LEDs mounted around LoCO's left camera, are used to monitor detector performance in real-time during experiments. We divide the LEDs into seven sections and corresponding sections individually flash when relevant classes are detected in the image frame. The colors denote the following classes: Blue-Cup, Yellow-Mug, Red-Bottle, Magenta-Starfish, Green-Can, White-Bag, Cyan-Glassbottle.

Figure 5.5: LED indicator setup on LoCO for monitoring object detection performance.

5.3.4 Robot Setup

We choose LoCO-AUV [4] to run the detector model with trained weights due to its capability to run deep learning models on its mobile GPU (*i.e.*, Nvidia Jetson TX2). We use image frames from the right camera of LoCO-AUV to make inferences. We adopt an LED lighting indicator system [114] (Fig. 5.5) installed on LoCO-AUV's left camera, to visually examine different weights' performance during deployments. The system consists of 40 LEDs, and we sectionalize them into seven groups. This way, a designated group(s) of LEDs lights up when a relevant class object(s) is detected in the current image frame. Additionally, we only use the LEDs around the left camera to avoid the image disruption caused by the light reflection on the acrylic shell in front of each camera.

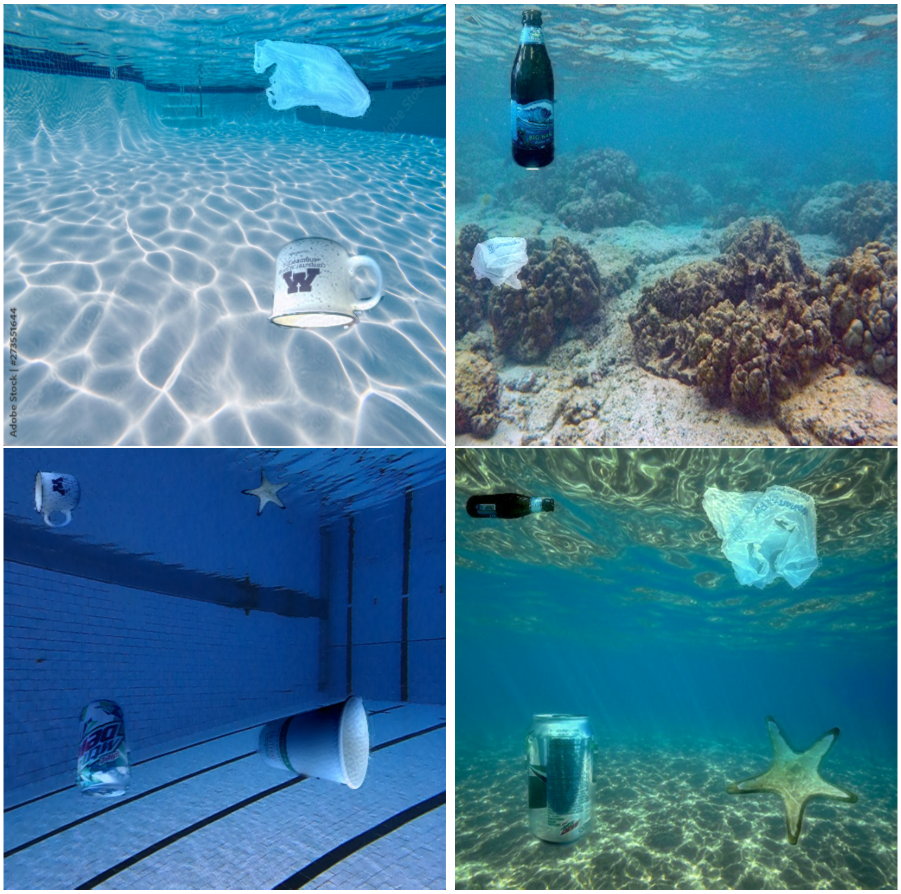
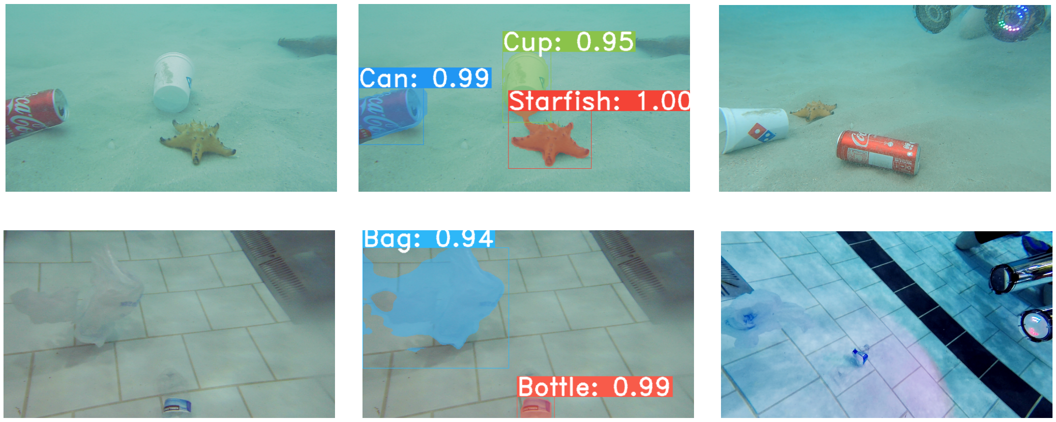


Figure 5.6: Sample images generated from IBURD. The first column shows images blended on pool backgrounds. The second column contains objects blended on ocean backgrounds.

5.4 Results

We generate images with various ocean and pool backgrounds(Fig. 5.6). Note that we do not lay much emphasis on the quantitative evaluation of the images themselves, since the scope of this work is not to improve the appearance of images for aesthetic purposes, but to create an effective dataset for training robust visual debris detectors. The desired properties of generated images are a smooth transition border between the



(a) Raw input images

(b) Visualization of detection

(c) Detection response using LEDs

Figure 5.7: The first row consists of real images from the ocean in Barbados. The second row contains real images from pool. (a) shows the raw images from the perspective of the camera on LoCO-AUV [4]. These images are used as input for the detector network. (b) visualizes the segmentation mask and bounding box predicted by the detector. The network detects the objects with high confidence scores. (c) shows the LED response in real-time. In the ocean image (first row) in (c) cup, starfish, and can are visible with their respective LED colours blue, magenta, and green. In the pool image (second row) in (c) bag and bottle are present with their LED indicating their class colours, white and red.

object and background images and the preservation of the object's content. Our pipeline is able to account for these properties and remove the background in the source object image completely from transparent (*e.g.*, plastic bottles) and translucent objects (*e.g.*, glass bottles and plastic bags) (Fig. 5.1). Additionally, the objects do not get heavily distorted even when other objects exist in the background (*e.g.*, coral reefs in Fig. 5.6). IBURD achieves five times faster runtime compared to the recent work, Deep Image Blending method [3]. Our method can generate an image with one object in 25 seconds. The runtime increases as more objects are blended in the background, reaching up to 50 seconds for four objects.

We train a detector on synthetically generated datasets and deploy it on LoCO-AUV [4]. We conduct experiments in pool and ocean environments as explained in Sec. 5.3. The detector performs at 1-3 frames/second (FPS) on Nvidia Jetson TX2, and we monitor detector output through the LED indicator. The detector successfully infers object classes, which are in the training datasets, during pool and ocean deployments

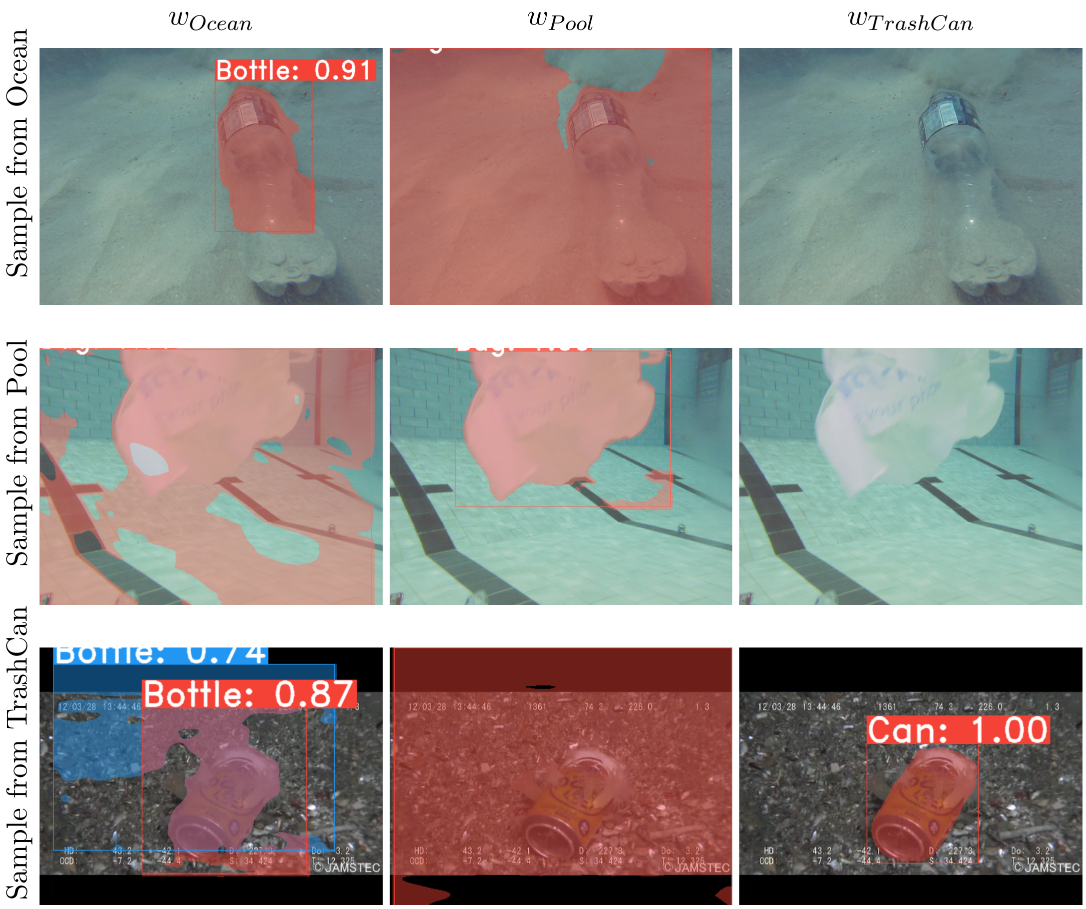


Figure 5.8: Demonstration of detector performance on real-world images: Images in each row are samples from three different sources (*i.e.*, Ocean, Pool, and TrashCan). Columns correspond to detector weights trained on ocean synthetic (w_{ocean}), pool synthetic (w_{pool}), and TrashCan ($w_{TrashCan}$) datasets. For each weight, the performance is evaluated on all three sample images. The detector performs best when the sample images are from a similar or the same environment as the training dataset.

(Fig. 5.7). We also evaluate the performance of detectors trained with three different datasets (*i.e.*, synthetic ocean, synthetic pool, and TrashCan) using images from three different environments (*i.e.*, Ocean, pool, and TrashCan), in total nine cases (Fig. 5.8). When the detector is trained and tested with images from the same environments, objects are inferred correctly. Real images from ocean, pool, and Trashcan are tested on the detector with weights w_{Ocean}, w_{Pool}, and $w_{TrashCan}$, respectively. However, if the weight and sample image pairs are from different environments, the detector shows degraded performance or completely fails to detect objects. This demonstrates the necessity of using relevant datasets for target environments. In our experiments, we show IBURD can generate realistic synthetic datasets for pool and ocean environments without using any real images of objects in the target environment. We also demonstrate that the detectors trained with synthetic datasets which have similar visual features to a target environment perform better than ones trained with publicly available datasets if the target environment is known a priori.

5.5 Conclusions

In this chapter, we present an image blending pipeline, *IBURD*, that generates synthetic images using source object images, their annotations and background images via a two-pass approach. Advantages of our pipeline include the capability to handle transparent source object images without creating artificial borders between the object and background. Additionally, the pipeline uses the blurriness score to dynamically synthesize source and target background images, resulting in the creation of more realistic synthetic images than previous methods. Our approach also provides pixel-level annotations for each synthetic image, eliminating the labor-intensive dataset annotation process. Our results show that the synthetic data-trained models can accurately detect objects without being trained on any real data which includes those same objects. This demonstrates that our pipeline enables training of object detection and instance segmentation networks suitable for the data-scarce underwater debris detection problem. This is the last chapter of Part I. We will move to Part II where we introduce our research for robotic exploration and localization.

Part II: Robotic Exploration and Localization for AUVs

Unlike terrestrial and aerial robots, underwater robots frequently encounter unknown environments as the aquatic domain is very sparsely mapped. Hence, robotic localization in such environments is extremely difficult. Additionally, to navigate safely, robots are required to approach objects cautiously. However, existing algorithms are incapable of adjusting distances to objects, leading to inefficient exploration and hampering search operations by colliding with obstacles. To address this issue, Part II provides methods to localize AUVs and navigate underwater, and it is composed of two chapters. Chapter 6 proposes to fuse depth and instance segmentation information with an artificial potential field algorithm (APF), which makes it possible for robots to keep flexible distances to objects (*i.e.*, avoiding or approaching) during exploration. After detecting underwater debris during exploration, accurate localization of AUVs is essential to estimate the location of detected debris. However, underwater localization is a challenging and open problem due to the unique challenges AUVs face. GPS and other forms of electronic communications are either completely unavailable or limited to extremely short ranges underwater. Chapter 7 presents a low-cost localization approach for an AUV by fusing bathymetry data of underwater environments with depth and altitude data from an AUV.

Chapter 6

Stereo Visual Obstacle Avoidance using Semantic Information

Mobile robots in unstructured environments must rely on an obstacle avoidance module to navigate safely. The standard avoidance methods measure the locations of obstacles with respect to the robot but are unaware of the obstacles' semantics. Consequently, the robot cannot exploit the semantic information about obstacles when making decisions about how to navigate. In this chapter, we introduce an efficient obstacle avoidance algorithm for robots combining semantic information and depth estimation [1] from a stereo vision system. The following sections describe our approach in detail.

6.1 Related Work

6.1.1 Instance Segmentation

Research in object detection has studied models to improve accuracy while keeping real-time inference speed since the appearance of YOLO [56], one of the first real-time object detection models. However, instance segmentation poses more complex challenges, and achieving good accuracy in real-time has been difficult. FCIS [115] is the first end-to-end CNN-based model for instance segmentation. It is built on R-FCN [116] and

[1] The term "depth estimation" in this chapter refers to the method which restores the distance in the third dimension from a given 2D image.

utilizes position-sensitive inside/outside score maps to generate instance segmentation proposals. Mask R-CNN [69], which is an extension of Faster R-CNN [57], performs segmentation in a two-stage process by generating Region of Interest (RoI) proposals first and then creating a mask based on the RoI from the first stage. PANet [117] improves the accuracy of segmentation from Mask R-CNN by enriching information propagation. MS R-CNN [118] outperforms Mask R-CNN by adding a *MaskIoU head* to align the scores of the masks. Although the aforementioned models show accurate results, their two-stage-based structures make real-time instance segmentation infeasible. In order to overcome the structural problem, YOLACT [113] conducts two predictions in parallel: mask prototypes and per-instance mask coefficients. Then, the predictions are combined linearly to yield masks. This allows a single-stage structure and inference in real-time with reasonable accuracy.

6.1.2 Obstacle Avoidance

Obstacle avoidance, unsurprisingly, has seen significant development (*e.g.*, [119, 120, 121]) given its importance in safe robot navigation. Here, we focus specifically on sensor-based approaches where no information about the environment is available beyond what is received from sensors (see [122] for a complete discussion). Sensor-based approaches typically plan a short-horizon trajectory at every time step [123, 124]. A classic obstacle avoidance technique is the Artificial Potential Field, first proposed by [125]. This technique assigns artificial repulsive fields to obstacles and attractive fields to goals, thereby guiding the robot toward a goal while simultaneously avoiding obstacles. Other approaches include vector field histograms (VFH) [126], receding horizon control [127], and Voronoi diagrams [128].

Most obstacle avoidance that incorporates semantic information is focused on developing socially-aware responses to human obstacles (*e.g.*, [129, 130, 131]). Similar to our approach, [130] instructs the robot to avoid humans more than inanimate objects. However, their work is focused on path planning in mapped environments and uses model-based methods to estimate the human's location.

Another approach for obstacle avoidance in marine robotics, based on conditional imitation learning, is presented in [121]. This approach uses data collected from expert users to learn what navigational action to take given an input image but does not explicitly model different behaviors for different types of obstacles.

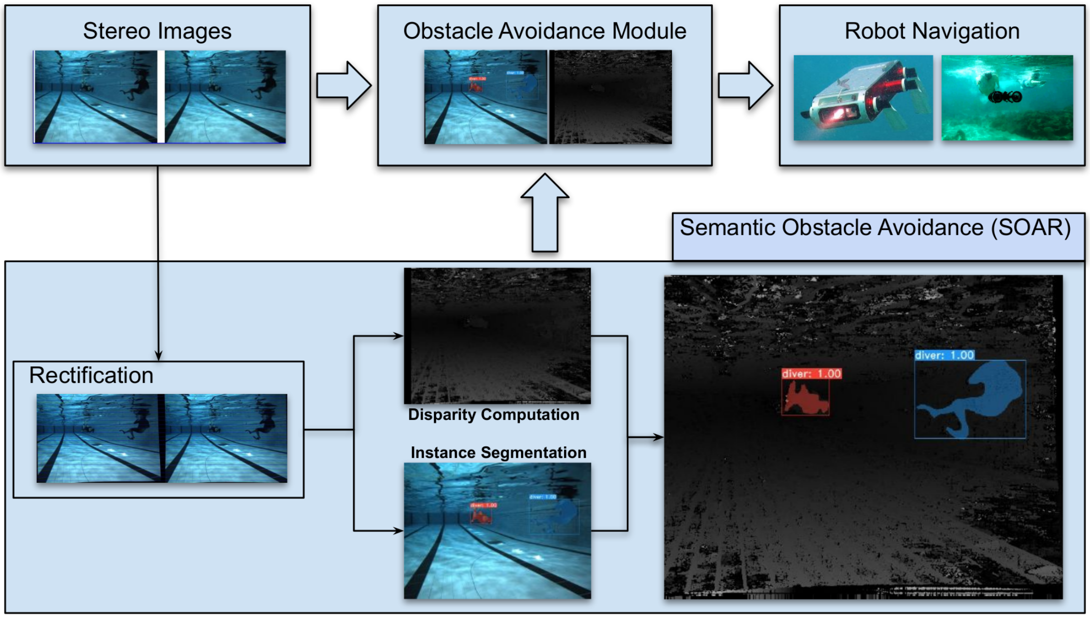


Figure 6.1: Illustration of our obstacle avoiding approach which fuses depth and semantic information for selective avoidance. The input is a pair of stereo images, which is used to both compute disparity and, through YOLACT, generate semantic labels for obstacles in the scene. Fusing these, a robot has both depth estimates and semantic information about potential obstacles, enabling it to select navigation strategies depending on the nature of the obstacle.

6.2 Methodology

The proposed approach incorporates both instance segmentation and depth information to intelligently avoid obstacles in unstructured and dynamic environments, with the goal to optimize robot paths (*e.g.*, in terms of distance traveled, energy spent, and time taken) without compromising obstacle avoidance capabilities. The system obtains object labels and pixel locations from instance segmentation and fuses the information with depth information to assign clearance distances to obstacles.

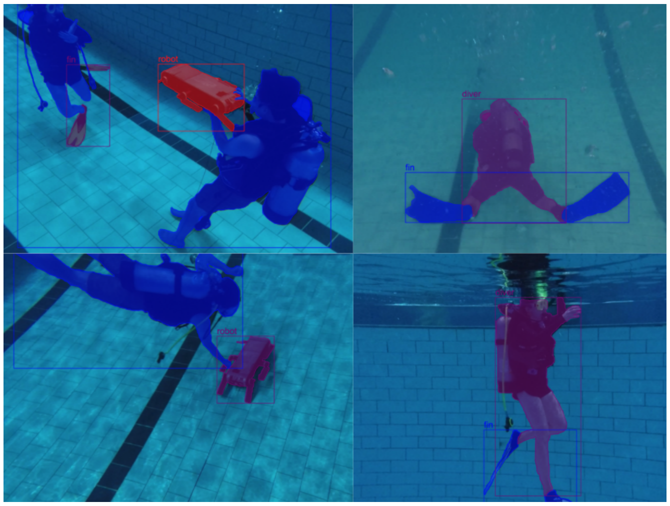


Figure 6.2: Examples of labeled training images from our underwater dataset showing three classes: *diver, robot*, and *fin* (the colors for each class are randomly selected per image).

6.2.1 Information Fusion using a Stereo Camera

We use a stereo camera as our only sensor to acquire (1) pixel-level masks and labels of each object using instance segmentation, and (2) depth information. Once both are acquired, we fuse them to provide semantic information to an obstacle avoidance module (see Fig. 6.1).

Instance Segmentation with Transfer Learning

We choose YOLACT [113] as the base instance segmentation module due to its real-time inference and competitive accuracy. We use ResNet50-FPN as a backbone network for achieving maximum inference speed since robots are likely to use low-power computation units (*e.g.*, an Nvidia Jetson TX2) to perform semantic inference. We collect a total of $2,263$ images, of which $1,682$ are labeled with *diver*, *robot*, and *fin* (*i.e.*, diver's flippers) classes; see Fig. 6.2 for training images labeled using the Supervisely [71] tool. In addition, we use 581 images from the SUIM dataset [132] which has *diver*, *fish*, and *robot* classes. We refine a pre-trained YOLACT model, initially trained with the MS COCO dataset [65], with this additional data.

Depth Estimation

Our depth estimation process is as follows:

1. We perform stereo rectification to obtain a transform matrix **R**, projection matrix **P**, and disparity-to-depth mapping matrix **Q** for each camera using a camera matrix **K** and distortion parameters **D** from each camera, a rotation matrix **R** between the first camera coordinate and the second camera coordinate, and a translation vector **T** between two cameras.
2. Next, we remove distortion from each image using the **K**, **D**, **R**, and **P** matrices.
3. After rectifying each pair of images, we run stereo matching to generate a disparity map.
4. Lastly, we estimate the depth information from the disparity map using **Q**.

6.2.2 Obstacle Avoidance

Our obstacle avoidance algorithm is inspired by the Artificial Potential Field (APF) method [125]. APF uses an attractive potential f_a to guide the robot toward the goal and a repulsive potential f_r to push the robot away from obstacles. The attractive potential is calculated as:

$$f_a(x) = c(|x - x_g|)^2 \tag{6.1}$$

Here c is a scaling constant, x is the current robot position, and x_g is the goal position. The repulsive potential is calculated as

$$f_r(x) = \begin{cases} \eta(\frac{1}{p(x)} - \frac{1}{d_0})^2 & \text{if } p(x) \leq d_0 \\ 0 & \text{if } p(x) > d_0 \end{cases} \tag{6.2}$$

where η is a constant, $p(x)$ is the closest obstacle to the position x, and d_0 is the largest distance from the obstacle at which the robot can sense the obstacle's repulsive force.

In our approach, we determine d_0 based on semantic information about the obstacles. For instance, we assign $d_0 = 0$ for objects we can ignore (*e.g.*, bubbles or sports balls) while assigning a larger value for the objects (*e.g.*, coral reef, robots, people) we intend to avoid. In our approach, unlike APF, the robot navigates around the boundary of an obstacle, keeping a constant distance of d_0 from the obstacle (Fig 6.3). The circumnavigation behavior is similar to the bug-2 navigation algorithm [133]. The robot, however, may face the challenge of maintaining fixed distances (*i.e.*, d_0) from obstacles when circumnavigating due to errors in state estimation and external forces (*e.g.*, surge for underwater robots operating in open waters and wind for aerial robots).

We introduce two unit vectors, $\hat{a}$ and $\hat{r}$, to implement the circumnavigation with the concept of attractive and repulsive potentials from APF. $\hat{a}$ points from the robot towards the goal and $\hat{r}$ points from the robot to the obstacle. With the two vectors, we update the robot's direction of movement $\hat{v}$ at any given point as follows:

$$\hat{v} = \begin{cases} \hat{a} & \text{if } |x_o - x| > d_0 \\ \hat{a} + c_1 c_2 \hat{r} & \text{if } |x_o - x| \leq d_0 \end{cases} \tag{6.3}$$

where c_1 is defined as a negative dot product between $\hat{a}$ and $\hat{r}$, and c_2 is an additional factor to keep the distance d_0 between the robot and the obstacle.

$$c_1 = -\hat{a} \cdot \hat{r} \tag{6.4}$$

We introduce the constant c_1 to make $\hat{v}$ perpendicular to $\hat{r}$ when the distance between the robot and the obstacle is d_0. However, due to the robot's momentum, the robot may still approach closer than d_0 to the obstacle. We use the additional factor c_2 to scale the repulsive component $c_1\hat{r}$ and to enforce distance d_0 from the obstacle:

$$c_2 = \frac{1 - b}{d_0}|x_o - x| + b \tag{6.5}$$

Here x_o is the obstacle position, x is the robot position, and b is a constant greater than 1 that represents the maximum value $c_1\hat{r}$ can be scaled by. c_2 scales inversely with the robot's distance to the obstacle, and obtains a value between 1 and b as we approach the obstacle.

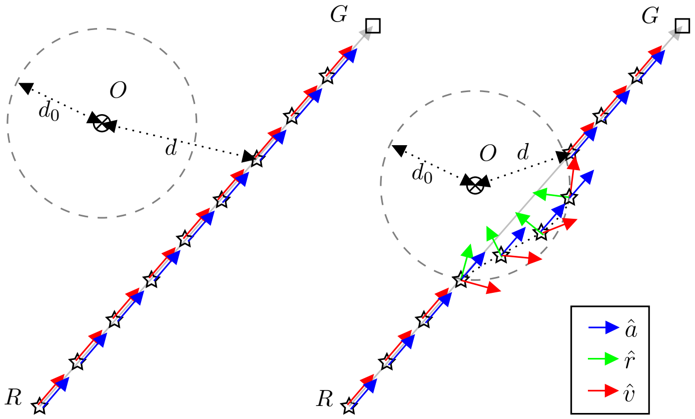


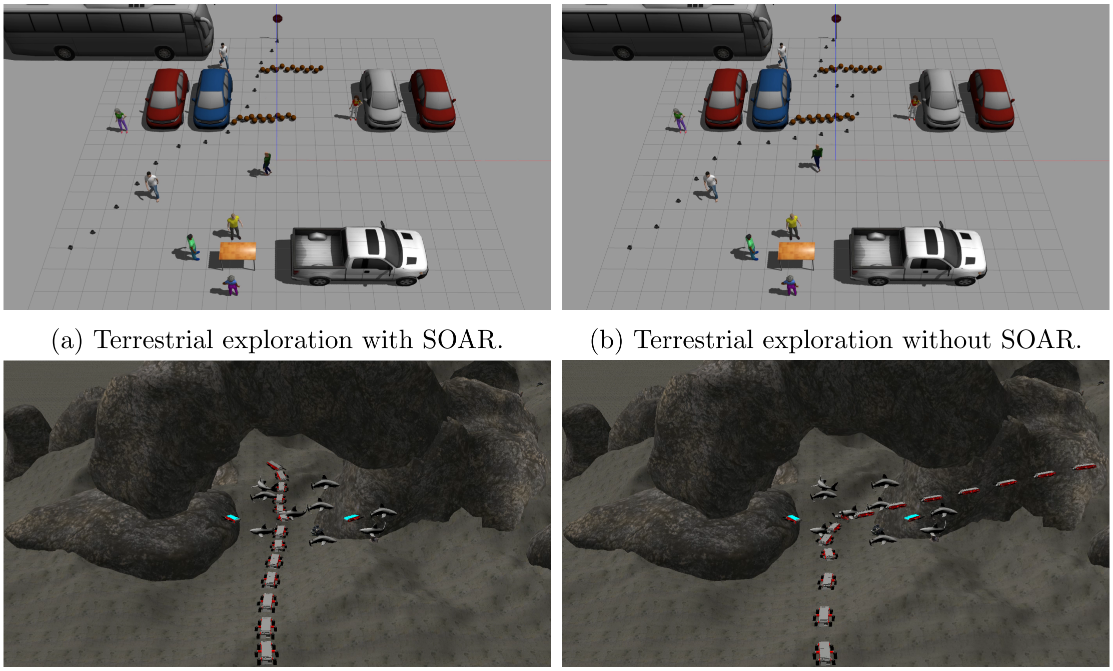
Figure 6.3: Example trajectories of a robot around an obstacle using the proposed approach. The repulsive potential $\hat{r}$ affects the robot only if $d \le d_0$. The position of object O is represented by a cross, the goal position G by a square, and the position of robot R over time with stars. (left): when the obstacle is not along the direct path from the robot to the goal, it does not affect robot navigation. (right): the robot's direct path to the goal brings it closer than d_0 distance to the obstacle; our algorithm forces the robot to circumnavigate around it.

6.3 Experiments and Results

The ongoing COVID-19 pandemic prohibited field trial validations of the proposed method. However, we use realistic simulation using ROS Gazebo [134] worlds for both terrestrial and underwater cases to validate our algorithm. Because our goal is local navigation of unstructured, mapless environments, we choose goal points for each case with one condition: the goal point should be something visible to the robot when the robot is at its starting position if there is no obstacle between the robot and the goal points. Additionally, we use a mobile GPU (Nvidia Jetson TX2) with a stereo camera (Intel RealSense) to evaluate the performance of our model to mimic realistic robotic hardware.

6.3.1 Simulated Terrestrial Trials

We have created a terrestrial world in Gazebo, simulating a parking lot environment. The scene was chosen to mimic a robot attempting delivery or curbside pickup from a departmental store, a relatively common occurrence in many parts of the world under the Coronavirus pandemic. The world has various types of objects, including *sports balls, cars, buses, people walking* or *standing*, and *tables*. We evaluate our model on a



(a) Terrestrial exploration with SOAR.

(b) Terrestrial exploration without SOAR.

(c) Underwater exploration with SOAR.

(d) Underwater exploration without SOAR.

Figure 6.4: Samples from the terrestrial and underwater simulation tests. In Fig. 6.4a and 6.4b, the Turtlebot moves from the bottom left corner to the upper middle area. In Fig. 6.4c and 6.4d, Aqua moves from the lower middle point to the middle of the arch. Fig. 6.4a and 6.4c show that SOAR finds more efficient paths to explore environments while avoiding significant obstacles.

workstation equipped with an Intel i5-8600K CPU and an Nvidia GTX 1080 GPU. We add a stereo camera to a simulated Turtlebot robot to estimate depth and infer instance segmentation. To validate our algorithm, we intentionally block the shortest path from a robot to a goal point with *sports balls*, as shown in Fig. 6.4a and 6.4b, since we select the *sports ball* category as an object the robot can safely run into (*i.e.*, a *not-an-obstacle* object). The Turtlebot starts from the bottom left corner of the world and aims to reach the stop sign at the top. We use pre-trained COCO weights with YOLACT for instance segmentation. We use the modified APF-based obstacle avoidance algorithm as described in Section 6.2 under two conditions: receiving semantic information about obstacles (SOAR) and without receiving any semantic information about obstacles (non-SOAR). To measure the effectiveness of the semantic obstacle avoidance approach, we measure the travel time from a starting point to a goal point to evaluate each model's performance by running 10 tests for each case. We also note the path chosen by the robot in each of the SOAR and non-SOAR cases.

6.3.2 Simulated Underwater Trials

Our underwater world has been designed to mimic the ocean floor environment, including corals, rocks, and fauna. The world includes *fish, robots*, and an *underwater rock arch formation*. As shown in Fig. 6.4c and 6.4d, the *arch* is blocked by *fish* and *robots* to test the efficacy of the SOAR approach compared to non-SOAR. We select the *fish* category as a *not-an-obstacle* object class. We simulate the Aqua AUV [135], equipped with a stereo camera, to test our model with both SOAR and non-SOAR algorithms. The robot starts from the bottom of the world and aims to travel through the arch to the other side of the rock formation. We use the instance segmentation model trained on our own dataset (as mentioned in Section 6.2.1) with the four classes (*i.e.*, *diver, robot, fish*, and *fin*). The hardware and trial configurations are unchanged from that of the terrestrial case.

6.3.3 Results

We use both quantitative and qualitative evaluation to test our semantic obstacle avoidance approach, as shown in Fig. 6.4 and Table 6.2. Vision algorithms are implemented

Table 6.1: Instance segmentation results (mAP) trained on our underwater dataset

	all	.50	.55	.60	.65	.70	.75	.80	.85	.90	.95
box	71.14	91.96	90.29	89.48	87.89	84.09	79.54	75.20	62.56	39.07	11.29
mask	69.38	93.72	93.51	92.14	90.47	85.64	81.71	69.39	55.94	29.55	1.71

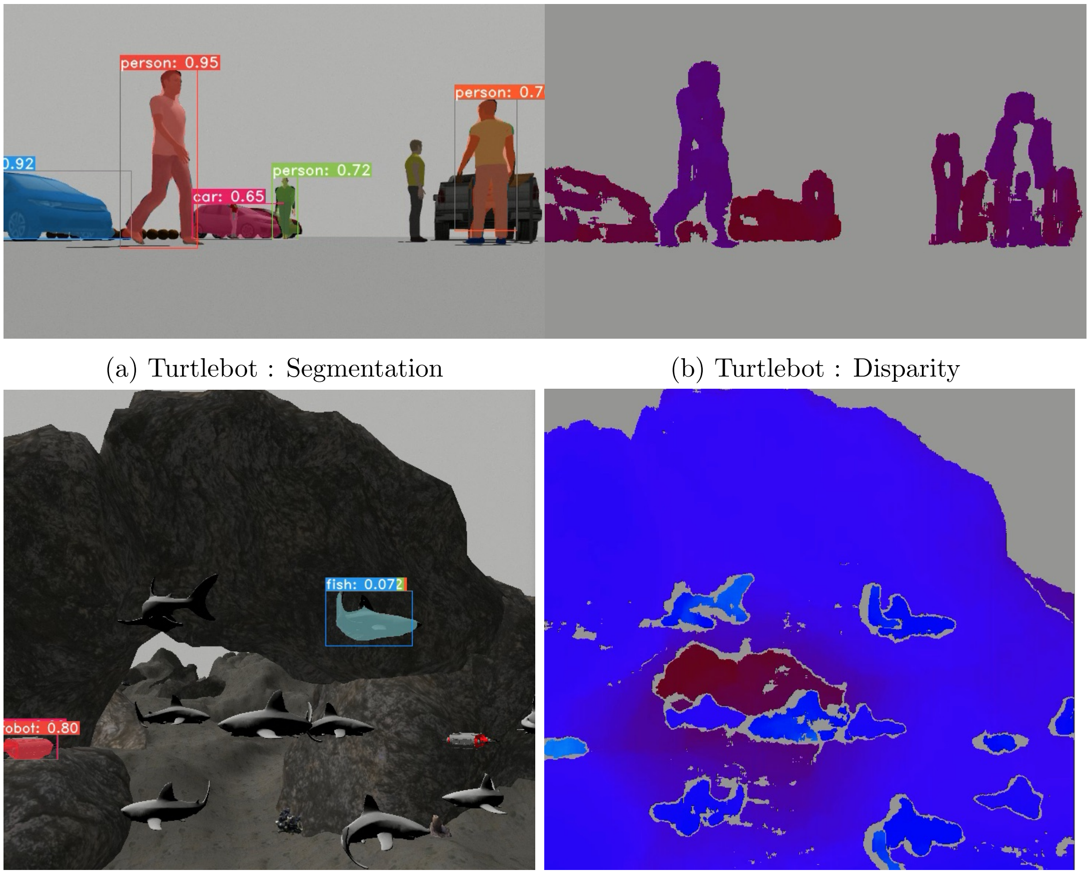
using the OpenCV [136] library.

Instance Segmentation

We train the instance segmentation model on a dataset of underwater imagery we collected, and also one that is openly available, as mentioned in Section 6.2.1. We train for $800,000$ epochs which took five days on an Nvidia Titan XP GPU. Table 6.1 shows the average mAP (Mean Average Precision) score over IoU (Intersection over Union) thresholds from 0.50 to 0.95. With higher accurate localization (higher IoU thresholds), the mAP values decrease. Overall, the mAP values of bounding boxes are slightly higher than those of segmentation masks.

Simulated Terrestrial Trials

We achieve $\approx$ 20 *fps* while simultaneously running simulation and algorithms on the Nvidia Titan XP GPU. Table 6.2a shows the average travel time (in simulation time) from the starting point to the goal for both SOAR and non-SOAR algorithms. Although both algorithms can reach the goal, the non-SOAR algorithm takes 14% longer time than when using SOAR. Samples from each case are shown in Fig. 6.4a and 6.4b. Fig. 6.5a and 6.5b show how Turtlebot understands scenes during its exploration. The SOAR algorithm reduces travel time by utilizing non-obstacle information (*i.e.*, *sports ball*) obtained from the instance segmentation model. This demonstrates how instance segmentation information can assist in efficient exploration while safely avoiding obstacles. Information from bounding box detection and *semantic* segmentation is not sufficient to provide detailed information for a robot to explore environments, particularly with *intra-class occlusion.*



(a) Turtlebot : Segmentation

(b) Turtlebot : Disparity

(c) Aqua : Segmentation

(d) Aqua : Disparity

Figure 6.5: Instance segmentation and disparity estimation from the view of robots. Fig. 6.5a and 6.5b are captured from the Turtlebot in the terrestrial world, and Fig. 6.5c and 6.5d are from the Aqua robot in the underwater world.

Simulated Underwater Trials

As our Gazebo world uses detailed hydrodynamic effects for the underwater simulation, we obtain ≈ 10 *fps* during our tests. Unlike the terrestrial case, the non-SOAR algorithm fails to reach the goal, as seen in Table 6.2b. This is because we terminate the non-SOAR algorithm in the following cases: 1) Aqua is heading in the wrong direction, 2) Aqua is stuck between rocks, and 3) it takes too long (≥ 2 minutes) to reach the goal. Sample cases from each scenario are captured in Fig. 6.4c and 6.4d. The cyan-colored robots shown are "obstacles", and immobile, to simulate a moving Aqua robot

Table 6.2: Terrestrial and Underwater Simulation Trial Results

	SOAR		non-SOAR	
	Travel time(s)	Goal	Travel time(s)	Goal
1	89	✓	99	✓
2	93	✓	98	✓
3	83	✓	120	✓
4	90	✓	99	✓
5	87	✓	100	✓
6	85	✓	98	✓
7	90	✓	101	✓
8	88	✓	97	✓
9	87	✓	98	✓
10	89	✓	101	✓
Avg	88.1		101.1	

(a) Turtlebot

	SOAR		non-SOAR	
	Travel time(s)	Goal	Travel time(s)	Goal
1	74	✓	97	✗
2	69	✓	75	✗
3	73	✓	88	✗
4	70	✓	120	✗
5	69	✓	91	✗
6	68	✓	94	✗
7	70	✓	122	✗
8	72	✓	111	✗
9	70	✓	111	✗
10	74	✓	99	✗
Avg	70.9		100.8	

(b) Aqua

(silver/red colors) avoiding other robots in the field. With the SOAR algorithm, Aqua is able to reach the goal (*i.e.*, under the *arch*) by ignoring the group of fish. The system ignores them because of the semantic knowledge that fish do not present a collision danger. Fig. 6.5c and 6.5d show a snapshot of Aqua's view from the trials. The results show that for underwater domains, obstacle avoidance without understanding the scene could significantly extend the travel time at best and fail to reach the goal at worst.

Bench Test

We achieved ≈ 5 *fps* inference speed while running the SOAR algorithm on the Jetson TX2. We expect to achieve faster inference speed in more capable mobile platforms (*e.g.*, AGX Xavier).

Limitations

When the inference produced by instance segmentation incorrectly classifies an obstacle as a non-obstacle, a collision can occur. Additionally, the inference time needs to be sufficiently fast to capture the objects' motion. In other words, if an object moves far faster than the inference speed, it could cause the obstacle avoidance algorithm to fail.

6.4 Conclusions

In this chapter, we propose an obstacle avoidance algorithm that incorporates both instance segmentation and depth information to perceive the robot's surroundings from only a pair of stereo images as an input. We are able to use the instance segmentation labels to inform a robot about which visible obstacles in its environment should be either avoided or ignored. Finally, we present our algorithm as a viable way to explore unstructured environments with the obtained visual information. We validate our algorithm on both terrestrial and underwater simulations. Quantitative results show that our algorithm can lead to efficient and intelligent robotic navigation decisions in unstructured environments, which can result in extending the duration of robot operations. In the next chapter, we introduce a method to localize AUVs to roughly estimate the location of underwater debris after detecting them.

Chapter 7

Depth-based Monte Carlo Localization for AUVs

For AUVs to navigate and operate missions autonomously, the ability to localize is essential. However, localizing AUVs underwater is a challenging and open problem due to the challenges mentioned in Chapter 1. In this chapter, we present bathymetry-based algorithms to localize AUVs without using pre-installed or expensive sensors. This method uses depth data from a pressure sensor and altitude data from a single-beam sonar as inputs; and localizes AUVs by applying four well-known Bayes filter-based methods: the Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), Particle Filter (PF), and Marginalized Particle Filter (MPF). The details of our approach are presented in the next sections.

7.1 Related Work

Underwater localization using landmark-based methods with acoustic sensors has been widely studied. For these methods, ranging-type sonars including the single-beam profiling and multi-beam varieties have been used [40]. Multi-beam and profiling sonars collect multiple measurements, and they can give more accurate results than single-beam sonars. Table 7.1 summarizes selected existing localization algorithms, along with the parameters of their state vector. The vector (ϕ, θ, ψ) represents the Euler angles, (u, v, w) is the AUV velocity in the body-fixed frame, and (b_x, b_y) is the velocity

Table 7.1: Selected existing localization algorithms

	Sensors	Parameters of state vector s	Algorithms
Teixeira et al. [140]	DVL, Single-beam sonar	$b = (b_x, b_y)$ $s = (x, y, b)$	PF
Fairfield and Wettergreen [138]	Multi-beam sonar	$q = (\phi, \theta, \psi, x, y, z)$ $s = (q, \dot{q}, \ddot{q})$	PF
Ura et al. [137]	Profiling sonar	$s = (x, y)$	PF
Williams and Mahon [142]	Single-beam sonar	$q = (x, y, z)$ $s = (q, \dot{q})$	PF
Meduna et al. [143]	Single-beam sonar	$s = (x, y)$	PMF
Kim and Kim [144]	Single-beam sonar	$p = (\phi, \theta, \psi)$ $q = (u, v, w)$ $s = (x, y, z, p, q)$	MPF
Ours	**Single-beam sonar**	$s = (x, y, z)$	**EKF, UKF, PF, MPF**

bias.

Although DVL, multi-beam sonar, and profiling sonar-based methods yield better results than those with single-beam sonar (*e.g.*, [137, 138, 139, 140]), such sensors can be prohibitively expensive. Single-beam sonars use a narrow acoustic projection to measure depth to the floor and are thus vulnerable to noise. However, they have been widely adopted to solve localization problems since they are among the most affordable acoustic sensors [141]. Williams and Mahon [142] proposed a localization algorithm based on the PF, but the computational burden of the algorithm is heavy. Meduna et al. [143] presented a point mass filter (PMF)-based algorithm, but it is limited to the (x, y) positions of an AUV. Kim and Kim [144] used a single-beam sonar with the MPF and estimated the 6-DOF position and orientation of an AUV along with the velocity. However, the algorithm requires a highly-accurate IMU, which can be expensive.

Several Bayes filter-based methods have been used to solve the localization problem [145] with sonar data. Among Bayes filters, the EKF and UKF have seen the most use in this domain (*e.g.*, [146, 147]). Karimi et al. [148] showed that the EKF can outperform the UKF when they estimate the position of an AUV by the sensor readings from DVL and inertial navigation system (INS). However, the UKF captures nonlinearity up to the second-order term in the state transition process [40], which in theory could outperform the EKF in similar applications. We thus compare the performance of EKF and UKF-based algorithms in AUV localization. Although the EKF and UKF exhibit high accuracy under Gaussian assumptions, they often fail to converge when the underlying distribution is multi-modal. The inherent nonlinearity of the underwater terrain and of AUV motions underwater makes it challenging for these methods to work reliably. To address these issues, the PF has been widely used (*e.g.*, [149, 150, 142, 151, 152, 137, 153, 139, 154, 155, 140]). However, the PF is computationally expensive and thus can be difficult to run on board AUVs for real-time localization. The MPF, on the other hand, has a lower computational cost and provides similar benefits to the PF, handling nonlinearity to some extent [156], thus making it potentially useful for underwater localization (*e.g.*, [157, 144]). Despite the strong potential shown by the above studies, localization with bathymetry data considering 3-DOF state vectors and using four Bayes filter-based algorithms (EKF, UKF, PF, and MPF) is yet to be extensively studied. The existing studies were conducted on diverse types of robots, and it is infeasible to do a fair comparison of the algorithms.

7.2 Problem Formulation

Localizing an AUV is a challenging problem, as mentioned in Chapter 1. Given bathymetry data of a target environment and measurements from AUV onboard depth and single-beam sonar sensors, we aim to localize an AUV with Bayesian filters (Fig. 7.1). Bayesian filters require motion and measurement models to estimate system states using the well-known propagate-predict-update process. We propose linear and mixed AUV motion models for updating the AUV position and a measurement model for collecting the depth and range measurements at each location. The motion models and measurement model are subsequently used to obtain AUV state estimates. To evaluate each filter's

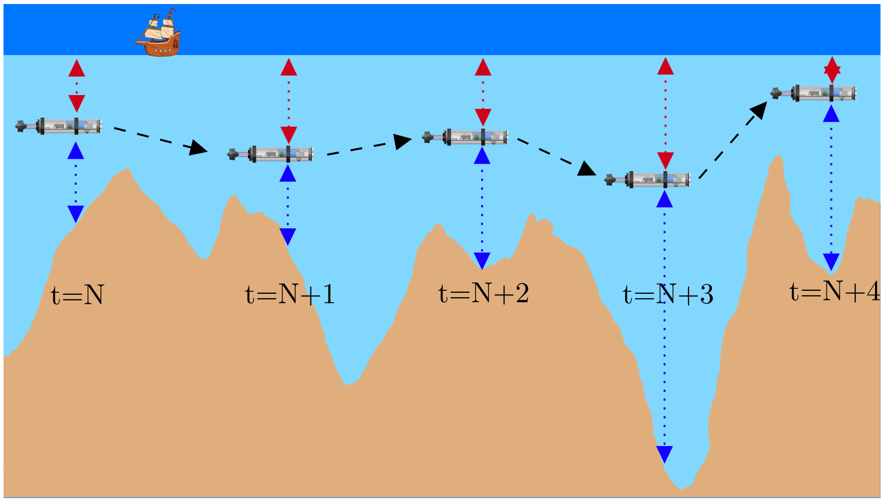


Figure 7.1: An example scenario of an AUV traveling underwater: A variation in the floor (light brown) is used to localize the AUV using depth (red) and range measurements (blue) at each time step t with our proposed approaches.

ability to estimate both linear and nonlinear states, we choose to use a linear and mixed motion model for the AUV. We use these models to analyze the performance of each filter in different water environments.

7.2.1 Motion Model

A general discrete time state-space model can be represented as Eq. 7.1 to formulate the AUV localization problem where x_t is a state vector, u_t is a control in put, and y_t is a measurement. In our work, only the 3D position of an AUV is included in the state vector. f and h can be either linear or nonlinear functions. q_t and r_t represent the noise from motion and measurements. We assume the noise is normally distributed [140, 143, 144]. The model in Eq. 7.1 is used for the EKF, UKF, and PF-based localization algorithms. The model for the MPF-based localization algorithm is introduced in Section 7.3.4.

$$\begin{cases} x_t = f(x_{t-1}, u_t) + q_t \\ y_t = h(x_t) + r_t \end{cases} \tag{7.1}$$

The state vector and control inputs are defined as follows:

$$x_t = \begin{bmatrix} p_{x,t} & p_{y,t} & p_{z,t} \end{bmatrix}^T \tag{7.2}$$

$$u_t = \begin{bmatrix} v_{x,t} & v_{y,t} & v_{z,t} \end{bmatrix}^T \tag{7.3}$$

where $\mathbf{p} = \begin{bmatrix} p_{x,t} & p_{y,t} & p_{z,t} \end{bmatrix}^T$ are the x, y, and z components of the robot's *position* in three-dimensional space at time t. We also use the term 'pose' to refer to the state vector x_t. The AUV motion models are defined in Eqs. 7.4 and 7.5.

Linear motion model

All state variables are updated linearly.

$$f(x_t, u_t) = x_t + u_t * dt \tag{7.4}$$

Linear/Nonlinear mixed motion model

Underwater current often causes an AUV to deviate from its original trajectory, which generates irregular movements of the AUV. To simulate such movements without using the Euler angles [158] explicitly, we propose a linear/nonlinear mixed motion model using the bathymetry data. Among the three state variables in the state vector (Eq. 7.2), $p_{x,t}$ and $p_{y,t}$ are updated based on the altitude $L(p_{x,t}, p_{y,t})$ at position $(p_{x,t}, p_{y,t})$. Therefore, this model will cause an AUV to have faster motions in x and y directions from positions with higher altitudes. Unlike the other two variables, the state variable $p_{z,t}$ is updated linearly as in the linear motion model. In Eq. 7.5, a, a_d, a_{off}, b, b_d, and b_{off} are constants specific to each water body such that the AUV is able to navigate in each without collision with their "shorelines".

$$f(x_t, u_t) = \begin{bmatrix} p_{x,t} + a\left[\frac{L(p_{x,t}, p_{y,t})}{a_d} + a_{off}\right]dt \\ p_{y,t} + b\left[\frac{L(p_{x,t}, p_{y,t})}{b_d} + b_{off}\right]dt \\ p_{z,t} + v_{z,t} * dt \end{bmatrix} \tag{7.5}$$

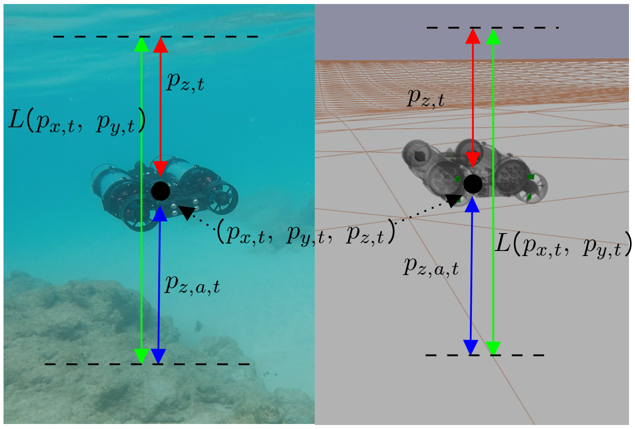


Figure 7.2: Visual representation of AUV location in a body of water. The surface and bottom of the body are indicated by the dashed black lines. $L(p_{x,t}, p_{y,t})$: height, $p_{z,t}$: depth, $p_{z,a,t}$: altitude at $(p_{x,t}, p_{y,t})$, (Left): LoCO deployed in an open water environment, (Right): LoCO simulated in Gazebo.

7.2.2 Measurement Model

The measurement function h is the same for both the linear and mixed models.

$$h(x_t) = \begin{bmatrix} p_{z,t} & p_{z,a,t} = L(p_{x,t}, p_{y,t}) - p_{z,t} \end{bmatrix}^T \tag{7.6}$$

L is bathymetry data (depths for each x and y location on a grid), $p_{z,t}$ represents the depth of the AUV from the surface measured by the pressure sensor, and $p_{z,a,t}$ represents the range measurement of the single-beam sonar on the AUV. Therefore, the sum of $p_{z,t}$ and $p_{z,a,t}$ is the altitude $L(p_{x,t}, p_{y,t})$ at the position $(p_{x,t}, p_{y,t})$ as shown in Fig. 7.2.

7.3 Methodology

To solve the underwater robot localization problem under highly nonlinear perturbation of AUV motion (Eq. 7.5), nonlinear Bayes filter algorithms are essential. The EKF and UKF are widely used to handle nonlinear state estimation with the assumption that the state variables follow the Gaussian distribution, but they could fail when the distribution is not Gaussian [159]. The PF [149] is resilient to various types of noise, but it is computationally expensive. The MPF [155] uses the PF for nonlinear state variables and the KF for linear state variables because the KF is an optimal filter for estimating linear state variables.

7.3.1 Extended Kalman Filter

To approximate a nonlinear system, the EKF takes the first-order of the Taylor series expansion [160]. Linear matrices in the KF are replaced with the Jacobians to make predictions. The Jacobians for the mixed motion model in Eq. 7.5 and the measurement model in Eq. 7.6 are shown in Eq. 7.7 and Eq. 7.8, respectively. The EKF requires the following inputs: state vector, state covariance, control input, and measurements. With these inputs, the EKF uses the following well-known two-step approach:

- *Prediction Step*: In this step, the state vector is updated based on the motion model and control input. Once it is updated, the Jacobian, state covariance, and noise covariance are used to find the state vector and state covariance matrix for the next time step.
- *Correction Step*: The Kalman gain is calculated using the Jacobian, state covariance, and measurement noise covariance matrices. The calculated Kalman gain is then used to refine the state vector from the prediction step.

The EKF assumes that state variables follow a Gaussian distribution with a single mode. Due to this assumption, the EKF is computationally efficient, and each update takes $O(d^3)$ where d is the dimension of the state x_t [161]. The EKF is directly applied to Eq. 7.1 for linear and mixed motion cases to localize AUVs using the depth and range measurements.

$$F_t = \begin{bmatrix} 1 + \frac{a}{a_d}\left[\frac{\partial L(x,y)}{\partial x}\right]dt & \frac{a}{a_d}\left[\frac{\partial L(x,y)}{\partial y}\right]dt & 0 \\ \frac{b}{b_d}\left[\frac{\partial L(x,y)}{\partial x}\right]dt & 1 + \frac{b}{b_d}\left[\frac{\partial L(x,y)}{\partial y}\right]dt & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.7}$$

$$H_t = \begin{bmatrix} 0 & 0 & -1 \\ \frac{\partial L(x,y)}{\partial x} & \frac{\partial L(x,y)}{\partial y} & 1 \end{bmatrix} \tag{7.8}$$

7.3.2 Unscented Kalman Filter

The UKF [162] is another approach to estimate a nonlinear system using the Unscented Transform instead of the Taylor series. It uses a sampling approach to capture the mean and covariance of Gaussian random distributions. Then, the UKF propagates the points through the true nonlinear system. In this way, the UKF can handle higher degrees of non-linearity than the EKF. The UKF requires the same inputs as the EKF, along with some additional parameters, namely $n, \alpha, \beta,$ and κ, described below. Like the EKF, the UKF has a two-step estimation process:

- *Prediction Step*: The UKF chooses $2n + 1$ sigma points from the Gaussian distribution where n is the number of dimensions. The UKF then passes these sigma points through the motion model f. The parameters $\alpha, \beta,$ and κ are used to determine weights for each sample and the distribution of the sigma points.

- *Correction Step*: The state vector from the prediction step is used to generate sigma points. Their measurements, noise covariance, and state covariance matrices are used to calculate the Kalman gain. As in the EKF, the state vector is updated using the Kalman gain. With this, the UKF approximates a Gaussian distribution with the sigma points.

In this way, the UKF can create a more accurate approximation of the Gaussian distribution than the EKF. Since the estimation is based on the Gaussian distribution assumption, it may not work for multi-modal distributions. In most cases, the UKF shows notable improvements compared to the EKF. The computational complexity of the UKF is close to the EKF [161]. The UKF is also directly applied to Eq. 7.1 for linear and mixed motion cases.

7.3.3 Particle Filter

The PF [152] implements the Bayes filter algorithm using sequential Monte Carlo methods. Unlike the EKF and UKF, the PF does not require any assumptions regarding the distribution. Instead, it uses N *particles*, representing potential locations of the state x_t, to approximate the distribution of the state x_t. The more particles there are, the more accurate the approximation of the distribution. The PF can handle distributions

Algorithm 1 Depth-based PF Localization

```
**D-PFL main**
$L$ = Target waterbody
$N$ = The number of particles
$x_{init}$ = Initial pose of a robot
$x_1$ = **Initialize_around_pose**$(L, N)$
$z_t$ = Sensor measurements.
$u_t$ = Control input
**for** $t = 1, ..., T$ **do**
    $x_p = x_t$
    **for** $m = 1, ..., N$ **do**
        $x(m, :)$ = **motion_update**$(u_t, x_p(m, :), L)$
        $w(m)$ = **sensor_update**$(z_t, x(m, :), L)$
    **end for**
    $w_{total} = sum(w)$
    **for** $m = 1, ..., N$ **do**
        $w(m) = w(m)/w_{total}$
    **end for**
    $x_t$ = **resample_particles**$(x(m, :), w, L, rand)$
    $w_t = w$
    $est\_pose$ = **PFL_get_pose**$(x_t, w_t)$
**end for**
**return** $est\_pose$
```

with high nonlinearity and multiple modes. However, the computational complexity of the PF for each update is $O(Nd^2)$ where d is the dimension of the state x_t, meaning that the runtime of the PF scales linearly with the number of particles used. When N is much larger than d, the PF can be much slower than the EKF and UKF [154], which is the situation in our specific case. This computational burden is often the main drawback of the PF for real-time implementation.

We propose a depth-based PF localization (d-PFL) in Algorithm 1; *i.e.*, the PF uses the AUV's *depth* at each iteration as its *measurement*. **PFL_update()** takes a bathymetry map, depth measurements, and control inputs and returns propagated particles representing the AUVs position in three dimensions and their weights. To evaluate weights w_t, we use the multivariate Gaussian distribution as shown in Eq. 7.9.

$$w_t = w_{t-1} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{7.9}$$

During the update process, only the particles not over the “shoreline” will have non-zero weights (*i.e,* inside a bathymetry map). Once the weights for the particles are calculated, they are normalized to ensure that they sum to 1. Then, particles are resampled based on their weights. To avoid a situation where all the particles are trapped in some local maxima in a bathymetric environment with similar or repeating profiles, some of the N *particles* are sampled randomly at each time step. Although this can degrade the accuracy of the algorithm, it decreases the chance of incorrect estimations occurring. The pose of the AUV is estimated (Eq. 7.10) once the propagated particles and the corresponding weights are updated.

$$\hat{x} = \sum_{m=1}^{M} w^{[m]} x^{[m]} \tag{7.10}$$

7.3.4 Marginalized Particle Filter

The MPF was proposed to reduce the computational complexity of a particle filter while retaining a similar performance when the model has a linear substructure [159]. The core idea of the MPF is to *marginalize* linear state variables from the state vector and use the KF to estimate those variables. The PF is then used to estimate the remaining nonlinear state variables. The computational complexity of the MPF is defined in [163] and can be simplified to $O(d_n^3 N)$ for our case where d_n is the dimension of the nonlinear state variables.

To apply the MPF [156], the model in Eq. 7.1 is separated into linear and nonlinear state variables as shown in Eq. 7.20 and 7.21. The motion model noise q_t^n, q_t^l, and the measurement model noise r_t are assumed to be Gaussian with zero mean. The matrices A, C, and G are determined by the motion model (Algorithm 2).

$$x_t = \begin{bmatrix} x_t^n \\ x_t^l \end{bmatrix} \tag{7.20}$$

Algorithm 2 MPF

```
Initialize particles
    Initialize nonlinear state variables
        $x^{n,(i)}_{0|-1} \sim p(x^n_0)$    (7.11)
    Initialize linear state variables
        $\{x^{l,(i)}_{0|-1}, P^{(i)}_{0|-1}\} = \{\bar{x}^l_0, \bar{P}_0\}$    (7.12)
for $t = 1, ..., T$ do
    PF measurement update
        Evaluate the weights
            $w^{(i)}_t = p(y_t | X^{n,(i)}_t, Y_{t-1})$    (7.13)
        Estimate nonlinear state variables
            $\hat{x}^n_{t|t} = \sum_{i=1}^{N} w^{(i)}_t x^{n,(i)}_{t|t-1}$    (7.14)
        Resample particles
            $Pr(x^{n,(i)}_{t|t} = x^{n,(j)}_{t|t-1}) = \tilde{w}^{(j)}_t$    (7.15)
    KF measurement update
        Estimate linear state variables
            $x^{l,(i)}_{t|t} = x^{l,(i)}_{t|t-1} + K_t(y_t - h_t - C_t \hat{x}^l_{t|t-1})$    (7.16)
            $\hat{x}^l_{t|t} = \sum_{i=1}^{N} w^{(i)}_t x^{l,(i)}_{t|t}$    (7.17)
    PF prediction
        Propagate nonlinear state variables
            $x^{n,(i)}_{t+1|t} \sim p(x^n_{t+1|t} | X^{n,(i)}_t, Y_t)$    (7.18)
    KF prediction
        Propagate linear state variables
            $\hat{x}^l_{t+1|t} = \bar{A}^l_t \hat{x}^l_{t|t} + G^l_t (Q^{ln}_t)^T (G^n_t Q^n_t)^{-1} z_t$
                $+ f^l_t + L_t(z_t - A^n_t \hat{x}^l_{t|t})$    (7.19)
end for
```

Table 7.2: Lake Bathymetry Information

Lake	Height (m)		Size ($\times 10^6 m^2$)
	Max.	Avg.	
Bde Maka Ska	-27.43	-10.00	1.71
Nokomis	-9.99	-4.11	0.84
Hiawatha	-9.45	-4.00	0.22
Harriet	-26.42	-9.75	1.39
Turtle	-8.53	-3.42	1.83
Howard	-10.97	-4.89	3.00
Waverly	-21.49	-7.59	1.99
Pulaski	-26.50	-11.04	2.98

$$\begin{cases} x_{t+1}^n = f_t^n(x_t^n) + A_t^n(x_t^n)x_t^l + G_t^n(x_t^n)q_t^n \\ x_{t+1}^l = f_t^l(x_t^n) + A_t^l(x_t^n)x_t^l + G_t^l(x_t^n)q_t^l \\ y_t = h_t(x_t^n) + C_t(x_t^n)x_t^l + r_t \end{cases} \tag{7.21}$$

In our case, the ratio $\frac{N(k)}{N_{PF}}$ is 1.1 where $N(k)$ is the number of particles that can be used for the MPF and N_{PF} is the number of particles used for the standard PF. This ratio means that the MPF can use 10% more particles than the PF while having the same computational complexity as the standard PF. However, the EKF and UKF are still more computationally efficient than the MPF, albeit less accurate.

The MPF localization algorithm is shown in Algorithm 2 along with selected simplified equations where Q and P are covariance matrices. The equations can be found in detail in [156].

7.4 Simulation Development and Experimental Setup

We developed simulations with real-world bathymetry data to evaluate the proposed localization algorithms. To this effect, we exploit the capabilities of the Robot Operating System (ROS) [164] in creating realistic simulations using Gazebo, to reduce overhead when transitioning to AUV field deployments. Our approach enables the algorithms to be used in our simulated evaluation, and then deployed in the field with minimal changes.

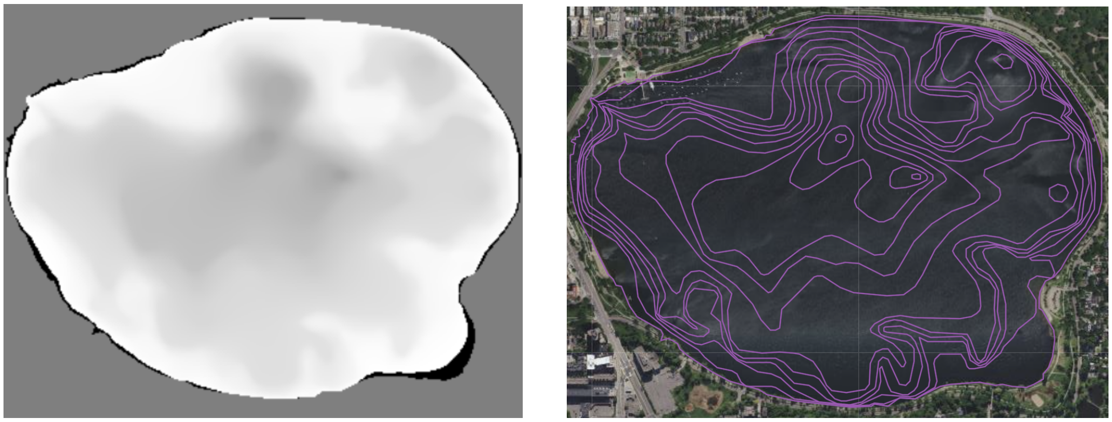

(a) (b)

Figure 7.3: Visualization of raw bathymetry data: (a) in tagged interchange file format (TIFF) of Bde Maka Ska, and (b) with a satellite image and lake contour. Source: Minnesota Department of Natural Resources. Darker pixels represent locations with deeper depths.

7.4.1 Bathymetry Data Collection

The following lakes located in Minnesota, USA were chosen as test locations: Bde Maka Ska[1], Lake Nokomis, Lake Hiawatha, Lake Harriet, Lake Turtle, Lake Howard, Lake Waverly, and Lake Pulaski (Table 7.2). We selected lakes that have a wide range of maximum altitudes, average altitudes, and sizes. Additionally, the lakes are well-studied, have an undulating floor, and are easy to access for future field studies. The raw bathymetry data was acquired from the Minnesota Department of Natural Resources (MN DNR) [165]. Fig. 7.3a shows an example of bathymetry data provided in a TIFF format. The original data contains the lake altitudes (measured in feet) every $5m$ in a grid pattern across the lakes.

7.4.2 Robotic Simulation Development

As mentioned, we use the Gazebo [166] simulation tool and ROS integration for this work (Fig. 7.4). Specific information on the various components of the simulation is provided below.

[1] No preceding ”Lake” is used for Bde Maka Ska, which was formerly referred to as Lake Calhoun and had its name restored in 2018.

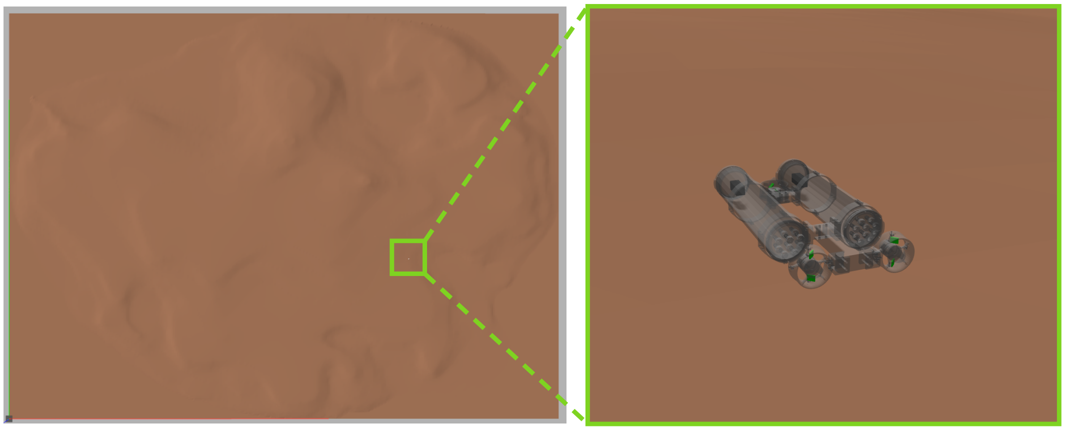


Figure 7.4: One example of simulated lakes (Bde Maka Ska) and LoCO AUV. From the top view, LoCO appears as an white dot in the left image due to the difference in actual scale between the lake and LoCO AUV. The right image shows a simulated LoCO AUV in Gazebo.

Bathymetry Data Processing for Simulation

Though Gazebo provides built-in ways to import raw bathymetry data (TIFF format), our preliminary testing shows that using Standard Triangle Language (STL) format provides a more accurate lake bottom profile. With Gazebo's built-in bathymetry methods, the water column depth at each point is only rounded to the nearest meter. We convert the raw data from the TIFF format to the STL format as follows:

1. Extract an array from the raw data, representing the altitude of the relevant lake at each grid location.
2. Convert the altitude from feet to meters.
3. Create a model STL file with this array data.

The STL file is created with a process that linearly interpolates solid surface material between neighboring bathymetric data points in a triangle mesh format. With this, MATLAB can also be used to interpolate and reduce the bathymetry data grid size to 1m from 5m. For any triangle already in the $5m$ mesh, the corresponding mathematical plane equation can be determined based on its vertices. Any of the finer 1m grid points that land within the triangle can have their depths calculated from this plane equation. The STL files are then imported into the Gazebo simulation world as solid surface models, with a solid ground plane at the top of the model to represent the surface of the lakes.

AUV and Sensors in Our Simulation

We use the LoCO AUV [167] Gazebo model from our previous work to develop the simulated experiments. We add two single-beam laser sensors to the LoCO simulation. Each laser sensor is a proxy for a depth sensor and an acoustic sensor, respectively. Both provide sensor measurements at a rate of 1 Hz. The depth sensor could be simulated directly by simulating water pressure instead of using the laser sensor. However, this would significantly complicate the simulation design, and developing hydrodynamics for the simulation is out of the scope of this work. Using the laser as a proxy is a convenient alternative since it provides sensor readings used to validate our approach.

Implementation of the Proposed Algorithms

Gazebo interfaces with ROS using individual processes called ‘nodes’ to simulate and test algorithms for various robotic functions. For our study, we develop three programs to facilitate the testing of the four localization filters. The first is the velocity control node. This program reads the simulation time from Gazebo and sends out predefined velocity commands to the robot according to the desired velocity profiles. These commands are also received by the second program, which is the particular filter estimating the position of the robot. The filter programs are triggered to run when they receive the depth and range measurements from the robot’s two sensors. The filter uses this state data, the velocity command, and the data from the previous time step to estimate the current position of the robot according to the algorithm in use. Position estimation data from the filter is received by the third program, which stores all data relevant to the current time step of the estimation filter. This information is stored in a final data file for analyzing each filter’s performance during simulations and providing an overall assessment of the simulated world.

Table 7.3: Model parameters

Parameter	Value
No. of particles for the PF, N_{PF}	5000
No. of particles for the MPF, N_{NPF}	500
Motion noise cov., Q (m)	$0.005^2 \begin{bmatrix} v_x^2 & 0 & 0 \\ 0 & v_y^2 & 0 \\ 0 & 0 & v_z^2 \end{bmatrix}$
Measurement noise cov., R (m)	$0.005^2 \begin{bmatrix} depth^2 & 0 \\ 0 & alt.^2 \end{bmatrix}$
Initial uncertainty cov., P (m)	$\begin{bmatrix} 1^2 & 0 & 0 \\ 0 & 1^2 & 0 \\ 0 & 0 & 1^2 \end{bmatrix}$

7.4.3 Simulation Settings

The goal of this study is to evaluate each proposed localization algorithm with real bathymetry data. For the several reasons introduced in Chapter 1, it is extremely difficult to evaluate AUV localization in physical, open-water testing, as ground truth to evaluate against is nearly impossible to obtain [168]. No highly accurate localization technique is readily available to provide ground truth for our algorithm, as GPS does not work underwater and options which require equipment such as USBL and DVL devices are prohibitively expensive. As our work also requires variations in bathymetric data, evaluations in a closed-water (*e.g.*, pool) environment is infeasible. Thus, to quantify the accuracy and efficiency of the algorithms, we use our proposed simulation to evaluate the performances of each filter's position estimates against the simulated AUV's motion, since accurate ground truth can easily be collected in simulation. The linear and mixed motion models are designed to test the performance of each localization algorithm on the real-world bathymetry data from different lake environments. As real-world AUV motion is seldom linear, it is important to evaluate the mixed motion models (which contain linear and nonlinear components) as well, as nonlinear motion may cause some localization algorithms to perform poorly. The non-linear motion model will also help account for perturbations to AUV motion arising from waves and general hydrodynamic factors.

Table 7.3 shows the model parameters for the simulation. The control inputs for each algorithm and lake are separately designed due to the lakes' different sizes and altitudes. The parameters are defined for the linear motion in Eqs. 7.3 and 7.4, and the mixed motion in Eq. 7.5. For the mixed model, x and y are nonlinear state variables, and z is a linear state variable. We measure the performance of our algorithms on an AMD Ryzen 7 5800X 8-Core processor with 3.8GHz clock speed running Ubuntu 20.04.2 LTS with 32GB of DDR4 memory with ROS Noetic and Gazebo 11.3. We select five paths per motion with the four Bayesian filters for total of eight lakes, accumulating to a total 80 unique simulated trials.

7.5 Results and Discussion

The results of our trials are summarized in Table 7.4 and Appendix A. We use the root-mean-square error (RMSE) in Eq. 7.22 as a metric to evaluate the performance for each axis of motion where T is the number of steps, p_g is the ground truth, and p_e is the estimated position.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(p_g - p_e)^2}{T}} \tag{7.22}$$

The widths and lengths of the lakes are significantly larger than the depths, meaning that the velocity of x and y axes is generally two orders of magnitude larger than the velocity of the z axis in our simulation. This yields notably smaller errors in the z axis than in the other axes (Table 7.4). We also evaluate positional errors ($|p_g - p_e|$) relative to velocity for each axis.

In Fig. 7.5, we show one particular trial in Bde Maka Ska as an example. For both the linear and mixed cases, the MPF generally performed well in this trial. We observe degraded performance from the PF and EKF evaluations in the linear case; and from the EKF and UKF evaluations in the mixed case compared to the other filters. The EKF had the worst performance overall in Bde Maka Ska. The PF outperforms the other filters in the mixed motion case, but it was the least accurate of the four filters in the linear case. A likely cause of the degraded performance of the PF is that the randomly selected particles can cause the estimation to diverge from the ground truth when a water body's bathymetry data does not have sufficient variation.

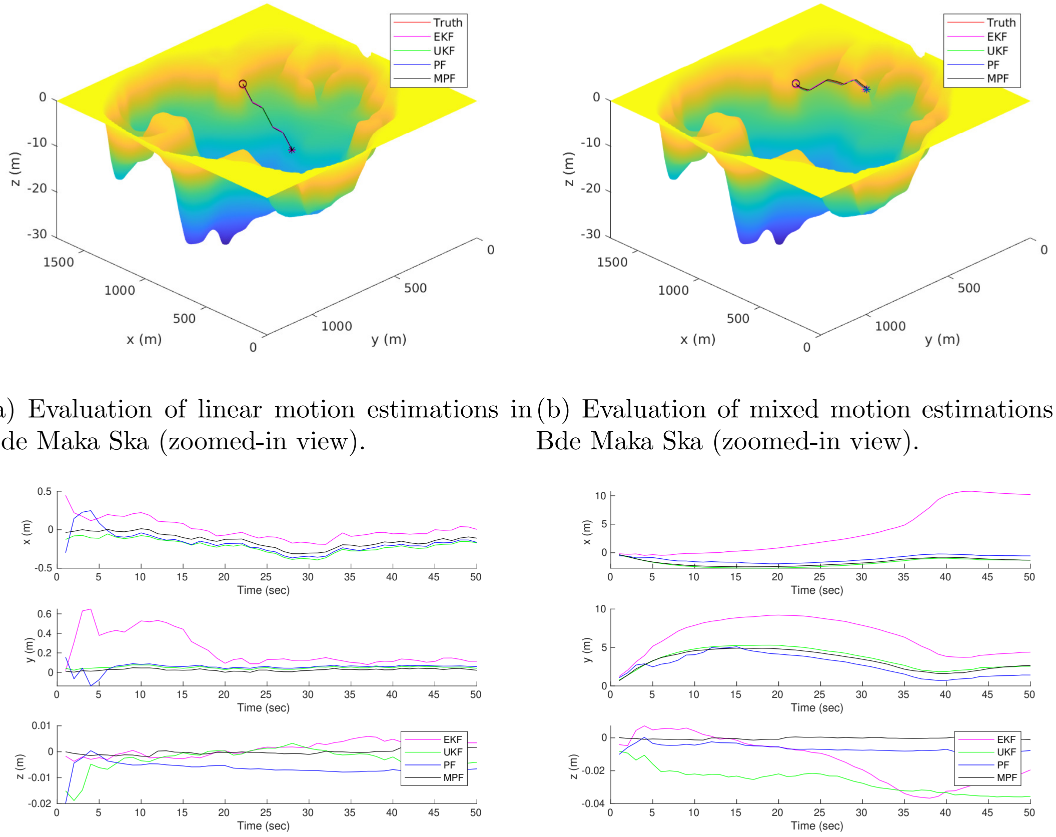
Table 7.4: Localization performance evaluation for an AUV; bathymetry data from the MN DNR.

Lake	Method	Linear motion model							Mixed motion model						
		Iter. Time (ms)	$RMSE(m)$			$RMSE(\%)$ w.r.t vel			Iter. Time (ms)	$RMSE(m)$			$RMSE(\%)$ w.r.t vel		
			x	y	z	x	y	z		x	y	z	x	y	z
Bde.	EKF	20.5	0.151	0.309	0.005	0.863	5.831	2.683	20.1	5.040	5.990	0.016	53.509	182.882	8.683
Bde.	UKF	0.3	0.185	0.035	0.003	1.087	0.666	1.534	0.3	1.786	3.488	0.019	33.166	110.868	12.082
Bde.	PF	115.0	0.395	0.540	0.006	2.492	10.286	3.861	113.1	0.906	3.000	0.009	15.796	88.855	5.816
Bde.	MPF	61.3	0.200	0.041	0.003	1.277	0.933	2.332	62.3	1.801	3.510	0.003	33.475	112.669	1.615
Nok.	EKF	24.0	0.349	0.295	0.004	1.523	2.687	1.831	22.3	2.007	1.282	0.004	10.530	12.605	2.254
Nok.	UKF	0.3	3.952	1.940	0.031	18.857	18.523	19.373	0.3	5.871	2.908	0.031	32.075	31.783	19.110
Nok.	PF	113.9	0.553	0.182	0.007	2.623	1.667	4.465	115.0	3.437	2.016	0.008	19.068	22.464	5.311
Nok.	MPF	61.1	0.216	0.055	0.003	0.931	0.503	1.582	62.3	2.806	1.378	0.002	15.462	15.188	1.470
Hia.	EKF	3.0	0.289	0.240	0.004	2.337	4.233	1.111	2.9	0.641	0.330	0.004	5.320	5.727	1.121
Hia.	UKF	0.3	0.121	0.037	0.004	1.020	0.692	1.001	0.3	0.708	0.345	0.004	4.965	4.912	1.125
Hia.	PF	112.5	0.472	0.272	0.004	4.224	4.709	1.334	112.4	1.073	0.509	0.006	8.539	8.873	1.728
Hia.	MPF	62.5	0.089	0.022	0.003	0.714	0.390	0.926	62.3	0.711	0.346	0.005	4.988	4.930	1.417
Har.	EKF	17.4	0.236	0.152	0.003	1.405	1.5170	0.925	17.2	1.385	0.822	0.003	9.367	9.844	0.856
Har.	UKF	0.3	0.812	0.406	0.003	6.820	6.931	1.222	0.3	1.610	0.810	0.003	11.460	11.088	1.077
Har.	PF	112.9	0.290	0.452	0.005	1.640	5.128	1.668	112.7	1.665	1.372	0.006	10.496	15.660	1.708
Har.	MPF	61.3	0.143	0.035	0.003	0.749	0.401	0.891	62.2	2.103	1.058	0.003	11.550	11.192	1.034
Tur.	EKF	24.7	0.355	0.211	0.003	3.090	1.993	0.831	22.4	0.479	0.347	0.004	3.366	12.087	0.930
Tur.	UKF	0.3	0.128	0.075	0.004	1.126	0.711	1.221	0.3	0.419	0.090	0.004	2.983	3.006	0.966
Tur.	PF	112.5	0.516	0.251	0.006	4.815	2.399	1.893	115.6	0.948	0.723	0.006	7.521	28.647	1.816
Tur.	MPF	62.0	0.091	0.041	0.003	0.775	0.366	0.763	62.2	0.451	0.098	0.003	3.210	3.295	0.864
How.	EKF	94.4	0.423	0.315	0.007	4.010	2.447	1.715	84.0	1.622	0.971	0.006	12.304	8.386	1.747
How.	UKF	0.3	0.115	0.074	0.005	0.991	0.640	1.351	0.3	1.771	1.807	0.006	13.722	14.078	1.789
How.	PF	113.2	0.745	0.747	0.008	6.478	7.003	2.330	115.5	1.730	1.820	0.007	13.379	14.387	1.971
How.	MPF	61.4	0.127	0.076	0.006	1.135	0.719	1.809	61.3	1.831	1.867	0.004	14.265	14.617	1.180
Wav.	EKF	37.1	0.154	0.100	0.004	2.877	1.609	0.985	40.5	0.649	0.572	0.004	6.526	9.100	1.034
Wav.	UKF	0.3	0.050	0.025	0.004	0.853	0.456	1.246	0.3	1.040	0.710	0.004	10.464	10.885	1.245
Wav.	PF	115.5	0.309	1.113	0.006	5.666	19.694	1.770	116.3	0.995	0.564	0.006	11.545	9.238	1.805
Wav.	MPF	61.7	0.059	0.033	0.006	1.029	0.612	1.727	61.5	1.057	0.721	0.005	10.670	11.084	1.515
Pul.	EKF	69.8	0.118	0.103	0.004	1.990	1.775	0.906	62.1	2.120	2.346	0.005	17.248	16.117	1.546
Pul.	UKF	0.3	0.055	0.028	0.003	0.936	0.514	0.792	0.3	3.781	3.708	0.009	25.876	25.425	2.775
Pul.	PF	113.3	0.563	0.865	0.004	10.618	16.022	1.087	113.9	2.951	2.978	0.006	20.136	20.368	1.679
Pul.	MPF	61.8	0.076	0.052	0.006	1.411	1.003	1.930	61.3	3.764	3.691	0.002	25.756	25.302	0.665

The intensity of each color is proportional to the error. Red, green and blue colors are used for the x, y, and z-axis, respectively. Grey color represents the percentage errors across all the axis.

7.5.1 Linear Motion Case

Looking at the overall results from the linear motion case, the EKF does not always perform best, but it provides reliable and relatively accurate results with lower computational complexity. While the UKF estimates the position of LoCO with the smallest RMSE for most lakes, it is not reliable, since its performance varies widely between



(a) Evaluation of linear motion estimations in Bde Maka Ska (zoomed-in view).

(b) Evaluation of mixed motion estimations in Bde Maka Ska (zoomed-in view).

(c) Errors (m) of linear motion estimations from each filter on x,y,z axis.

(d) Errors (m) of mixed motion estimations from each filter on x,y,z axis.

Figure 7.5: Localization performance of the four algorithms for an AUV with an altimeter and depth sensor within Bde Maka Ska using bathymetry data (Start:○ End:∗).

lakes. For instance, the average UKF RMSE of the x,y, and z axes is ten times bigger than the average MPF RMSE in Lake Nokomis. The average UKF error is three times greater than the average EKF error in Lake Harriet. The PF's performance is the worst overall, particularly in Lake Howard and Waverly Lake since the paths chosen for the two lakes have fewer variations in altitude compared to the paths in the other lakes. Although the MPF does not presents the smallest RMSE in every lake, it does not show significantly degraded accuracy in any of the lakes. The MPF generally gives accurate and reliable results, and it is computationally cheaper than the PF. Overall, the MPF is the most reliable and accurate filter based on the results of these experiments. It is worth mentioning that the EKF is a good option if an AUV does not have sufficient computational power for the MPF and if high accuracy is not required. The PF can be used for localization if the bathymetry data has enough variation in altitude and has an asymmetrical structure, provided that the AUV has enough computational power.

7.5.2 Mixed Motion Case

Unlike in the linear motion cases, the EKF is much more unreliable for mixed motion cases, achieving the least accurate estimation out of the four filters in some lakes while performing best in other lakes. For example, the EKF RMSE is larger than twofold of the other filters' RMSE in Bde Maka Ska. The UKF performs poorly in some lakes due to the high nonlinearity of the motion. However, it needs the least amount of computational resources among all the filters. The PF displays the same issue in mixed motion cases that it has in the linear motion cases: the estimations diverge when there are not enough variations in bathymetry data (*i.e.*, Lake Hiawatha, Lake Nokomis, and Turtle Lake). Similarly to the linear cases, in the mixed motion cases the MPF gives reliable and accurate results overall. Therefore, our recommendation for using the MPF remains the same for mixed motion cases, but we do not recommend using the EKF in these cases.

7.5.3 Discussion

The MPF consistently produces reliable and accurate results for both the linear and mixed motion cases while maintaining relatively low computational costs. If an AUV needs a well-rounded algorithm for the localization problem, the MPF is the best filter among those we have evaluated. However, if the bathymetry data of a lake has sufficient variation and the task requires high accuracy, then the PF is a better option, albeit at a higher computational cost. Determining what constitutes enough variation is a complicated question affected by several factors: the size of littoral zone, the paths of the AUV, and the average and maximum altitude of the lake. We perform our analysis on lakes that have a wide range of such factors. Defining the magnitude of variation quantitatively is a problem that should be explored in future work. While PF and MPF have high accuracy in some cases, they require memory to store the state of their estimated particles, making them computationally expensive. The MPF is significantly less computationally demanding than the PF, but still requires more memory and processing power than the EKF and UKF. Therefore, for an AUV with low computational power, the EKF could be the best filter for localization, but lower accuracy should be expected, particularly if the AUV is not constrained to linear motion.

7.6 Conclusion

In this chapter, we present several affordable AUV localization algorithms with low-cost sensors (*i.e.*, depth sensor and single-beam sonar) and their evaluation results. We first developed the localization algorithms using four Bayesian filters (EKF, UKF, PF, and MPF), with single-beam sonar altimeter and depth sensor readings as inputs. We then built a realistic simulation for an AUV using Gazebo with real-world bathymetry data. With the simulation and ROS, we evaluated the performance of the algorithms in various aquatic environments and with multiple robot motion policies. Since our algorithms are implemented using ROS, they are easily deployable on any ROS-supportive AUVs. Our approach will reduce the cost and time to develop algorithms for field robots when access to fields is limited. This chapter concludes Part II, and we will continue to discuss our proposed algorithms for enabling effective Human-Robot Collaboration in Part III.

Part III: Algorithms for effective human-robot collaboration

AUVs can face extreme perception challenges when they operate in fully unknown environments or observe previously unseen objects. In these situations, human-robot collaboration (HRC) can be used to augment robotic perception underwater. Additionally, HRC can reduce risks posed to divers by sending AUVs to patrol potentially dangerous areas first. Successful HRC tasks require AUVs to find divers, position themselves, and identify the divers prior to starting collaboration. However, methods allowing AUVs to perform these operations remain under-studied. Overall, Part III presents novel algorithms that enable HRC tasks by providing AUVs with capabilities to approach and identify divers. These algorithms can be used to augment robotic autonomy with human expertise while reducing the physical load for divers, thereby increasing human safety. Chapter 8 introduces the first underwater human-approaching algorithm using monocular vision and human-body priors. This algorithm enables robots to find the diver and achieve diver-relative positioning autonomously while requiring no additional sensors other than a monocular camera, which is commonly installed on most AUVs. Chapter 9 presents two novel algorithms for enabling AUVs to identify divers using face and body pose information using a visual sensor. These methods allow AUVs to collaborate with only authorized divers to secure human-robot collaboration.

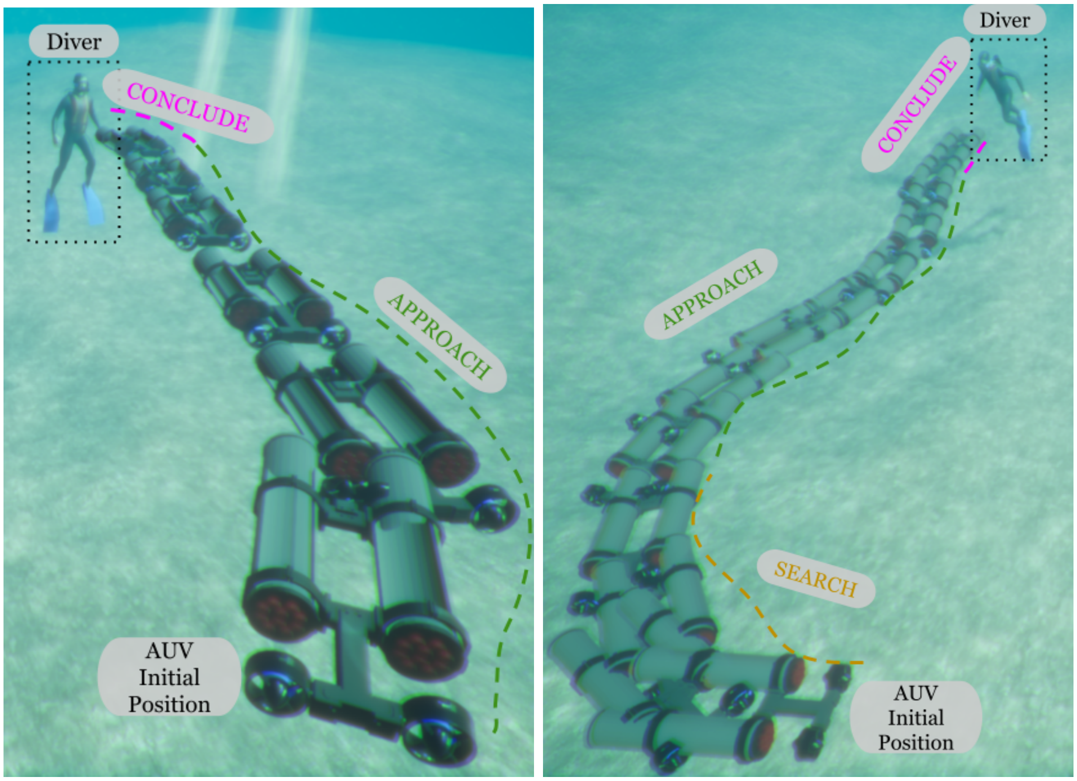
Chapter 8

Autonomous Diver-Relative Operator Configuration for Human-robot Collaboration

As AUVs have been used more commonly for human-robot collaboration tasks (*e.g.*, rescue operations, pipeline and ship inspection), the importance of AUVs' capability to perform diver-relative positioning has increased. However, this has not gained much attention. This chapter presents a novel method for enabling AUVs with such capability. Our method uses only a monocular visual sensor with a body prior and is computationally efficient. In this chapter, we provide details about our algorithm and its evaluations, along with the related background.

8.1 Related Work

Autonomous underwater vehicles (AUV) have traditionally been used for standalone missions, with limited or no direct human involvement, in applications where it is infeasible for humans to closely collaborate with the robots (*e.g.*, long-term oceanographic surveys [169, 170]). However, in recent decades, the advent of smaller AUVs [171, 172, 4] suitable for working closely with humans (termed co-AUVs) has enabled robots and humans to collaborate on tasks underwater. A variety of human-robot interaction (HRI)



(a) Head-on

(b) Turned-away

Figure 8.1: Two examples of ADROC diver approaches: (a) a diver is in the AUV's field of view, so the AUV simply moves to the diver, (b) no diver is visible, so ADROC begins a search procedure, approaching once the diver is found.

capabilities have been developed for co-AUVs to facilitate this work: from gestural programming [173, 174, 175] and various modes of robot communication [176, 177] to diver following [178, 179]. However, underwater HRI systems typically often require that the diver and AUV be close to one another and oriented correctly to operate effectively [180]. When working in challenging, unstructured underwater environments, human dive partners are often required to find one another to begin their interactions, and the same is true of AUVs working with divers. Despite its critical importance, the problem of **how an AUV should find its dive partner and position itself to facilitate interaction** has not yet been addressed. While divers could swim to the AUV themselves, placing the burden of maintaining an appropriate position for interaction on the divers both adds to their existing physical and cognitive loads and robs them of precious time and oxygen. This increases the overhead of working with AUVs, decreasing their usefulness and their adoption rate in the field. An AUV having the ability to seek out and position itself relative to its operator (as seen in Fig. 8.1) will increase the utility of HRI-enabled co-AUVs in underwater work.

The problem of an AUV approaching a diver to facilitate underwater human-robot interactions has not been sufficiently addressed in the past, despite the fact that robots approaching humans in indoor scenarios (offices, warehouses, etc.) have been studied for decades [181, 182, 183, 184, 185]. Service robots, for example, have been approaching humans effectively for years, but the task of achieving this kind of behavior underwater is much more difficult due to the environmental challenges of the domain: *e.g.*, degraded sensing quality, less reliable localization and motion planning, and lack of wireless communication.

When we consider the realm of underwater HRI, the sub-field with the most relation to diver approach scenarios is the topic of diver following, which has been widely studied [178, 179, 186]. Diver following, however, is a very different problem. In diver following scenarios, there typically are no strict constraints on the distance between diver and AUV, no precise evaluation of the position of the AUV relative to the diver, and no functionality for searching for a target who has been lost or is as of yet unseen. All of these are critical aspects of the AUV diver approach scenario. Furthermore, the few works which do impose strict constraints on the position and distance of an AUV relative to a diver [187] require the use of expensive sensors (*e.g.*, stereo cameras, sonar, Doppler velocity log (DVL), ultra-short baseline (USBL) and long baseline (LBL) localization hardware). This prevents AUVs without those sensors from taking advantage of the precise diver-relative navigation abilities of these works, abilities key to effective diver approach scenarios. No existing diver-based AUV navigation method is capable of reliably approaching a diver (robustly achieving a stable, desired relative position, orientation, and distance) using only monocular visual information.

To address this need, we propose **Autonomous Diver-Relative Operator Configuration (ADROC)**: a novel algorithm that gives any AUV with a monocular camera the ability to approach a diver in order to facilitate further human-robot interaction.

8.2 Autonomous Diver-relative Operator Configuration

To enable AUVs to track and approach divers for the purpose of initiating interactions, we present the ADROC algorithm. The proposed algorithm was implemented on the LoCO AUV [4]. However, it could be transferred to any AUV with a monocular camera and five or more degrees of freedom due to its minimal sensor and control requirements.

8.2.1 Desired Behavior

To guide our algorithm's development, we select the following set of desiderata (features desired or required) from our understanding of the AUV interactions we plan to support. We focus on AUVs working alongside humans in a supporting role, with limited sensing and no global localization, as this is the environment where we believe our algorithm will be most useful.

ADROC Desiderata:

D1: Reliably navigate to a beneficial position for interaction with the diver regardless of the initial location and orientation of both parties.

D2: Accurately approach the diver regardless of diver movement after ADROC initiation.

D3: Approach the diver in a timely manner.

D4: Operate without global localization, 3D diver information, or any off-AUV computation whatsoever.

These desiderata are integrally tied to our expected deployments and create a very challenging problem. As such, we make the following assumptions to simplify the problem allowing us to formulate a feasible solution.

ADROC Assumptions

A1: The diver and robot are generally within the visual observation range of one another, and visibility is sufficient to detect the diver in the AUV's cameras.

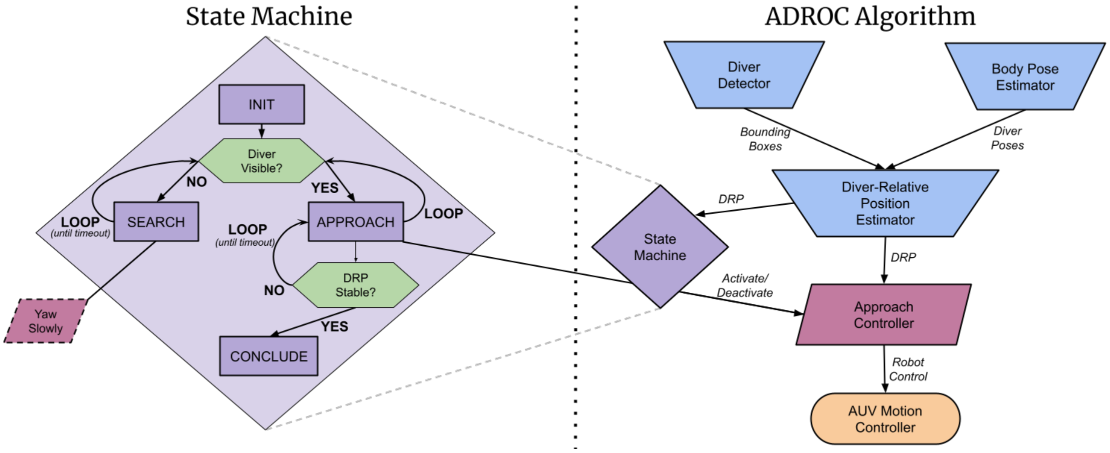


Figure 8.2: Diagram showing the ADROC algorithm *(right)* with detail on the state machine *(left)*. Different components of the algorithm (perception, states, approach controller, and conditions) are presented in consistent colors and shapes.

A2: The diver is generally upright with respect to the robot.

A3: Only one diver will be present in the scene.

Assumptions **A1** and **A2** are likely to be true in actual deployment scenarios. While we cannot guarantee ideal visibility conditions in the field, we must assume some level of visibility (**A1**) for the dive to occur. The diver's orientation relative to the robot (**A2**) affects the accuracy of our body pose estimation algorithm, likely because the algorithm is trained on images of humans in upright positions (standing, sitting, etc.) Our approach algorithm continues to operate with divers in nonstandard orientations, but has the highest accuracy when the divers are in upright positions. This shortcoming could be overcome by developing a body pose estimation algorithm and training it on divers with a wide variety of body positions. Lastly, **A3** is not likely to be true in deployment scenarios, so future versions of this algorithm should account for this by enabling the algorithm to differentiate between divers and approach only a chosen target.

8.2.2 The ADROC Algorithm

ADROC has 3 components: a **state machine**, a **diver-relative position estimator**, and a Proportional-Integral-Derivative (PID) [188] **approach controller** tuned for diver approach operations. These components, along with an AUV motion controller and diver perception modules, are pictured on the right of Fig. 8.2 where the interactions between these components are shown. The diver perception modules are considered external to the algorithm as they may be replaced with other systems if necessary; these modules are briefly discussed in Section 8.3.1. The diver-relative position (DRP) estimator, which is responsible for producing information on the position and distance of a diver relative to the AUV, is addressed in Section 8.3.2, and we discuss the approach controller in Section 8.3.3.

Therefore, we largely focus on the state machine portion of ADROC in this section, which is pictured on the left side of Fig. 8.2. The state machine consists of four states and two conditions which manage transitions between them. To begin, the state machine checks if a diver is currently visible in the scene. If there is an estimate of the diver's relative position available, it transitions from the INIT state to the APPROACH state, which activates the approach controller. During the APPROACH state, the approach controller manages the robot's orientation and distance relative to the target diver, continually checking if a diver remains visible. If at any point the AUV does not see the diver, it transitions into the SEARCH state. The SEARCH state triggers the AUV's search behavior, which is currently a slow yaw motion, turning in circles and scanning for divers. In the future, this could be redefined with a more complex behavior, utilizing information about the environment the robot is in and the last known location of divers in that environment. Once a diver is visible, the SEARCH state transitions to the APPROACH state. Finally, the APPROACH state is successfully terminated once the diver-relative position is stable at a point close to the ideal position within a configurable margin of error. Once the AUV is stable in that position, the state machine transitions to the CONCLUDE state, which currently simply terminates the ADROC algorithm. To understand the APPROACH state and how the conditions responsible for state transition are determined, we must first explain the components of ADROC.

8.3 Methodology

8.3.1 Diver Perception Modules

ADROC is currently designed to operate with diver bounding boxes and diver body pose estimations, which can be obtained from any source. We present the modules currently in use, which operate on monocular images, allowing ADROC to be used by any AUV with at least one camera. The modularity of our approach allows the adaptation of new sensors or perception algorithms in the future.

Diver Detection

While a number of approaches to diver detection have been proposed [178, 179], recent work [189] profiled the efficacy of popular deep detectors for the diver detection task. Based on the analysis of [189], we selected a YOLOv4-Tiny diver detector trained on VDD-C [189] due to its relatively high accuracy with a fast inference time on embedded hardware. The detector outputs a set of detections with associated confidences, from which we select the highest confidence detection (due to assumption **A3**).

Diver Body Pose Estimation

Body pose estimation aims to locate a set of body joints from a given image. We obtain a diver pose with a real-time pose estimation implementation [190] (a TensorRT implementation of human pose estimation [191, 192]). From the pose, we only utilize the left and right shoulder joint coordinates of a single body pose detection for our algorithm.

8.3.2 Diver-Relative Position Estimation

With the perception information being provided by our diver detector and pose estimation modules, we process the resultant data to produce a diver-relative position estimate (DRP), composed of a **target point** (TP) and **pseudo distance** (PD). This data is used by our approach controller to center the diver in the frame of the camera and to reach a desired distance from the diver. In calculating both the target point and pseudo distance, we utilize all available information from both the bounding boxes and shoulder coordinates returned by our diver perception modules. Our algorithm is robust to missing information from either module.

Target Point

We define the target point based on bounding box information to be the centroid of the bounding box. Alternatively, based on body pose estimates, we define the target point as the center point between shoulder joints. If both bounding boxes and body pose estimates are available, the target point is defined as the mean of these two points, but if only one is available, it is used as-is.

Pseudo Distance

Common approaches to estimate the range to a diver would be to use sonar data or calculate disparity and estimate depth from stereo imagery, which has its own challenges [193, 194]. Our algorithm is designed to work with only monocular vision available so that it functions even on the least sensor-equipped AUVs. This creates a challenge, as monocular images do not contain sufficient information to accurately estimate distances, which we need to navigate to an appropriate distance relative to the diver. We overcome the problem of estimating the range to the diver without introducing new sensors by making a rough estimate which utilizes the shoulder width of the diver, called *Biacromial breadth* [195]. Biacromial breadth can differ based on demographic and individual variation, but we use average data available from national surveys [195], with the option of fine-tuning our estimate with specific measurements of diver shoulder width for an individual. For body pose estimation input, the shoulder width is the distance between the estimated shoulder points, but for diver detector input we treat the bounding box width as a rough estimate of shoulder size.

Table 8.1: The Pseudo Distance (PD) metric based on distance d between a diver and robot.

Distance (d(mm))	**PD**
$d > D_{ideal}$	$0 < PD < 1$
$d = D_{ideal}$	$PD = 1$
$d < D_{ideal}$	$PD > 1$

We propose a metric embedding distance information, *Pseudo Distance (PD)*, using diver detection and pose estimation as individual sources of shoulder width information (Eq 8.1). Psuedo distance is defined as inversely proportional to the ideal interaction distance. It is greater than 0 at any distance, less than 1 when farther from the diver than desired, 1 at the ideal distance, and greater than 1 when the robot is closer than the ideal distance (Table 8.1).

The following camera sensor, image, and physical specifications are used to develop PD: CMOS sensor size (mm), focal length (mm), image size ($pixel$), and shoulder width of a diver (mm, $pixel$). The detailed steps are:

1. Empirically select an ideal distance $D_{ideal}(mm)$ for interaction between a diver and AUV.

2. Take the average shoulder width reported in [195] as the width of the diver's shoulder. Alternatively, measure the true shoulder width of the diver.

3. Measure the shoulder width in image pixels ($w_{shoulder}$) from an image with the diver at the ideal distance $D_{ideal}(mm)$.

4. Select $target\_shoulder\_ratio$ as the ratio of the shoulder width ($w_{shoulder}$) to the image width (w_{img}).

The $target\_shoulder\_ratio$ is separately defined for diver detection and pose estimation information, as the source of the shoulder width estimate varies drastically in accuracy, with body pose estimation being more accurate. This is due to the fact that a bounding box changes size significantly based on the diver's body pose (*e.g.*, a diver with open arms yields a much wider bounding box than one with arms held at the side), while the distance between shoulder joints does not change based on the rest of the diver's body pose.

$$PD = \frac{w_{shoulder}}{w_{img} \times target\_shoulder\_ratio} \tag{8.1}$$

Whenever body pose estimation information is available, we default to using a pseudo distance based on pose data, as it is significantly more accurate. However, when such information is not available, we fall back to our estimate based on diver detection bounding boxes.

8.3.3 Approach Controller

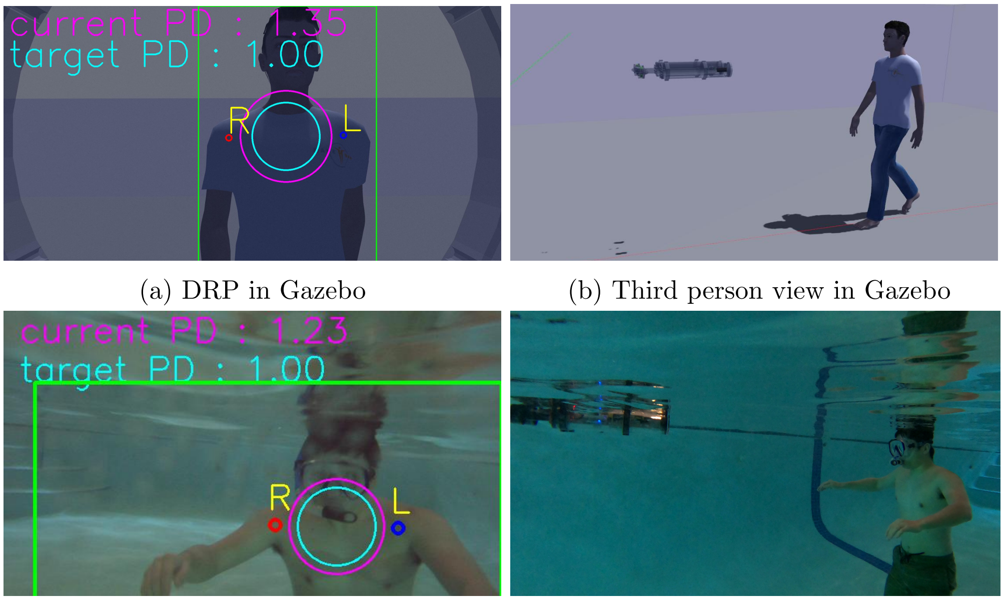
The calculated DRP is passed on to our approach controller, which manages the approach procedure based on the target point and pseudo distance estimations produced by the DRP estimator. The approach controller maintains three separate PID controllers for the three axes of control it has: one controller for surge (forward and back), one for yaw (left and right), and one for pitch (up and down). These controllers are based off of error measurements (Eq. 8.2-8.4) comparing the target point to the image's center point and comparing pseudo distance to the ideal pseudo distance, which is defined as 1.0 (see Table 8.1).

$$error\_forward = 1.0 - PD \tag{8.2}$$

$$error\_yaw = \frac{target\_x - center\_x}{image\_width} \tag{8.3}$$

$$error\_pitch = \frac{target\_y - center\_y}{image\_height} \tag{8.4}$$

The PID controllers in the approach controller operate on these errors in the standard fashion [188], which we will not detail here. Their parameters are tuned separately based on experimental results. The approach controller constantly calculates motor control values based on the available target point and pseudo distance, but only applies those controls to the motors (resulting in motion) when enabled by the ADCROC state machine switching to the APPROACH state. The approach controller limits its speed to 60% of the AUV's maximum velocity for safety.



(a) DRP in Gazebo

(b) Third person view in Gazebo

(c) DRP in pool.

(d) Third person view in pool.

Figure 8.3: Diver-relative position (DRP) visualizations with a third person view, displayed in the Gazebo simulator and a pool scene. In the DRP visualization, the center of the circle is the target point while the radius represents pseudo distance.

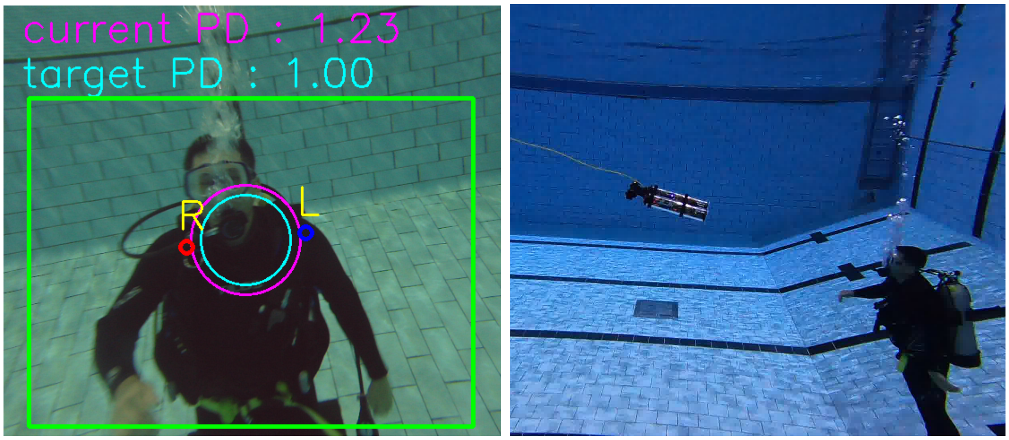
8.4 Experimentation

8.4.1 Experimental Platforms

To test this work, we used the the LoCO AUV [4], which is a modular, low-cost, open-source AUV equipped with dual monocular cameras and three thrusters. All the computation required for ADROC was done onboard, primarily using a Nvidia Jetson TX2 mobile GPU. Additionally, we used a LoCO simulation in ROS Gazebo [196] (Fig. 8.3) as a tool for developing the algorithms and evaluating them.

8.4.2 Pool Experiments

A set of pool experiments was performed to validate the ADROC algorithm's ability to successfully approach a diver. These experiments took place in two different pools, one with clear water and bright, even lighting, and another with murky water and dim, irregular lighting. One example is shown in Fig. 8.4. A total of 9 divers were used as the target of the approach algorithm, with shoulder widths between 33cm and 48cm.



(a) DRP in pool.

(b) Third person view in pool.

Figure 8.4: Diver-relative position (DRP) visualizations for a scuba diver in a deep pool.

Six of the divers were tested in the clear pool (Experiment #1) and three in the cloudy pool (Experiment #2). In each trial, the AUV was made to approach the diver from the combination of three initial distances (3 meters, 6 meters, 9 meters) and from three initial angles between the AUV and the diver (0°, 45°, 90°) as shown in Fig. 8.5. These variables of distance and orientation combine to create nine distinct conditions of AUV approach. For each condition, two trials were conducted per diver for a total of 162 trials. For each trial, we allowed the AUV to enter the SEARCH state up to twice after the first APPROACH state. If the AUV reached the CONCLUDE state at the desired position before entering the third SEARCH state, we considered the case as a success, but as soon as the AUV entered the third SEARCH state after an APPROACH, the case was deemed a failure. Additionally, we recorded the time taken for each case, regardless of success.

8.4.3 Results

Types of Failures

Of the 162 recorded trials, 30 failures were recorded. These failures can be grouped into three categories: **search failures**, **early conclusion**, and **no conclusion**. **Search failures** (12 cases) were characterized by repeated returns to the SEARCH state, usually due to one of two issues: overshooting the initial rotation required for an approach, or

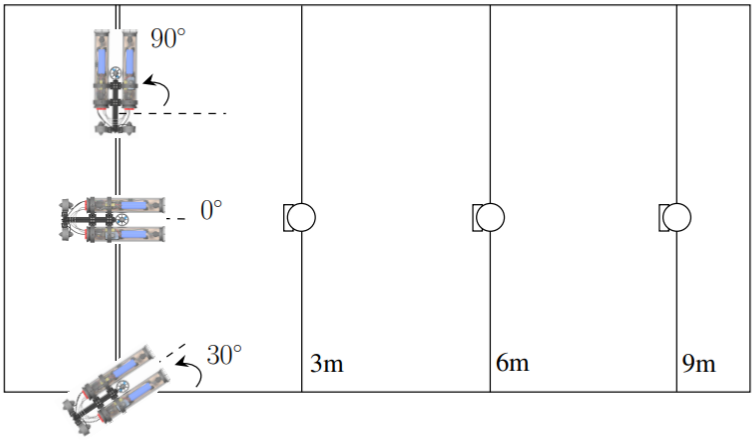


Figure 8.5: Pool experiments setup: three distances ($3m$, $6m$, $9m$), three angles (0°, 45°, 90°). Circles represent diver positions for the experiments.

receiving inaccurate detection data from the diver perception modules. Re-tuning the approach controller may help to reduce these failures. **Early conclusion failures** (11 cases) were caused by ADROC entering the CONCLUDE state prior to reaching the appropriate relative distance to the diver. These occurred almost exclusively for participants with small shoulder widths and were caused by the pseudo-distance based on bounding boxes reaching a stable point at a distance of 5 meters, prior to the AUV entering the detection range of the diver pose estimation. This type of failure can likely be entirely resolved by tuning the *target_shoulder_ratio* as described in Section 8.3.2. Lastly, **no conclusion failures** (7 cases) were caused by ADROC failing to detect a stable position while in the appropriate relative position to the diver. This failure is likely due to tuning issues with the approach controller, but should also be resolved by further improvement of the approach controller.

Aggregate Data

The summarized data in Table 8.2 represents the averages of our results. We note that Experiment #1 has a higher overall success rate (88.9%) compared to Experiment #2, likely due to the better visual environment which led to higher quality perceptual inputs. Success rates reduce as the initial distance increases, most likely due to reduced accuracy of diver detection and pose estimation, which leads to fewer successful searches. Average times for an approach increase as well, but this is mostly due to the increased

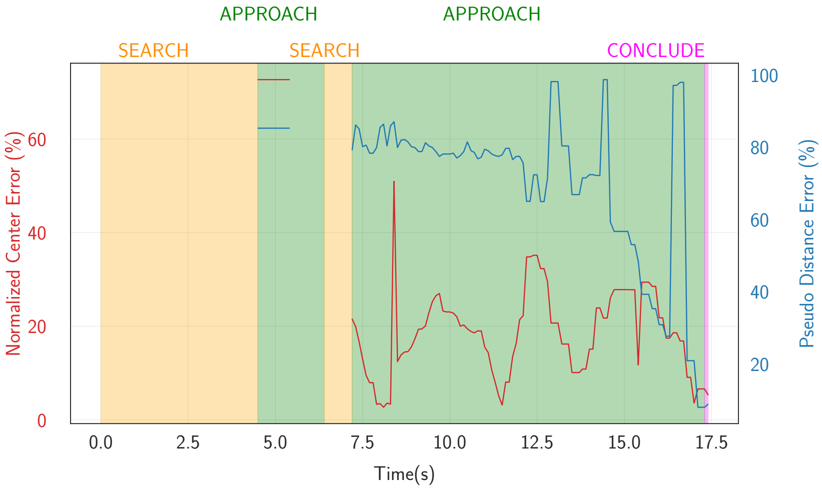
Table 8.2: The success rates and average operation time of ADROC based on the trial, distances, and angles.

Experiment	**Success**	**Avg. Time**
Exp. #1 (clear pool)	88.9%	25s
Exp. #2 (cloudy pool)	66.7%	28s
Distance	**Success**	**Avg. Time**
3 meters	92.6%	21s
6 meters	83.3%	28s
9 meters	68.5%	29s
Angle	**Success**	**Avg. Time**
0	77.8%	22s
45	83.3%	24s
90	83.3%	33s
Overall	**81.5%**	**26s**

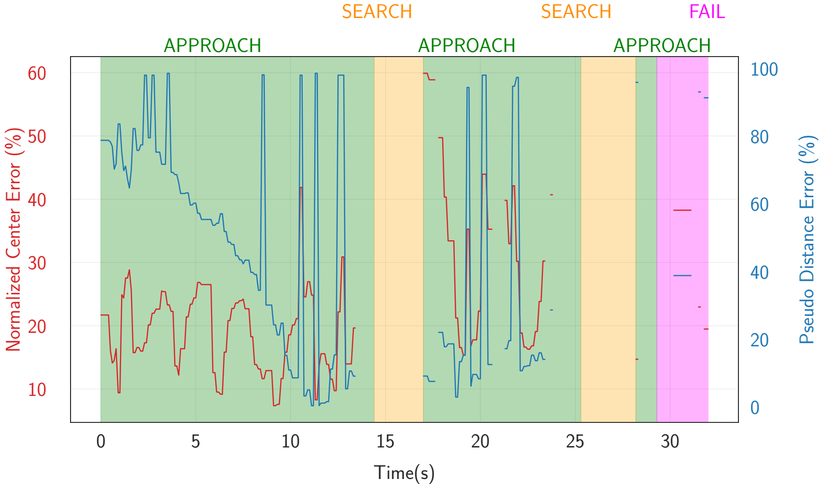
distance that the robot has to travel. When considering the effect of the initial angle, we see that approaches from the 0 condition have the lowest accuracy, but not by a statistically significant amount. This is likely due to random chance, as the difference in success between the 0 and the other angle conditions (which have the same success rate of 83.3%) is only 3 failures.

ADROC Runtime

All components of ADROC were run on the onboard computers of the LoCO-AUV: the Raspberry Pi 4 and the Nvidia Jetson TX2. The diver detector and diver pose estimator, running concurrently on the TX2, ran at 15 fps and 10 fps respectively. Also on the TX2, the diver-relative position estimator ran at a set frequency of 20 Hz, while the ADROC state machine ran at 10 Hz. The only component which ran on the Raspberry Pi was the approach controller, which runs at a set frequency of 10 Hz. This frequency of the overall system (about 10 Hz) was sufficient to operate the AUV at relatively low speeds, although an improved frequency is possible with a more powerful GPU.



(a) Success example with narrow shoulder width diver ($35cm$), long distance ($9m$), and turned-away angle(90°)



(b) Failure example with wide shoulder width diver ($45cm$), long distance ($9m$), and head-on angle(0°)

Figure 8.6: (a) LoCO was turned away from a diver at the beginning. Through the SEARCH and APPROACH states, LoCO detected the diver and placed itself at the designed distance from the diver, (b) LoCO was facing a diver and started with the APPROACH state. However, spikes in PD error (bounding box only DRP estimation) caused unstable control and failure.

Individual Cases

Fig. 8.6 presents one of each success and failure cases from our experiments. In Fig. 8.6a, the AUV was at the turned-away angle, thus starting with the SEARCH state. During its second APPROACH state (t=7.3-17.3s), the AUV occasionally failed to detect pose estimates of the participant, and the diver detection estimate was used to yield the PD. The spikes (t=12.9s, 14.4s, and 16.4s)in the Pseudo Distance Error were caused when the AUV controlled its distance to the participant based on the diver detection as opposed to body pose estimation. In Fig. 8.6b, the AUV started with the APPROACH state since it was at the head-on angle and was able to see the participant right away. The participant was detected by the diver detection consistently while the pose estimation only failed occasionally. More frequent PD estimation based on the diver detection caused an overshoot from the approach controller (more spikes from the beginning), and it resulted in failure. Additionally, the missing data points during the second APPROACH phase (t=20.7-21.2s) are consistent with system lag, possibly due to processing issues with the camera or some competition between processes for computational resources.

8.4.4 ADROC Limitation Experiments

To explore the limits of ADORC, a small number of trials were performed with more challenging conditions, in a deep pool similar to the pool used for Trial #1. These trials are not included in the aggregate results above, because of the small number of participants for each. We repeated ADROC trials for one participant who was included in the normal trials in scuba equipment such as a wetsuit instead of the general swimwear that our participants wore. No reduction in accuracy compared to non-scuba trials was detected. Trials were also conducted with an adversarial condition, in which the AUV was pushed partway through an approach, mimicking a wave knocking the robot away from its chosen path. This also did not cause a reduction in accuracy. Lastly, trials were conducted up to 15 meters away from the diver. While the algorithm generally still functions, the AUV struggles to effectively switch from SEARCH to APPROACH as the approach controller sends yaw commands of a high intensity causing the AUV to quickly overshoot and miss its target. This could likely be improved with pseudo distance-relative tuning of the approach controller.

8.5 Conclusion

We present a novel algorithm, Autonomous Diver-Relative Operator Configuration (ADROC), to facilitate diver-AUV collaboration. We demonstrated that ADROC enables an AUV to robustly approach divers with various shoulder widths from different distances and angles in our experiments. The advantages of our method are that it only requires a monocular camera, needs no extra sensors to estimate the distance to a diver, functions without global localization, and runs at real-time speed using on-board AUV. Our presented work clearly demonstrates the efficacy, reliability, and potential of our algorithm, despite the minimal information and sensors utilized. In the next chapter, we advance one more step and introduce two diver identification methods that enable secure human-robot collaboration tasks.

Chapter 9

Diver Identification for Human-robot Collaborations

To operate secure missions as underwater human-robot teams, AUVs need to identify divers accurately. However, this remains an open problem due to divers' challenging visual features, mainly caused by similar-looking scuba gear. This chapter presents our approaches to addressing a diver identification problem using data obtained by a visual sensor, which is commonly available on AUVs. We introduce two methods: (1) DiverFace, to identify divers using their facial features, and (2) DiverID, to recognize divers using their body pose information. Each method is presented in detail in the following sections.

9.1 Related Work

One of the preliminary works on person identification was described in [197], where the proposed method used anthropometric data to identify different individuals. As the technology evolved, anthropometric data-based identification methods were replaced by identification systems using biometric information, such as fingerprints, voice, and iris scans [198]. Spectral information extracted from electroencephalogram (EEG) signals can also be used to identify a person [199]. Furthermore, person identification using facial cues has seen incredible success [200, 201], especially with the availability of large-scale human face datasets [202, 203, 204], superior computing power, and advances in deep convolutional neural networks [205]. Alternatively, gait detection can also be used to uniquely identify a person, as it has been found that walking patterns have a high correlation to a subject's identity [206].

The above-mentioned techniques were developed for terrestrial applications. Person identification techniques in an underwater environment (*i.e.*, diver identification), on the other hand, are significantly less studied. Existing vision-based diver identification methods [207, 208] use motion characteristics to recognize divers, which would work neither in close proximity nor in the case of immobile divers. A possible solution would be to use facial cues for identification. In the literature, human subject identification using faces has been widely studied for terrestrial robots [209, 210]. In contrast, diver identification using faces has received little attention, partially due to the fact that divers' faces are heavily obscured by their masks and regulators.

In recent years, deep convolutional neural networks have yielded high accuracy in face recognition tasks [211, 212, 213]. In general, most of these methods include the following steps:

1. **Face detection**: by learning a feature pyramid [214], using a "proposal and refinement" mechanism [215], or densely sampling locations across multiple scales [216];

2. **Face classification**: achieved by either training a multi-class classifier to classify different identities in the training set using softmax loss [217] or directly learning the face representation (*i.e.*, embeddings) to classify different identities which may not be present in the training set [218].

For the face recognition task, the primary objective is to ensure compactness in *intra-class distances* (*i.e.*, the same faces) while maximizing the *inter-class distances* (*i.e.*, different faces) [219, 220].

Generally, traditional methods discussed above exhibit poor performance on partially-occluded faces. The work in [221] proposed to extract local face descriptors only from the non-occluded facial areas. In [222], a deep network is used as a feature extractor to classify faces from partial data. However, the discriminative power of these methods is limited. In [223], the authors code the occluded face image as a sparse linear combination of the training samples and the occlusion dictionary. But, it fails to generalize because of the assumption that test samples have identical subjects as the training

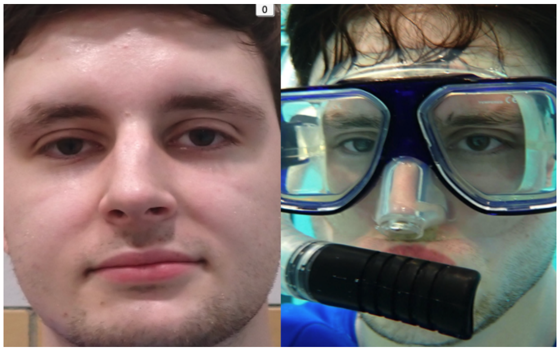


Figure 9.1: An example of facial occlusions by breathing apparatus underwater. The left side is the face from the in-air, and the right is the face wearing the apparatus.

samples. In contrast, the work in [224] trains a deep network to detect facial keypoints from partial face images and then the angles between the keypoints are used to perform face identification. However, none of these methods show how to tackle significant facial occlusions as those occurring from underwater breathing apparatus (Fig. 9.1). In addition, the work in [225] shows that the network trained on a large dataset is unable to achieve 50% accuracy when faces are cropped by more than 40%. This research gap is mostly due to the difficulty in collecting real-world diver face datasets, distortions present in underwater imagery, and the complex facial occlusions occurring on diver faces.

In addition to using facial cues for identification, Munsell *et al.* [226] show that anthropometric data (AD) can be used to identify different individuals. They demonstrate that these data are less sensitive to photometric differences and more robust to obstructions, *e.g.*, glasses, and hats. Similarly, the work in [227] shows that anthropometric and gait features can be used to uniquely identify different people. The authors extract such features from data of different individuals walking. In this chapter, we also choose to use anthropometric data to represent and identify different scuba divers. To compute these AD values, we turn to 2D pose estimation methods which can find the location of different human body joints. There are a number of 2D pose estimation approaches (*e.g.*, OpenPose [228], trt_pose [190], MediaPipe [229], DeepLabCut [230] to name a few) which achieve high accuracy in terrestrial applications. For our purpose, these methods generated a high number of incorrect pose estimations for the divers we used in our study, possibly due to water turbidity and challenging lighting conditions. A recent work [231] uses high-resolution representation learning to achieve semantically rich and spatially precise pose estimations. The authors employ several high-to-low-resolution convolutional branches in parallel to achieve this. We also exploit this method in our work to estimate diver poses.

9.2 DiverFace: Diver Identification Using Facial Cues

The overall pipeline of the proposed approach consists of both online and offline components. Once trained, the inference process runs entirely on-board AUV.

9.2.1 Offline Module

Data Augmentation

A significant challenge in the diver face identification task is the unavailability of diver face datasets. To circumvent this issue, we resort to using publicly available traditional regular face datasets and perform a series of operations to transform them into diver faces. Our augmented face dataset thus includes both regular and diver face pairs. First, we perform *frontalization* to the faces, as described in [232]. That is, given a query photo, we detect facial features on both the query photo and a rendered 3D face model. The 2D coordinates on the query image and the corresponding 3D coordinates on the 3D model enable us to estimate a projection matrix to project the 2D keypoints from the query photo onto a reference 3D coordinate system. Finally, the missing pixels' color intensities are filled in with the original query image's symmetric regions. We perform an automatic crop on the processed image to retrieve the original composition of the image. The motivation behind the *frontalization* step stems from the fact that the scuba divers will likely be in a frontal-looking position while directly interacting with an AUV.

Next, we use a facial keypoint prediction network, which uses four convolutional layers and three fully connected layers to output 15 (x, y) keypoint coordinates. These 15 keypoints are used to put different kinds of masks and regulators on top of the faces to make them look like diver faces. We process the masked images further to achieve the underwater green/blue hue [5]. In addition, to account for the distortions, such as

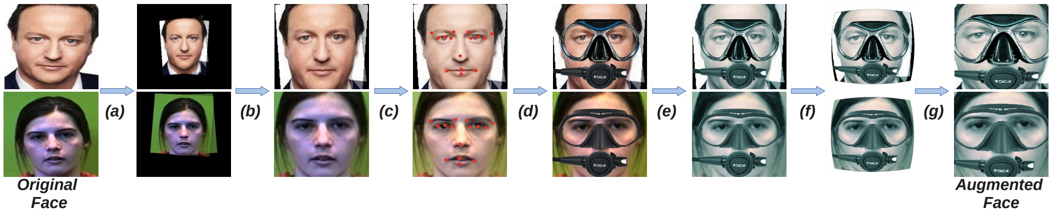


Figure 9.2: Data augmentation steps for preparing the training dataset for the facial feature extractor network. (a) *Frontalization*. (b) Automatic crop. (c) Color correction and keypoints regression. (d) Apply mask and regulator based on the 15 keypoints. (e) Colorization [5] to incorporate underwater lighting conditions. (f) Applying underwater effect [6]. (g) Crop to tighten the boundary.

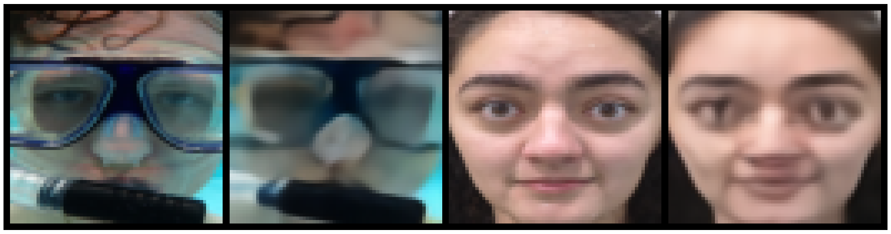


Figure 9.3: Reconstructed faces of two participants. The left side of each pair is the original image, and the right is the reconstructed one: (left pair) diver face and (right pair) regular face.

barrel distortions [233] and longer focal lengths caused by underwater optics, we add some underwater effects [6] to the face images so that they closely resemble the pictures taken underwater. Fig. 9.2 demonstrates all the steps of the data augmentation process.

Feature Extraction Method

To extract discriminative features from facial images, we use a convolutional autoencoder [234]. The autoencoder first encodes the face as a compact hyperdimensional feature vector from which it decodes the feature vector to reconstruct the face. The convolutional layers of the autoencoder help to capture features from the images. Essentially, the network learns to obtain a highly discriminative 512-dimensional feature embedding to represent a face. We vary the composition of the training dataset (*e.g.*, regular + diver faces and diver faces only) to see the impact on the reconstructed faces. Moreover, we vary the size of the encoder output to create different dimensional feature embeddings for comparison purpose. Fig. 9.3 shows two sample outputs from the autoencoder where it tries to reconstruct the faces of two participants.

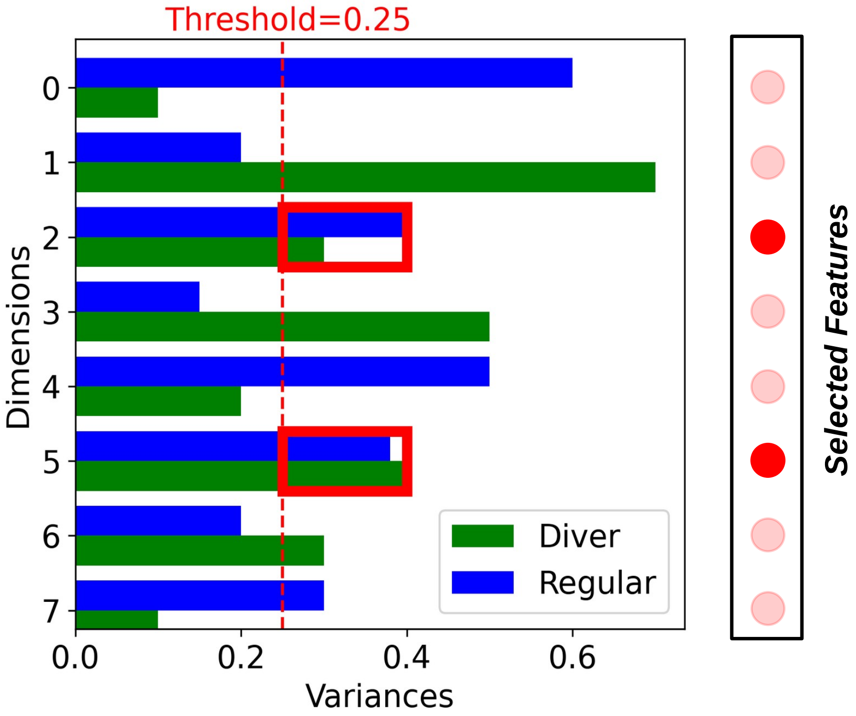


Figure 9.4: An example of variance analysis for dimensionality reduction. The blue and the green values represent variance values for regular and diver faces, respectively. The variance values for both regular and diver faces from dimensions 3 and 6 are more than a predefined threshold value (0.25), making these two dimensions desirable for representing regular and diver faces.

Variance Analysis for Dimensionality Reduction

To correctly match a person's diver face with their regular face, the diver's face without the mask and the breathing apparatus would be ideal to have. We achieve this by performing a variance analysis (VA) on both regular and diver faces to find the feature dimensions that contribute to highly discriminative facial cues only (*i.e.*, highly variant dimensions). First, we use the autoencoder to extract 512-dimensional feature embeddings from the faces (regular + diver) in our dataset. With this embedding, we compute the variance of all 512 feature values across all the regular faces which gives us a 512-dimensional variance vector. We perform a similar computation on diver faces to yield a similar length variance vector. Second, we compare these two 512-dimensional variance vectors to find the highly variant dimensions. Specifically, we use a predefined threshold to find these dimensions. To find the threshold, we start at the initial value of 0.009 and increment by 0.005 to generate a set of thresholds. Then, we use values from this set to lower the feature dimensions of the faces from the training dataset and calculate the identification accuracy with the reduced features. We pick the threshold value which gives the highest identification accuracy. This way, we find a lower-dimensional (R-d, $R << 512$) feature representation for both diver and regular faces which contain highly variant feature values. VA converts the original feature embedding (*i.e.*, encoded output from the autoencoder) to a lower-dimensional feature representation by filtering out inconsistent features between each diver and regular face pair. This will also require less memory to store the feature representations. Fig. 9.4 shows a notional example of the VA method for 8-dimensional feature representations. From this, we see that only the third and the sixth dimensions appear to be highly variant based on the set threshold value. Therefore, the final feature representations of diver and regular faces should have only these two dimensions.

Diver Face Identification Database

To correctly identify a scuba diver for a secure HRC task, an AUV must know beforehand who the particular individuals are to interact with. To serve this purpose, we construct a diver face identification database that stores feature embeddings for regular faces of the *authorized* divers. Specifically, we store the lower-dimensional embeddings, computed as described in Section 9.2.1, of the regular faces. We store at least two facial feature embeddings for each subject to reduce outlier data, resulting in minimizing the false negative rate. Note that we do not store any information directly obtained from the diver faces in the identification database.

9.2.2 Online Module

Diver Face Localization

When multiple divers are present near an AUV, the AUV needs to authenticate a diver before they are authorized to issue commands. First, the AUV needs to localize or detect diver faces. To this end, we found that RetinaFace [235] performed the best in localizing diver faces in underwater images compared to other face detection algorithms

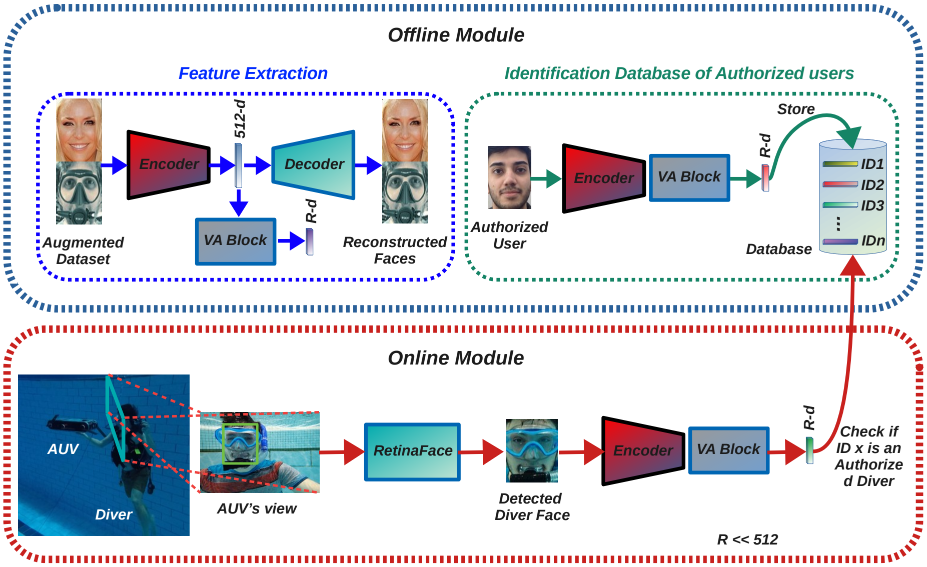


Figure 9.5: Complete pipeline of the proposed diver face identification system. The overall system consists of an online as well as an offline component. In real-world scenarios, only the online module is executed by AUVs to recognize scuba divers, *e.g.*, to ensure they are allowed to operate the AUVs.

(*e.g.*, [236, 237]). RetinaFace is a single-stage pixel-wise face detector that employs a joint extra-supervised and self-supervised multi-task learning strategy to localize faces at various scales.

Diver Face Feature Extraction and Matching

Once the AUV has detected a diver face from the captured image, it will extract facial feature embedding from the diver face using the method described in Section 9.2.1. Since we infer salient dimensions from extracted facial features (Section 9.2.1), we extract the diver face feature embedding from the feature values in those dimensions only. Then, this embedding is compared against all the stored embeddings in the diver face identification database. If a match is found, the AUV would interact with the diver.

Fig. 9.5 shows the complete pipeline of the proposed diver face identification framework.

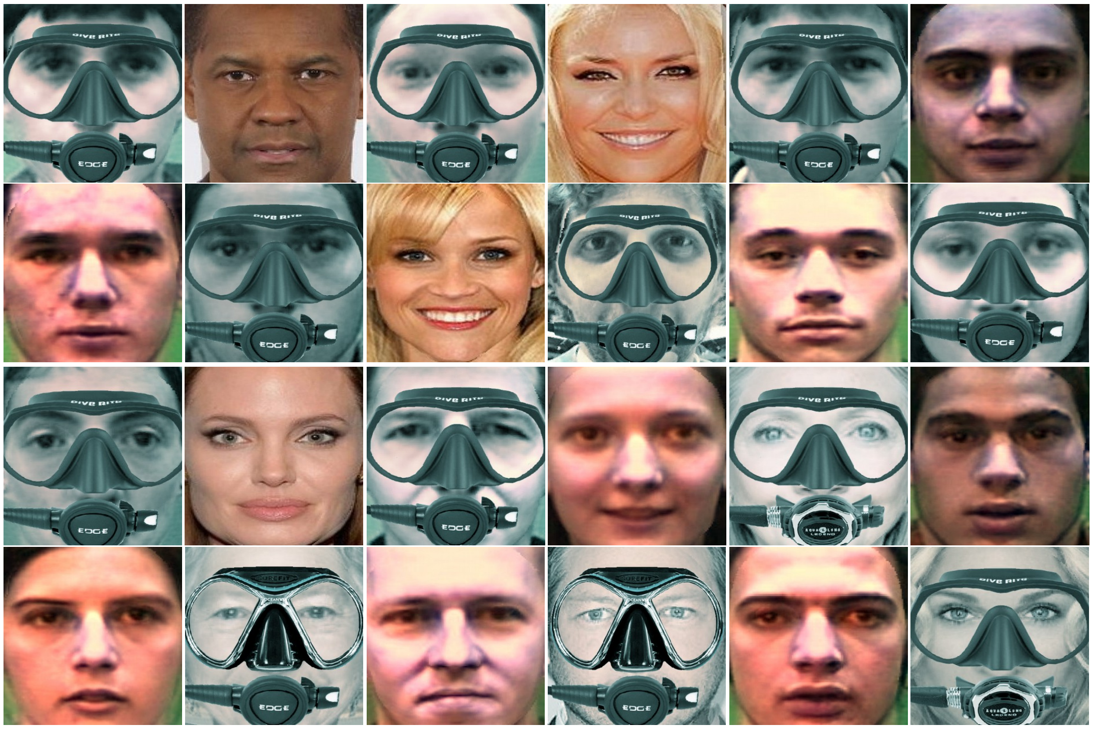


Figure 9.6: A few instances sampled from the proposed dataset. It includes a total number of 22, 031 (regular + diver) face images with 184 different subjects.

9.3 Experimental Setup and Evaluation

9.3.1 Dataset

Training Set

There are 184 different subjects in the dataset, collected from the following sources [238, 239]. Each subject has multiple regular face images. We augment the dataset by putting four different types of masks and two different types of regulators on top of the regular faces. This gives us a total number of 22, 031 (diver + regular) samples of 184 different subjects. We use different combinations of masks and regulators so that our feature extraction model learns generalized discriminative face features. Finally, we carry out the steps described in Section 9.2.1 to prepare the final dataset. We show a few sample images from the prepared dataset in Fig. 9.6.

Evaluation Set

We conducted a human study[1] with 27 participants to create an evaluation dataset for the proposed method. We took their facial photos while they were in the following conditions: 1) fully submerged in the pool wearing a mask and a snorkel and 2) outside of the pool without dive apparatus. Diver face images are collected from three different pool environments, under different lighting conditions, and at various water depths. Even though our augmented dataset has regulators on faces while the participants were wearing snorkels, our proposed method appears to be invariant with the type of breathing apparatus from our experiments.

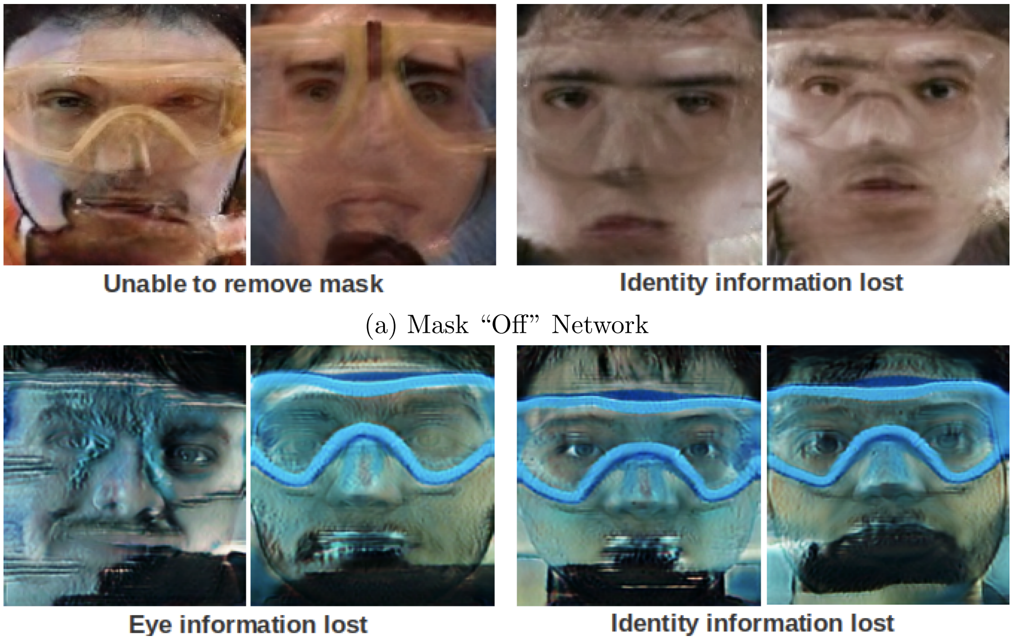
9.3.2 Evaluation Criteria

We evaluate the performance of the proposed approach on real-world data by comparing a *query* diver face against the stored regular faces in a feature space. Specifically, a *query* embedding is compared against the stored embeddings by calculating the cosine similarity (CS) [240] between them. The CS metric provides better distinction between our test images than other similarity measures (Euclidean and Manhattan distance). The CS between two feature vectors f_q (embedding of the query face) and f_a (embedding of the stored face) is defined as,

$$CS(f_q, f_a) = \frac{f_q^T f_a}{||f_q|| \, ||f_a||} \tag{9.1}$$

Our algorithm uses the CS values to classify the ID of the query embedding by selecting the ID from stored embeddings which gives the highest CS. We check if they are the *same subject*. If they are the same, the prediction is considered correct. To calculate the accuracy of the predictions, we take a number of test samples (*i.e.*, real-world diver faces) to match them against the stored subjects and keep a count of the correct predictions. Finally, we divide this number by the total number of test samples to get the final prediction accuracy.

[1] The study (reference no. 00013497) was reviewed and approved by the University of Minnesota's Institutional Review Board.



(a) Mask “Off” Network

(b) Mask “On” Network

Figure 9.7: Failure cases of the domain transfer techniques. (a) Fails to completely remove masks from the diver faces. (b) Improperly trained data generation model completely destroys the underlying information of the image.

9.3.3 Model Selection and Evaluation Results

To evaluate the performance of our approach against other methods, we conduct two baseline experiments in addition to our approach on the evaluation datasets. In the first experiment, we test SOTA face recognition algorithms to extract features. In the second experiment, we perform domain transfer (diver-to-regular and vice versa) on the evaluation dataset first and extract the features with the face recognition algorithm. Lastly, we use our method to extract features from the dataset. We provide details in this section how we perform feature extraction for each case.

Face Recognition Algorithms

We select ArcFace [201] and FaceNet [213] as baseline face recognition models because they are two of the most accurate face recognition algorithms. We train both models with our augmented dataset (see Section 9.3.1). We choose the following parameters for the networks. For ArcFace, we use a batch size of 128 and an input size of 112 × 112.

As the optimizer, we choose stochastic gradient descent with a learning rate of 0.01 and a momentum of 0.9. For FaceNet, we select a batch size of 128 and resize the input size from 112×112 to 160×160, which is required by FaceNet. We train the network using Adagrad optimizer with a learning rate of 0.1 and a learning rate decay factor of 1.0. From our tests, ArcFace and FaceNet show 14.8% and 18.5% accuracy, respectively. During our tests, we see that facial embeddings of size 512 converges to the lowest validation loss and therefore, we use 512-dimensional feature embeddings to represent faces. The number of training parameters (ArcFace: 40M and FaceNet: 140M) of both networks enable them to learn facial features from large datasets (*e.g.*, VGGFace2 [241] (3.31M) and MS-Celeb-1M [203] (1M)), but they overfit with our training dataset since it is much smaller compared to the large datasets.

Domain transfer and Face Recognition Algorithm

To minimize the error coming from the discrepancy between stored regular faces (ground domain) and query diver faces (underwater domain), we apply domain transfer from both sides: 1) Mask Off: domain transfer from diver faces to regular faces and 2) Mask On: domain transfer from regular faces to diver faces. We choose CUT [242], which is a generative model to perform domain transfer. With this approach, we can extract features from faces within the same domain (*i.e.*, underwater vs underwater, ground vs ground). We use the following parameters to train CUT: batch size of 1, Adam optimizer with learning rate of 0.0002, $\beta_1 = 0.5$, and $\beta_2 = 0.999$. While it shows improved performance compared to the former experiment in the case of "Mask On" (25.9%), it is prone to failure when the generation is not reliable. Fig. 9.7 shows a few instances when generation from CUT failed and worsens the identification process. "Mask Off" shows much worse performance (7.4%) compared to other experiments since the generation process of regular faces were susceptible to many factors (*i.e.*, type of faces, mask color, snorkel shape, etc). Furthermore, the generated faces often have smudged facial features (*e.g.*, nose and mouth), and this degrades the performance of this approach.

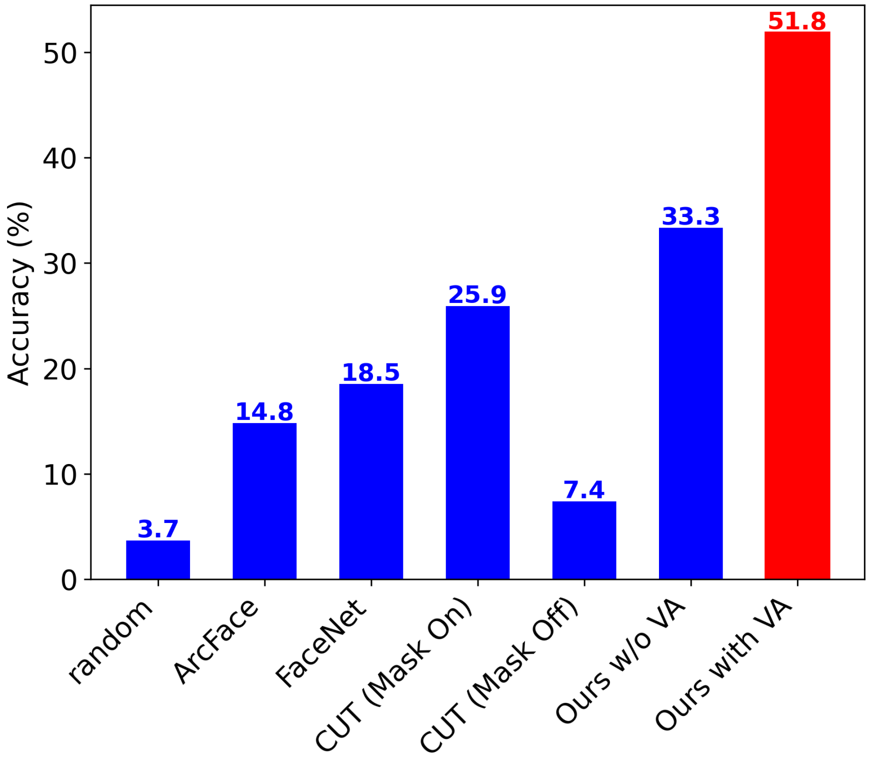


Figure 9.8: Diver face identification accuracy results evaluated on real-world data with different approaches. Our approach with VA shows the highest accuracy among all the approaches.

Proposed approach

For our feature extractor, we use a convolutional autoencoder [234] with both regular and diver faces from our augmented dataset. The model parameters are 19M and 20M for the encoder and the decoder, respectively. We use a batch size of 128, and we train the network with Adam optimizer, having a learning rate of 0.001. We perform VA to find a highly variant 194-dimensional feature embedding (instead of 512) to represent faces. As can be seen from Fig. 9.8, our approach achieves an accuracy of 51.8% in correctly identifying divers, which is higher than all other comparing methods. Our experiments with smaller initial embeddings, *e.g.*, 128- and 256-dimensional, show their best accuracies of 25.93% and 37.04% even after performing VA.

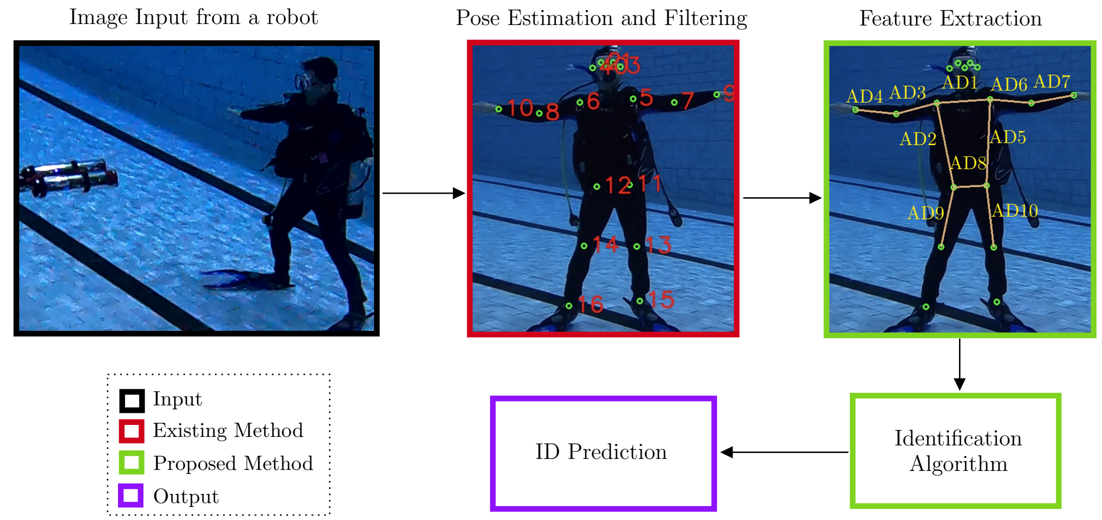


Figure 9.9: Diver Identification Framework: Given an RGB image input, our algorithm first extracts features from the diver's pose. We then predict the diver's ID using these highly distinguishable features. The numbers in the middle figure represent 17 body keypoints. Additionally, the AD$\{1, \cdots, 10\}$ in the right-side figure shows features extracted from the pose estimation as discussed in Section 9.4.

9.3.4 Practical Feasibility

The accuracy of the proposed method is around three times higher than the SOTA face recognition algorithms for a diver identification task (see Fig. 9.8). It shows that our approach is effective in capturing discriminative features from diver faces even though they are heavily obscured with mask and breathing apparatus. Furthermore, it only requires 194-dimensional feature vectors for each individual information, which uses minimal disk space to store them. Our model only requires 19M parameters, which is much less than the SOTA facial algorithms, such as ArcFace(40M) and FaceNet(140M). This helps reduce the computational load on embedded robotic computing hardware and will lead to faster inference time. From our testing on an Nvidia Jetson TX2 embedded GPU, it took approximately 433 ms to load the model and approximately 133 ms to make the inference. The results lead us to believe that the proposed framework can run in real-time. However, the model loading time needs to be improved by reducing its size in case multiple deep learning models are being simultaneously used on board AUV.

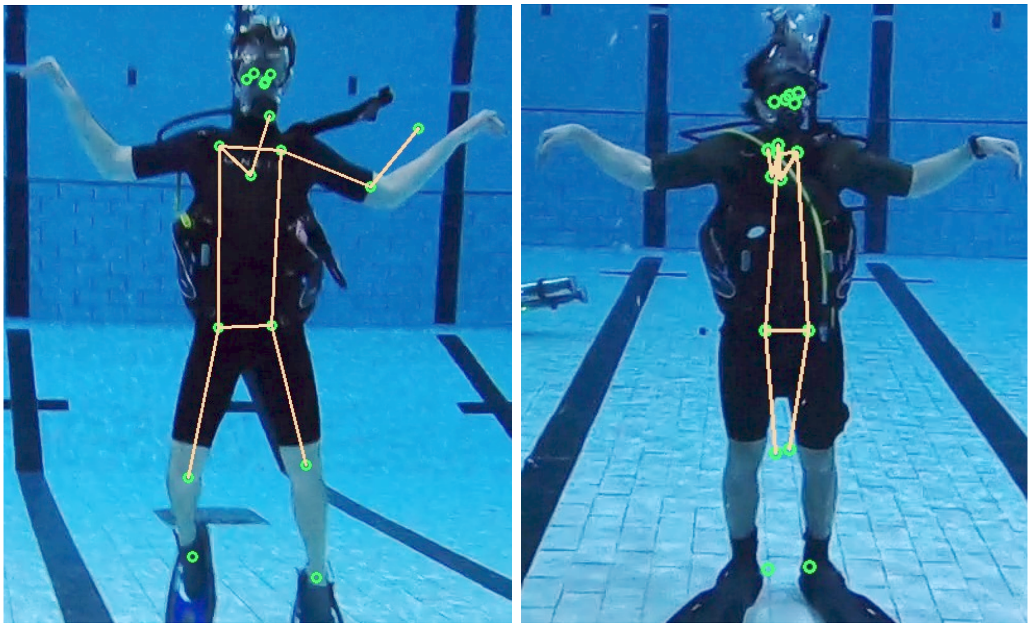


Figure 9.10: Examples of pose estimation failures caused by divers' erratic movements underwater and highly obscured (by dive equipment and bubbles) facial features.

9.4 DiverID: Diver Identification Using Anthropometric Data Ratios

9.4.1 Diver Identification Algorithm

Our identification algorithm consists of four major components: (1) Pose Estimation and Filtering, (2) Feature Extraction, (3) Metric Learning, and (4) Training Models for Metric Learning and Classification. We perform pose estimation on the image frames containing a diver and filter them using anthropometric data statistics. Then, we compute features from the filtered pose estimations to train different models to identify divers. Fig. 9.9 visualizes this process.

Pose Estimation and Filtering

To maximize the probability of getting stable pose estimation, our system requires the divers to take a standing-upright and frontal-facing position (see Fig. 9.9) from the robot's perspective. We use HRNet [231], a robust 2D pose estimation method, to predict the joint locations of divers. While such posture requirements can help to produce good predictions, they cannot completely prevent the pose estimation method from making false predictions. Especially, scuba divers' erratic movements often make it extremely difficult for the pose estimation algorithm to accurately predict the poses. Fig. 9.10 demonstrates two instances where the pose estimation algorithm fails to compute the locations of elbows, wrists, or shoulder joints correctly. Originally HRNet estimates 17 body keypoints, from which we select the following 10 keypoints: *left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left hip, right hip, left knee, right knee.* While including the remaining seven keypoints (*i.e., nose, left eye, right eye, left ear, right ear, left ankle, right ankle*) could make our feature set richer, we observed that their predictions tend to be inconsistent across frames compared to the other 10 keypoints. Additionally, the unselected seven keypoints require the underwater robot to constantly capture the whole body of the diver, which may not be realistic during underwater missions due to environmental factors (*e.g.*, waves). We find that the pose estimation algorithm consistently captures the selected 10 keypoints from our preliminary tests.

To create robust anthropometric data ratio (ADR) features from the predicted pose, erroneous data must be removed from the dataset. We achieve this by formulating a number of filtering conditions motivated by anthropometric data statistics [243]. Specifically, we use the following filtering conditions, where hw_{min}, k_{th}, and sw_{min} are hyperparameters.

- hips are located lower than shoulders.
- knees are located lower than hips.
- hip width must be larger than hw_{min}.
- both hip to knee distances must be similar.
- both elbow to wrist distances must be similar.
- upper arm is slightly longer than the lower arm.
- distance between knees must be larger than k_{th}.

Table 9.1: Euclidean distances between the listed pairs are used to calculate the corresponding anthropometric data (AD).

AD No.	**Pairs**
1	(left shoulder, right shoulder)
2	(left shoulder, left hip)
3	(left shoulder, left elbow)
4	(left elbow, left wrist)
5	(right shoulder, right hip)
6	(right shoulder, right elbow)
7	(right elbow, right wrist)
8	(left hip, right hip)
9	(left hip, left knee)
10	(right hip, right knee)

- shoulder-to-hip distance must be slightly longer than shoulder width.
- shoulder-to-hip distance cannot be twice as large as the hip-to-knee distance.
- shoulder width must be larger than sw_{min}.
- shoulder-to-hip distance must be larger than the sizes of the lower and upper arm.

As a result, we only keep the pose estimations that meet all the filtering conditions.

Feature Extraction

Once we obtain the filtered pose estimations, we calculate 10 anthropometic data (AD) values using the 10 body keypoints, as shown in Table 9.1. While the keypoint predictions are relatively stable, their locations in each frame constantly change as the robot moves. Therefore, keypoints and also the AD values are distance-variant. To address this issue and make the features distance-invariant, we propose to use ADRs as features to represent the divers. We compute the ADRs by taking the ratios of two different AD values, generating a total of 45 ratios from unique pairs, excluding the pairs formed

Table 9.2: Our proposed embedding network, which takes 45-d ADR features and project them into 16-d features.

Input:	ADRs ($F \times 45$)
Layer 1	Linear $(45, 1024)$; Leaky ReLU; BN
Layer 2	Linear $(1024, 512)$; Leaky ReLU; BN
Layer 3	Linear $(512, 256)$; Leaky ReLU; BN
Layer 4	Linear $(256, 16)$

by identical ADs. We observed that the ADRs stay relatively consistent for each diver even if the distance between the robot and the diver changes.

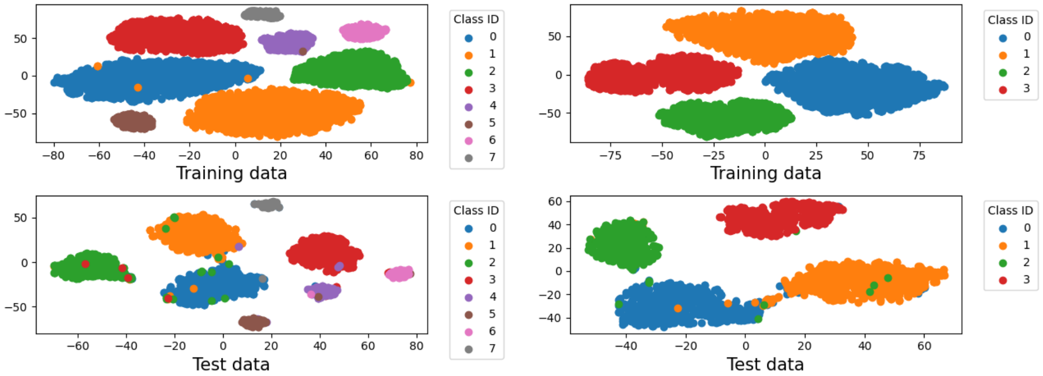
Metric Learning

Metric learning [244] is a method to learn a distance measure for maximally separating inter-class distances in the feature space, using an appropriate distance metric. Examples of classical approaches are the K-nearest neighbor (KNN) [245] and support vector machine (SVM) [246]. We apply metric learning when we train our proposed embedding network (as shown in Table 9.2), and we select the triplet loss [213] $\mathcal{L}$ as our loss function due to its simplicity and effectiveness. Our loss function is formulated as follows:

$$\mathcal{L}(A, P, N) = max(0, \mathcal{D}(A, P) - \mathcal{D}(A, N) + m), \tag{9.2}$$

where A is an anchor point (reference data), P is a positive point, N is a negative point, and m is a predefined margin. For instance, if A is a data point of diver 1, P is another data point of diver 1, and N will be a data point belonging to one of the other divers (*e.g.*, diver$\{2, \cdots, \mathcal{X}\}$). By training our model with this loss function, we enforce intra-class data points (*i.e.*, A and P) to stay close while increasing the distance between inter-class data points (*i.e.*, A and N). In [213], Euclidean distance was used for the distance function $\mathcal{D}$. While this is one of the common distance metrics, it cannot capture nonlinearity in the data. Thus, we use cosine similarity as a distance metric in our work.

Additionally, we created the following datasets for training the embedding network: (1) All-class dataset (four divers and four swimmers) and (2) Diver dataset (four divers



(a) Trained on the All-class dataset.

(b) Trained on the Diver dataset.

Figure 9.11: Results of the metric learning technique (Section 9.4.1) with our proposed embedding network. We visualize the 16-d features in 2D plot using t-SNE technique. We train our embedding network with (a) All-class dataset and (b) Diver dataset. The top row shows the clustering results on the training data, and the bottom row shows the clustering results on the test data. In both (a) and (b), the clusters show a clear separation between the classes for both datasets.

only). The datasets are described in more detail in Section 9.5. Our embedding network takes 45-d features and outputs 16-d feature vectors. Interestingly, upon plotting these 16-d features from both training and test sets, they are observed to form well-defined clusters (see Fig. 9.11). We use the t-SNE [247] technique to plot the 16-d features in a 2D plot.

Training Models for Metric Learning and Classification

We use 10 models to evaluate the performance of the diver identification algorithm (as shown in Table 9.3). The model names starting with *All* are trained with the All-class dataset, and those starting with *Diver* are trained with the Diver dataset. For All_KNN, Diver_KNN, All_SVM, and Diver_SVM models, we use only either KNN or SVM to train for classification without using an embedding network. For All_NN_KNN, Diver_NN_KNN, All_NN_SVM, Diver_NN_SVM, All_NN, and Diver_NN models, we first pass the raw 45-d ADR features through our embedding network to generate highly distinguishable 16-d features. We then train classifiers with the 16-d features for identification. In the case of the All_NN and Diver_NN models, we use a two-layer neural network, attached with a final softmax layer to perform the multi-class classification

Table 9.3: Network structure and online training capabilities of different models used for our diver identification framework.

Model Name*	**Embedding**	**Classification**	**Offline**	**Online**
All_KNN	-	KNN	✓	-
Diver_KNN	-	KNN	✓	✓
All_SVM	-	SVM	✓	-
Diver_SVM	-	SVM	✓	✓
All_NN_KNN	NN	KNN	✓	✓
Diver_NN_KNN	NN	KNN	✓	✓
All_NN_SVM	NN	SVM	✓	✓
Diver_NN_SVM	NN	SVM	✓	✓
All_NN	NN	NN	✓	-
Diver_NN	NN	NN	✓	-

*All_KNN and Diver_KNN refer to the same model for online training. Same goes for All_SVM and Diver_SVM.

using the 16-d features.

9.4.2 Diver Identification Framework

With our proposed diver identification algorithm, we develop a diver identification framework (see Fig. 9.12) using ROS [248]. The framework enables a robot to find divers and identify them, and it is designed to be compatible with both offline (models are trained before deployment) and online frameworks (models are trained during deployment). The goal of our proposed framework is to find a 'target' diver. A target diver will be determined differently for each framework: (1) Offline framework: one of the divers in the dataset will be assigned as a target diver before deployment, and (2) Online framework: the robot will not have a target diver before its deployment and will be assigned a target diver during its mission. Our framework consists of a state machine, a diver-relative position (DRP) estimator, and a robot controller, which is built based on Autonomous Diver-Relative Operator Configuration (ADROC) [249] pipeline. The DRP estimator [249], which is running in the background, approximates the distance to a diver from the robot using pose estimations and bounding box detections of the diver. When the distance information becomes available, *i.e.*, when the robot detects a

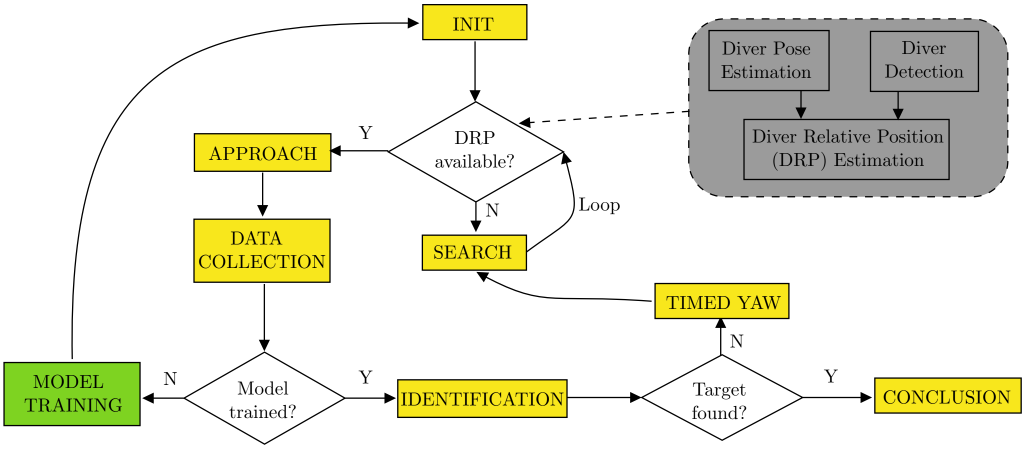


Figure 9.12: Complete pipeline of our proposed diver identification framework as described in Section 9.4.2. The yellow and green color boxes represent the states of our framework. The yellow color boxes represent common states between the offline and online frameworks. The green color box is only used for the online framework to perform model training during deployment. The gray colored box shows the DRP estimation which is continuously running in the background.

diver, its motion is governed through a PID controller using information from the state machine.

To integrate the identification algorithm, we use a state machine with the following eight states: INIT, SEARCH, APPROACH, DATA COLLECTION, TIMED YAW, MODEL TRAINING, IDENTIFICATION, and CONCLUSION. For both offline and online frameworks, our system starts from the INIT state and enters either the SEARCH or APPROACH state depending on the availability of the DRP estimation. In the following sections, we present how each framework works in detail.

Offline Framework

After the state machine starts in the INIT state, it immediately transits and stays in the SEARCH state until the DRP estimator detects a diver and provides DRP estimations. Once the DRP estimations become available, the robot enters the APPROACH state. In this state, the robot attempts to maintain a preset distance between itself and the diver. After reaching the desired distance, the robot enters the DATA COLLECTION state, where the robot computes the 45-d ADR features from F (a predefined number) *filtered* pose estimations. With the ($F \times 45$) ADR features, the robot goes into the IDENTIFICATION state since the models are already trained in this framework. If the robot fails to match the currently observed diver with a target diver, the robot enters the SEARCH state again after staying briefly in the TIMED YAW state. We add the TIMED YAW state to prevent the robot from performing identification on the same diver again. This will be repeated until the robot identifies a target diver. If successful, it will go to the CONCLUSION state.

Online Framework

The online framework operates the same way as the offline framework up to the DATA COLLECTION state. Since the models are not trained at the beginning, the state machine enters the MODEL TRAINING state from the DATA COLLECTION state. In this state, the models (*e.g.*, SVM and KNN) are trained using ($F\mathcal{X}\times45$) ADR features collected from $\mathcal{X}$ number of divers. Since we only train SVM and KNN classifiers at this state, the training time is acceptable on typical robotic hardware (*e.g.*, a mobile GPU or CPU). Once the training is done, a target diver will be chosen. In our evaluation, we arbitrarily select the second diver the robot sees as a target. However, our framework has the flexibility to update the target diver as needed, *e.g.*, through specific hand gestures, or human-to-robot interaction. Then, the robot goes to the INIT state, and the state machine subsequently follows the same logic as the offline framework since the models are now trained.

9.5 Experiments

This section describes experimental setups for evaluating our proposed algorithm including dataset creation, model building and training, and robot trials with divers. Human data collection and trials were conducted with the approval of our institution's ethics oversight panel.

9.5.1 UDI Dataset

To facilitate the learning of our proposed algorithm, we need to create a dataset that contains anthropometric data (AD) of divers. Since it is extremely challenging to find and recruit many certified divers to collect such data in a closed-water environment (*e.g.*, pool), we collect the AD data from both swimmers and divers. In total, we have data

Table 9.4: Comparison of average accuracy for the metric learning embedding network and the 10 models on our UDI dataset. The values are in percentages.

	Embedding Network		**Models**	
Dataset Type	**All-Class**	**Diver**	**All-Class**	**Diver**
Training Accuracy	97.78	97.89	97.21	97.97
Testing Accuracy	96.39	96.65	97.56	97.89

from four swimmers and four divers. To build this dataset, we ask the participants to take frontal-facing and standing-upright postures towards the camera, and we capture their full body images using GoPro cameras with 2-megapixel and 1080p resolution at varying distances. We collect a total of $16,557$ images across the eight participants. We then perform pose estimation on these images, filter out erroneous poses, and compute the ADR features. This gives us a total of $13,994$ ADR features and their corresponding labels, which we use to formulate the underwater diver identification (UDI) dataset. For the filtering step, as described in Section 9.4, we heuristically set hw_{min}, k_{th}, and sw_{min} as 10.

9.5.2 Setup

Model Setup for Offline Framework

We use two versions of the UDI dataset to train the models for the offline framework: (1) All-class dataset (whole UDI dataset) and (2) Diver dataset (a subset of the UDI dataset containing only divers). For All_KNN and All_SVM, we use the whole UDI dataset to train. We use scikit-learn libraries [250] to implement the KNN and SVM. For the KNN models, we set the neighbor size as 5. We train our metric learning embedding network (as shown in Table 9.2) for $1,000$ epochs with a batch size of 512, learning rate of 5×10^{-4}, and margin, $m = 0.3$. We use stochastic gradient descent (SGD) as the optimizer [251]. The embedding network is used to train the All_NN_KNN, Diver_NN_KNN, All_NN_SVM, and Diver_NN_SVM models where it works as a backbone. That is, the embedding network first brings the 45-d feature vectors down to 16-d feature vectors then train the final KNN and SVM models with the 16-d features. Finally, to train the All_NN, and Diver_NN models, we add a two-layer classification neural network after the embedding network and train them together. We use PyTorch libraries [252] to implement both the embedding and the two-layer classification neural network. All the models in the offline framework are trained on 80% of the data and tested on 20% of the data. Table 9.4 presents the average training and testing accuracy for both the embedding network and 10 models.

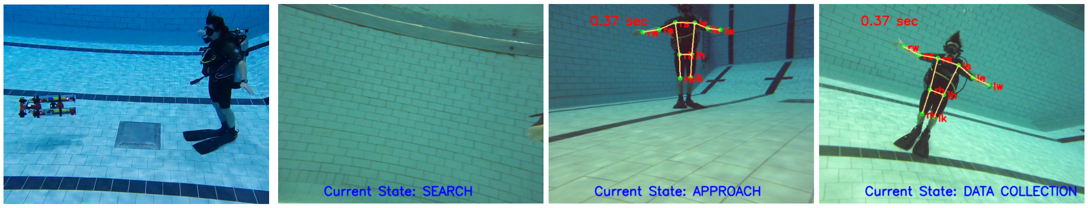
Model Setup for Online Framework

The six models (see Table 9.3) for the online framework can be trained with either (1) 45-d raw ADR features (Diver_KNN and Diver_SVM) or, (2) 16-d features computed using the pre-trained embedding network (All_NN_KNN, Diver_NN_KNN, All_NN_SVM, and Diver_NN_SVM). Additionally, we train the models with features computed from either 50 or 100 frames to evaluate the correlation between the number of training data and the performance of each model. To keep the online training time short, we do not consider training the All_NN model.

9.5.3 Experimental Scenarios

We use three divers in a closed-water environment to perform our experiments using the offline and online frameworks. In the offline framework, our AUV collects ADR features from a diver and then feeds the features through pre-trained models and gets a prediction. If the prediction is accurate, then the algorithm concludes and enables the robot to initiate an interaction with the diver. Otherwise, the robot turns to the next diver. The robot repeats this for all the divers present in the scene. We assume that the robot knows *a priori* the total number of divers present in the scene.

In the online framework, the robot aims to identify a diver without using pre-trained models. To achieve this, first, the robot collects the ADR features from all the observable divers. It then trains six models online, to create highly separable clusters of divers. Upon being assigned a target diver (in the online case, a random selection from the present divers), the robot uses these newly trained models to identify divers individually. Whenever there is a correct match, the algorithm concludes without checking the rest of the divers. If the robot is unable to find a correct match after checking all divers, it is considered as a failure case.



(a) Third person view of the diver identification process.

(b) Robot's view during different states of the diver identification process.

Figure 9.13: Demonstration of the proposed diver identification framework on a physical AUV platform in a closed-water environment. Fig. (a) shows an external view of the diver interaction process, while Fig. (b) shows some of the states in our framework from the robot's viewpoint.

9.6 Results

We deploy the system onboard an AUV in a closed-pool environment and evaluate the performance of the proposed system in both offline and online frameworks. Fig. 9.13 visualizes the overall diver identification framework during our deployment. As discussed in Section 9.4.2, the AUV goes through a number of states for identifying a target diver. The AUV's view during three such states (SEARCH, APPROACH, and DATA COLLECTION) are shown in Fig. 9.13b.

9.6.1 Offline Framework

In the offline framework, the models have pre-trained weights. Whenever they are given test features to infer on, they will be able to make the prediction instantaneously. We test the performance of all 10 models pre-trained with the All-class and Diver dataset (see Table 9.3) on three divers placed in a closed-water environment. Table 9.5 and Fig. 9.14a present the quantitative results of our experiments. Table 9.5 lists the average accuracy of different models as the number of frames on which the final predictions are performed changes. We notice that All_NN_KNN and All_NN_SVM perform the best. This is because both these models use metric learning as their backbone to first project the raw ADR features into lower dimensional (16-d) feature vectors that form highly separable clusters in a 16-d hyperspace. These projected features are then used to perform the final classification task. The bottom plot in Fig. 9.14a shows the average accuracy of each model across all numbers of test frames. From the figure, the superior

Table 9.5: Comparison of the diver identification accuracy of the proposed system across 10 different models. The darker colors represent higher accuracy.

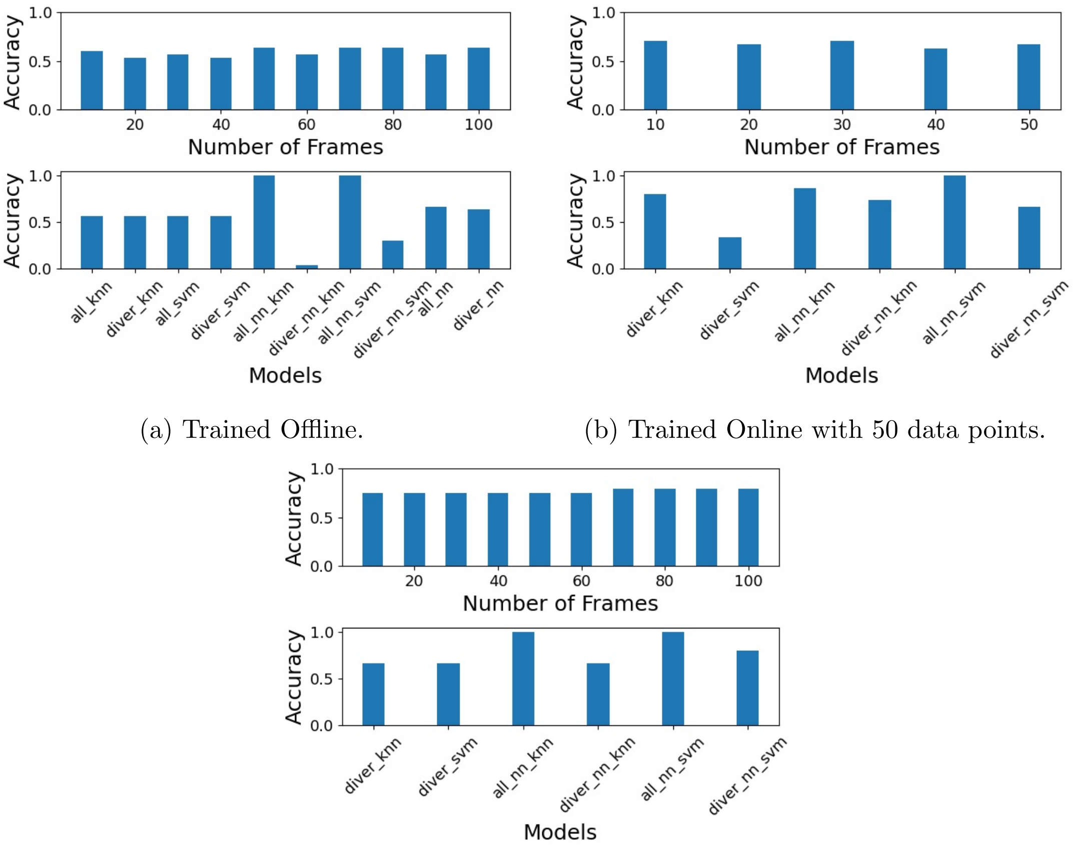
Model Name	All_KNN			Diver_KNN			All_SVM			Diver_SVM			All_NN_KNN		
Framework	Offline	Online		Offline	Online		Offline	Online		Offline	Online		Offline	Online	
Number of Frames		50	100		50	100		50	100		50	100		50	100
10	0.67	-	-	0.67	0.67	0.67	0.67	-	-	0.33	0.33	0.67	1.00	1.00	1.00
20	0.33	-	-	0.33	1.00	0.67	0.67	-	-	0.67	0.33	0.67	1.00	0.67	1.00
30	0.33	-	-	0.33	1.00	0.67	0.67	-	-	0.67	0.33	0.67	1.00	1.00	1.00
40	0.33	-	-	0.33	0.67	0.67	0.33	-	-	0.67	0.33	0.67	1.00	0.67	1.00
50	0.67	-	-	0.67	0.67	0.67	0.67	-	-	0.67	0.33	0.67	1.00	1.00	1.00
60	0.67	-	-	0.67	-	0.67	0.33	-	-	0.33	-	0.67	1.00	-	1.00
70	0.67	-	-	0.67	-	0.67	0.67	-	-	0.67	-	0.67	1.00	-	1.00
80	0.67	-	-	0.67	-	0.67	0.67	-	-	0.67	-	0.67	1.00	-	1.00
90	0.67	-	-	0.67	-	0.67	0.33	-	-	0.33	-	0.67	1.00	-	1.00
100	0.67	-	-	0.67	-	0.67	0.67	-	-	0.67	-	0.67	1.00	-	1.00

Model Name	Diver_NN_KNN			All_NN_SVM			Diver_NN_SVM			All_NN			Diver_NN		
Framework	Offline	Online		Offline	Online		Offline	Online		Offline	Online		Offline	Online	
Number of Frames		50	100		50	100		50	100		50	100		50	100
10	0.33	1.00	0.67	1.00	1.00	1.00	0.33	0.67	0.67	0.67	-	-	0.33	-	-
20	0.00	0.67	0.67	1.00	1.00	1.00	0.00	0.67	0.67	0.67	-	-	0.67	-	-
30	0.00	0.67	0.67	1.00	1.00	1.00	0.33	0.67	0.67	0.67	-	-	0.67	-	-
40	0.00	0.67	0.67	1.00	1.00	1.00	0.33	0.67	0.67	0.67	-	-	0.67	-	-
50	0.00	0.67	0.67	1.00	1.00	1.00	0.33	0.67	0.67	0.67	-	-	0.67	-	-
60	0.00	-	0.67	1.00	-	1.00	0.33	-	0.67	0.67	-	-	0.67	-	-
70	0.00	-	0.67	1.00	-	1.00	0.33	-	1.00	0.67	-	-	0.67	-	-
80	0.00	-	0.67	1.00	-	1.00	0.33	-	1.00	0.67	-	-	0.67	-	-
90	0.00	-	0.67	1.00	-	1.00	0.33	-	1.00	0.67	-	-	0.67	-	-
100	0.00	-	0.67	1.00	-	1.00	0.33	-	1.00	0.67	-	-	0.67	-	-

performance of All_NN_KNN and All_NN_SVM is evident. Finally, we calculate the average accuracy across all the models (the top plot in Fig. 9.14a) and notice that the accuracy remains almost the same. This suggests that All_NN_KNN and All_NN_SVM can even make faster predictions using a lower number of frames while maintaining their high accuracy.

9.6.2 Online Framework

In the online framework, we do not use any pre-trained weights for the models. Instead, the system first computes the ADR features from three divers, trains six models with online training capability (see Table 9.3), and makes predictions on the three divers using these trained models. We first use 50 frames *per* diver to train the models and evaluate the diver identification performance. From Table 9.5 and Fig. 9.14b (bottom), we notice that All_NN_SVM performs the best and All_NN_KNN shows comparable performance. That means the metric learning technique we are using to train these



(a) Trained Offline.

(b) Trained Online with 50 data points.

(c) Trained Online with 100 data points.

Figure 9.14: Average accuracy from our experiments: (Top row): average accuracy for a specific number of test frames across all models, (Bottom row): average accuracy of each model across all frames.

two models are indeed benefiting the identification task. Even if we change the number of frames *per* diver to train the models to 100, we see similar trends. From Table 9.5 and Fig. 9.14c (bottom), we see that both the All_NN_SVM and All_NN_KNN achieve the highest accuracy. Finally, from Fig. 9.14b (top) and 9.14c (top), we notice that the number of frames used for making inferences does not have any significant effect on the system's performance for the online framework, as was the case for the offline framework.

As these results suggest, our diver identification framework can successfully identify divers within both the offline and online frameworks. In both cases, the All_NN_KNN and All_NN_SVM models outperform the other models. Especially, the All_NN_SVM model shows consistently accurate identification performance regardless of the type of the framework and the number of frames for training and inference. Additionally, we show that the models with our embedding network achieve higher classification accuracy than the models without the network. More importantly, our results demonstrate that the ADR features are distance and photometric-invariant considering the fact that (1) our data has been captured at varying distances, and (2) images from the camera (0.9-megapixel, 720p resolution) that we used to build our training dataset with and the robot camera (2-megapixel, 1080p), are significantly different. Fig. 9.13a and Fig. 9.13b show sample pictures taken by these two cameras, respectively.

9.6.3 AUV Trial Failure Cases

The PID controller in our robot is highly sensitive to proper parameter tuning. As a result, the robot occasionally fails to stabilize its motion which resulted in unexpected behavior of our state machine. For instance, during some experiments, the state machine could not reach the IDENTIFICATION state at all, and remained between the SEARCH and APPROACH states. However, when the controller remained stable, our robot successfully identified divers by correctly following the logic of our state machine. In this paper, we focus on developing a robust identification algorithm and leave the improvement of the robot controller as future work.

9.7 Conclusions

In this chapter, we present two novel methods to identify divers using facial features and ADR features, respectively. For the diver identification system, we leverage a series of data augmentation steps and a dimensionality reduction method to learn discriminative features from heavily-obscured scuba diver faces. Our results demonstrate that the proposed pipeline outperforms the SOTA face recognition algorithms for the diver identification task. For the diver identification with the proposed ADR features, we design an embedding network that can project the ADR features to a 16-d hyperspace, making them highly separable. We incorporate our diver identification algorithm within both offline and online frameworks. This enables the AUV to identify divers even if we do not have their data in our dataset prior to deployments. The results show that the ADR features can be captured and used for identification even if divers' bodies and faces are heavily obscured. Together, these diver identification methods make substantial progress toward enabling secured HRC missions. This chapter marks the end of Part III. In Part IV, we discuss robotic platforms that we designed to develop and test research presented in this thesis.

Part IV: Robotic Systems Development Toward On-board Object Detection

Many AUVs have been introduced since the appearance of the first AUV in 1957. However, only a small number of AUVs are open-source, and most of such AUVs' computational power is not powerful enough to run state-of-the-art computer vision and machine learning models. To handle this matter, Chapter 10 presents two open-source robotic systems designed for developing, deploying, and testing robotic algorithms for underwater environments. First, we introduce LoCO, a low-cost open-source AUV. LoCO is a general-purpose, vision-guided AUV rated to a depth of 100 meters, and a small size AUV that is single-person-deployable. LoCO is composed of off-the-shelf and 3D printed parts. Additionally, it has an open and modular design that can be expanded to provide additional capabilities. This feature allows users to install additional sensors as needed, unlike most proprietary AUVs. The other system is a data collection device called HydroEye. We design HydroEye to reduce the overhead of robotic deployments and assist a scuba diver in collecting visual and other sensory data. HydroEye is a single-person-portable device, and it provides instant access to the collected data via either Wi-Fi or tether. Both systems have been intensively used to develop the algorithms presented in this thesis.

Chapter 10

Robotic Systems Development Toward On-board Object Detection

While AUVs are a critical instrument for conducting research underwater, they are generally costly and hard to maintain. In this chapter, we propose two open-source robotic systems developed to assist in the presented research of this thesis as well as to lower the barrier to researching the underwater domain. One of them is a Low-Cost and open-source AUV (LoCO). The other is a device (HydroEye) specifically designed to aid a scuba diver in collecting visual and other sensory data. In the following sections, we provide the details of the AUV and device.

10.1 Motivation

The underwater domain is a high-risk environment for robots (*e.g.*, risk of water leaks and AUV losses during deployments). Especially when we test under-development algorithms, the risk is even higher. Additionally, commercially available AUVs are proprietary and less modular. For example, with our Aqua AUV [135], because of its tight packaging and power constraints, it is extremely difficult to add more sensors without compromising other systems. These factors have motivated us to develop a low-cost, open-source, modular, and expandable AUV to support our research with low maintenance costs and development overheads.

Person-portable devices [253, 254, 255] make it more convenient to gather data than using robots since it avoids the overhead of robot deployments (*e.g.*, arranging tether cables for remotely operated vehicles (ROVs), preparing sets of batteries for AUVs, and finding a safe spot to deploy AUVs). It is especially useful to use the device if an environment is challenging to navigate with ROVs and AUVs (*e.g.*, cave exploration [254]). Because of these reasons, we have also investigated and built a specialized device that can collect sensory data, which we discuss more in detail in Section 10.3.

10.2 LoCO

LoCO [167] [1] is designed to be an all-purpose AUV, adaptable to a variety of missions. It is an open platform so additional sensors can be installed. In its base configuration, LoCO is a dual-camera, vision-guided AUV with three thrusters. The following subsections describe the robot's system design and software, which enable us to deploy the proposed algorithms.

10.2.1 System Design

Overall System Layout

LoCO is comprised of two water-tight enclosures, each containing various components, with one thruster mounted between the enclosures and two mounted behind, as seen in Fig. 10.1. While a number of designs are considered, we select the two-enclosure design for a variety of reasons. Firstly, it provides space in between the enclosures, which can be used to mount sensors, thrusters, or manipulators in the future. Additionally, we can distribute components into two enclosures, allowing us to use smaller enclosures than the design with one enclosure. This two-enclosure design reduces the LoCO's forward profile, and less resistance will be applied to LoCO when moving forward. Finally,

[1] The author has contributed as one of the main developers and worked on configuring a mobile GPU, designing the power distribution system, and developing an autopilot system.

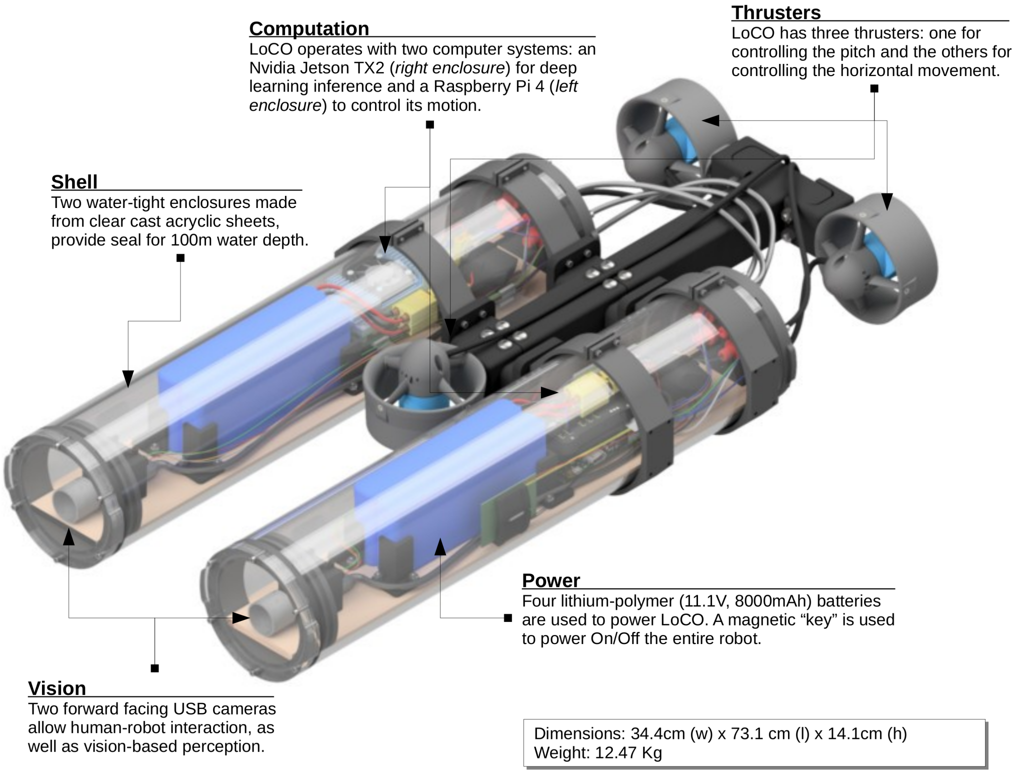

Computation
LoCO operates with two computer systems: an Nvidia Jetson TX2 (*right enclosure*) for deep learning inference and a Raspberry Pi 4 (*left enclosure*) to control its motion.
Thrusters
LoCO has three thrusters: one for controlling the pitch and the others for controlling the horizontal movement.
Shell
Two water-tight enclosures made from clear cast acryclic sheets, provide seal for 100m water depth.
Power
Four lithium-polymer (11.1V, 8000mAh) batteries are used to power LoCO. A magnetic "key" is used to power On/Off the entire robot.
Vision
Two forward facing USB cameras allow human-robot interaction, as well as vision-based perception.
Dimensions: 34.4cm (w) x 73.1 cm (l) x 14.1cm (h)
Weight: 12.47 Kg

Figure 10.1: 3D model of LocO AUV with IMU, camera, thruster, and robot frames. *Image courtesy of the IRV Lab.*

the separation into two enclosures enforces a base level of modularity. Most control-related electronics are in the left-hand enclosure, with the computational hardware for deep learning inference in the right-hand enclosure. While any hardware modification requires changes throughout the system, changes or replacements can be made with minimal impact on the layout of internal components.

Computing Systems and Sensors

LoCO has two primary computer systems: a Jetson TX2, located in the right enclosure for deep learning inference, and a Raspberry Pi 4 (4GB), located in the left for control. The Jetson TX2 is mounted on a Connect Tech Orbitty carrier board for interfacing. The Raspberry Pi is used to process images from the camera in the left enclosure and used as the controller interfacing with the Pixhawk autopilot over a serial connection. The Jetson and the Raspberry Pi are connected via Ethernet with a Cat5e cable. LoCO was designed with a stereo vision system in mind. We are currently investigating several stereo cameras, but finding the camera small enough for the enclosure's diameter has proved challenging. Currently, two monocular cameras are mounted in each enclosure. LoCO also employs a pressure sensor to measure its depth under the surface. The sensor is mounted on the backplate of the left enclosure. There is also an inertial measurement unit (IMU) contained within the Pixhawk autopilot unit. Lastly, while it is not a part of the design, we are currently working on integrating a sonar altimeter, as it is needed for our proposed localization algorithm in Chapter 7.

10.2.2 Software

LoCO has a variety of computing devices: a Raspberry Pi 4, an Nvidia Jetson TX2, a Pixhawk autopilot unit, and an Adafruit Trinket Pro microcontroller. The Raspberry Pi and Jetson TX2 both run versions of Ubuntu, a popular open-source operating system. They both have ROS, and the Jetson TX2 has deep learning libraries (*e.g.*, Pytroch [256] and Tensorflow [257]) with a ROS interface, which enables the Jetson TX2 to run deep learning inferences using images coming from the Raspberry Pi. The Pixhawk runs a real-time open-source OS with ArduSub, and the Trinket is flashed directly with open-source software. The software which allows LoCO to function as an untethered autonomous vehicle rather than an ROV is distributed across the computing devices and consists of a mixture of ROS packages, Ardusub, and Arduino code. All of this software is under some form of permissive, open-source license, which makes LoCO's software stack completely accessible for users to explore, expand, and enhance.

State Estimation

Several options are under development for state estimation in LoCO. Firstly, the Pixhawk provides an IMU-based state estimation using the Extended Kalman Filter [258]. While this is useful, LoCO currently uses the `robot_localization` [259] ROS package to estimate its orientation via IMU data directly from the Pixhawk's IMU, as this provides greater control over the tuning of the Extended or Unscented Kalman filters provided by the package. Additionally, the package allows the fusion of multiple sources of information into one estimate, so if additional IMUs or sources of information became available, they could easily be integrated. Two possible sources of this information are currently in development: a downward-facing camera for monocular visual odometry and an odometry estimate based on a combination of thruster inputs and a hydrodynamic motion model of LoCO.

10.3 HydroEye

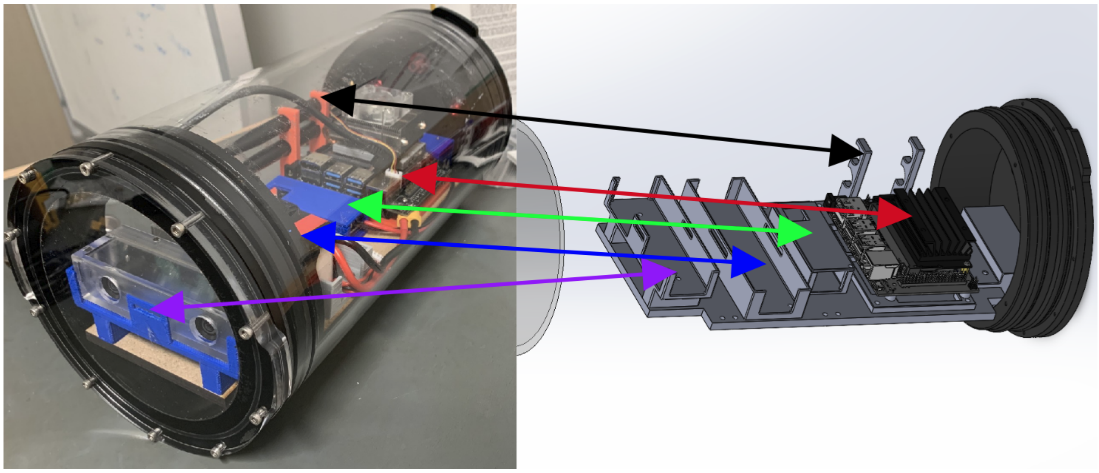


Figure 10.2: Fully assembled HydroEye on the left side and the 3D CAD model on the right side. Purple: a stereo camera, Blue: a battery, Green: a buck converter, Red: a mobile GPU, and Black: a Wi-Fi antenna.

While a single person can deploy LoCO, it still has overheads in preparation for deployments (*e.g.*, charging a set of four batteries, ballasting each tube, and checking wiring connections). To reduce the overheads and accommodate a variety of off-the-shelf sensors, we have created a custom data collection device called HydroEye (Fig. 10.2), which is designed to be hand-carried by a swimmer or a scuba diver to assist in data collection. Our goal is to reduce the time and workforce overhead in data collection. We design it with the following desiderata:

- **Portability**: The device needs to be sufficiently small to be carried by a single person underwater.

- **Minimum operation time**: The device should record data for at least an hour, which is a typical dive duration from a scuba tank.
- **Easy access to the collected data**: We occasionally need to check the quality of the recorded data when a diver rises back to the surface. Thus, it is essential to access the data without opening the device to minimize the risk of water leaking.
- **Buoyancy control**: The device in salt water is more buoyant than the one in fresh water. To deploy the device in different water bodies, the weight of the device needs to be easily adjustable.
- **Easy to start and stop data recording**: The recorded data can take large disk space for storage. Thus, we should be able to start and stop the recording process as needed to minimize disk usage.

10.3.1 System Design

HydroEye consists of a water-tight enclosure, two end caps, laser-cut medium-density fiberboard (MDF), 3D printed mounts, and electrical parts. We use a 14.8 V lithium-poly (LiPo) battery to power the system. The buck converter steps down the voltage from 14.8 V to 5.2V, which is the required input voltage of our mobile GPU. The stereo camera is powered by the mobile GPU via a USB port. Our bench test results show that the device can record data for up to four hours on average. Table. 10.1 presents the detailed specification of each component.

Table 10.1: Electrical Parts list of HydroEye

Parts Name	Power-related Specification
Buck Converter	Input DC 9-24V, Output 5.2V/6A/30W
Battery	4S 14.8V LiPo Battery 2200mAh, Discharge Rate: 50C
Mobile GPU	Input 5V and up to 4A, 10W and 5W modes
Stereo Camera	Power Consumption Max 1.15W, Min 0.77W, Current Max 1.4A, Min 0.4A

10.3.2 Deployment and Usage

Before deploying HydroEye underwater, we first connect the battery to the buck converter and assemble the enclosures. When the GPU is on, it activates its access point

(AP), also known as mobile hotspot. Then, we connect to the AP and start our recording process over Secured Shell (SSH) using a laptop. After these steps, either a scuba diver or swimmer carries the device by hand and collects data as needed (Fig. 10.3). Once data collection is done, we can reconnect to the AP and transfer the data over SSH. This allows us to check the quality and integrity of the data during field deployments without disassembling the enclosure, which can potentially cause water leaks in the system. Additional data can be collected by starting the process via SSH again. When the data collection is completed, the device will be powered down by removing the rear-end cap.

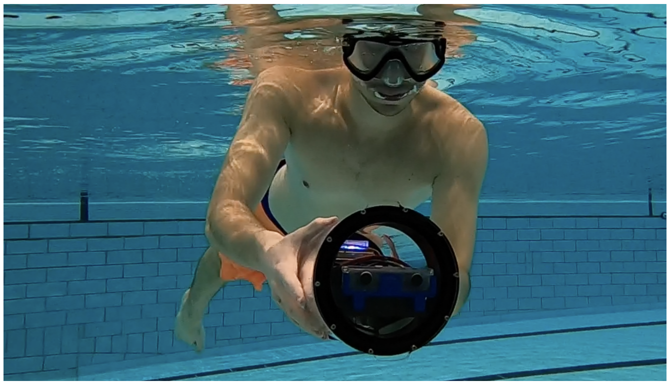


Figure 10.3: A swimmer holds HydroEye and collects the data during a pool evaluation.

10.4 Conclusion

In this chapter, we describe two robotic systems, an AUV (LoCO) and a data collection device (HydroEye), to aid us in developing and deploying our algorithms (*e.g.*, object detection, localization, and exploration). We briefly present both systems and their underlying design philosophies. In the next chapter, we summarize the presented research in this thesis and discuss potential research directions to enhance the autonomy of robots further.

Chapter 11

Conclusion and Future Work

The research presented in this thesis has focused on understanding the underwater domain's unique robotic challenges, addressing them with algorithms, and implementing these algorithms on real robotic platforms. The algorithms empower field robots to manage large-scale and time-sensitive tasks without depending on commands from operators. In this chapter, we will summarize our presented research, introduce the future directions of the research, and conclude with final remarks.

11.1 Summary of Presented Research

Part I, which consists of Chapter 2 through Chapter 5, introduced significant first steps toward overcoming perceptual problems in challenging environments. In Chapter 2, we presented the first approach of using a deep learning-based method [29] to detect debris with AUVs and address visual challenges underwater. Chapter 3 presented *TrashCan* [30]. It is the first publicly available underwater debris image dataset and consists of $7,212$ pixel-level annotated images of debris, robots, and a wide variety of undersea flora and fauna. Collecting imagery of underwater debris is inherently difficult and dangerous to divers, and building such a dataset requires a work-intensive process. This makes *TrashCan* a substantial contribution to the robotics community and beyond. In Chapter 4, we presented a generative approach [31] to producing realistic fake debris images to augment existing datasets. This approach minimizes the data collection efforts, which results in reducing the risk and labor issues of such efforts. Chapter 5 took one step further from augmenting existing datasets. We presented a novel image blending method generating images and their pixel-level annotations together. This significantly reduces labor-intensive dataset labeling efforts for creating datasets to train object detection models.

Part II, which is composed of Chapter 6 and Chapter 7, brought forth novel methods to find and identify objects underwater. Our research introduced in Chapter 6 fuses depth and instance segmentation information with an APF. This research enables robots to keep flexible distances to objects using the semantic information of the objects. In Chapter 7, we presented a low-cost approach by fusing bathymetry data of underwater environments with depth and altitude data from an AUV using real-world bathymetry data. This approach enables the localization of AUVs using low-cost sensors.

Part III, which consists of Chapter 8 and Chapter 9, introduced novel algorithms providing AUVs with HRC capability by enabling them to approach and identify divers. These algorithms can reduce the physical and cognitive load of divers as well as augment robotic autonomy with human knowledge. Chapter 8 presented the first underwater human-approaching robotic algorithm without making divers wear special sensors or move. This algorithm allows AUVs to find a diver and achieve diver-relative positioning autonomously with only a monocular vision and prior knowledge of average human shoulder width. Chapter 9 introduced two novel diver identification algorithms for AUVs using divers' face and body pose information with a visual sensor. These algorithms are designed to allow AUVs to perform HRC tasks securely by collaborating with only authorized divers, even when there are multiple divers around them.

Part IV, which is composed of Chapter 10, presented two robotic systems designed for developing, deploying, and testing robotic algorithms for underwater environments. These systems have been intensively used to develop the research presented in this thesis. In Chapter 10, we first introduced LoCO, a low-cost open-source AUV. LoCO is a general-purpose, vision-guided AUV rated to a depth of 100 meters and a small size AUV that is single-person-deployable. LoCO is composed of off-the-shelf and 3D-printed parts. Additionally, LoCO has an open and modular design that can be expanded to provide additional capabilities. This feature allows users to install additional sensors as needed, unlike most proprietary AUVs. The second system was a data collection device called HydroEye. We designed HydroEye to reduce the overhead of robotic deployments and assist a scuba diver in collecting visual and other sensory data. HydroEye is a single-person-portable device, and it provides instant access to the collected data via either Wi-Fi or tether.

11.2 Future Work

Multiple directions can be considered to extend the research presented in this thesis.

11.2.1 Efficient learning for robotic perception

Humans can learn about new objects from only a few images or simple descriptions. However, robotic perception algorithms often require training with thousands of images to recognize objects accurately, as shown in Part I. This is because the general perception algorithms do not utilize any background information that can aid their learning process. One future direction for improvement is making object detectors less dependent on datasets to tackle this limitation. Zero-shot learning (ZSL) is one potential approach for this. It links unseen and seen objects using supplementary information, which embeds object features. Using language descriptions of objects as the supplementary information can be a promising start. Furthermore, another potential area for improving robotic performance lies in designing convolutional neural networks light enough to run this process on-board AUVs with acceptable performance.

11.2.2 Risk-aware semantic mapping

Demands for embedding semantic information of surroundings to a map have increased as robots need to perform complex tasks (*e.g.*, grasping and transporting). However, there is still a gap between extracting and utilizing the semantic information from the semantic map. One viable direction for future refinement is developing a model to predict risks given the semantics of observed objects and the geometry of surroundings. Such a model would bridge the gap and allow robots to assess risks continuously during their tasks. This would potentially provide safer navigation capability to robots.

11.2.3 Physical interaction ability for robots

Robots need the means to interact with their environment to handle physical tasks autonomously, and so it is crucial to add a manipulator to AUVs to augment the physical interaction ability of the AUVs. While AUVs with manipulation capability already exist, open-source AUVs supporting such capability are not present. The addition of the manipulator to the AUVs will play a key role in developing a robotic system that can perform tasks (*e.g.*, picking up underwater debris autonomously after finding and identifying them) fully autonomously. Using LoCO (Chapter 10) as a robotic platform to add a manipulator would be a good starting point for developing such open-source AUVs. There are several factors that need to be considered for adding a manipulator to LoCO, including material, the size of the manipulator, and installation location on the AUVs. Therefore, collaborating with mechanical systems experts to optimize the design for AUVs would be another possibility to pursue this research direction.

11.3 Concluding Remarks

Our research presented in this thesis gives a new perspective on developing deep learning models for underwater robotics, and the outcome is highly generalizable. This research can be used to operate robots in any visually degraded cases, such as driving in severe weather, underwater pipeline inspection, search and rescue with drones in bad visibility, and more. We believe we will have robotic systems that can explore unstructured environments in the near future, and we hope this thesis contributes towards realizing such systems. More importantly, we would be pleased if our research could benefit society through addressing sustainable environmental protection, thusly improving human well-being.

References

[1] Rich Horner. The ocean currents brought us in a lovely gift today... - YouTube. https://www.youtube.com/watch?v=AWgf0ND2y68. Accessed 09-16-2018.

[2] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson Image Editing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, page 313–318, New York, NY, USA, 2003. Association for Computing Machinery.

[3] Lingzhi Zhang, Tarmily Wen, and Jianbo Shi. Deep Image Blending. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 231–240, 2020.

[4] Chelsey Edge, Sadman Sakib Enan, Michael Fulton, Jungseok Hong, Jiawei Mo, Kimberly Barthelemy, Hunter Bashaw, Berik Kallevig, Corey Knutson, Kevin Orpen, and Junaed Sattar. Design and Experiments with LoCO AUV: A Low Cost Open-Source Autonomous Underwater Vehicle. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1761–1768, Las Vegas, Nevada, USA, October 2020.

[5] Fred Weinhaus. Fred's ImageMagick Scripts: UWCORRECT. http://www.fmwconcepts.com/imagemagick/uwcorrect/index.php, 2018. Accessed 10-28-2020.

[6] Gil Mor. iFish. https://github.com/Gil-Mor/iFish, 2020. Accessed 10-28-2020.

[7] Mars 2020 Perseverance Rover. https://mars.nasa.gov/mars2020/. Accessed 12-01-2022.

[8] Robotic Deep-sea vehicle lost on dive to 6-mile depth. https://www.whoi.edu/press-room/news-release/Nereus-Lost/, May 2014. Accessed 12-01-2022.

[9] Njoroge G. Kimani. Environmental Pollution and Impact to Public Health;Implication of the Dandora Municipal Dumping Site in Nairobi, Kenya. Technical report, The United Nations Environment Programme (UNEP), 2007.

[10] SB Sheavly and KM Register. Marine Debris & Plastics: Environmental Concerns, Sources, Impacts and Solutions. *Journal of Polymers and the Environment*, 15(4):301–305, 2007.

[11] Wai Chin LI, HF Tse, and Lincoln FOK. Plastic waste in the marine environment: A review of sources, occurrence and effects. *Science of the Total Environment*, 566:333–349, 2016.

[12] Beatrice Dupuy Star Tribune. *1,500 pounds of trash later, Mpls. lake cleanup continues.* http://www.startribune.com/minneapolis-lake-creates-concern-for-local-community/313495861/. Accessed 02-10-2018.

[13] Kara Lavender Law and Richard C Thompson. Microplastics in the seas. *Science*, 345(6193):144–145, 2014.

[14] Andrea Rigamonti Niels Philipsen. Marine Pollution - WP3 Quantitative Analysis. Technical report, The European Union Action to Fight Environmental Crime (EFFACE), 2015.

[15] Lidija Grozdanic inhabitat. *Floating Seabin trash collector could rid the oceans of plastic waste.* https://inhabitat.com/floating-seabin-sucks-up-ocean-waste-including-oil-and-detergents/. Accessed 04-11-2019.

[16] Carolyn Gramling Science News. *A massive net is being deployed to pick up plastic in the Pacific.* https://www.sciencenews.org/article/massive-net-being-deployed-pick-plastic-pacific. Accessed 04-11-2019.

[17] Fukushima Daiichi Nuclear Accident. https://www.iaea.org/topics/response/fukushima-daiichi-nuclear-accident. Accessed 12-01-2022.

[18] Robots come to the rescue after Fukushima Daiichi nuclear disaster. https://www.cbsnews.com/news/robots-fukushima-daiichi-nuclear-disaster-60-minutes-2021-07-11/. Accessed 11-01-2021.

[19] Matthew Price and Seren Jones BBC News. *Achieving the impossible: Thai cave rescue a year on*. https://www.bbc.com/news/world-asia-44791998. Accessed 04-14-2023.

[20] Richard C. Paddock Hannah Beech and Muktita Suhartono The New York Times. *'Still Can't Believe It Worked': The Story of the Thailand Cave Rescue*. https://www.nytimes.com/2018/07/12/world/asia/thailand-cave-rescue-seals.html. Accessed 04-14-2023.

[21] Liberty Zabala and Brie Stimson NBC 7 SAN DIEGO. *Robotics Team That Helped With Thai Cave Rescue Competing in San Diego*. https://www.nbcsandiego.com/news/local/robotics-team-that-helped-with-thai-cave-rescue-competing-in-san-diego/174125/. Accessed 04-14-2023.

[22] ONLINE REPORTERS and AGENCIES Bangkok Post. *Underwater robot, airborne drones aid cave search*. https://www.bangkokpost.com/thailand/general/1492678/underwater-robot-airborne-drones-aid-cave-search. Accessed 04-14-2023.

[23] Christopher Von Alt. Autonomous Underwater Vehicles. In *Autonomous Underwater Lagrangian Platforms and Sensors Workshop*, volume 3, page 2, 2003.

[24] Russell B Wynn, Veerle AI Huvenne, Timothy P Le Bas, Bramley J Murton, Douglas P Connelly, Brian J Bett, Henry A Ruhl, Kirsty J Morris, Jeffrey Peakall, Daniel R Parsons, et al. Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience. *Marine geology*, 352:451–468, 2014.

[25] Avilash Sahoo, Santosha K Dwivedy, and PS Robi. Advancements in the field of autonomous underwater vehicle. *Ocean Engineering*, 181:145–160, 2019.

[26] Ma Shiela Angeli C Marcos, Maricor N Soriano, and Caesar A Saloma. Classification of coral reef images from underwater video using neural networks. *Optics express*, 13(22):8766–8771, 2005.

[27] Md Moniruzzaman, Syed Mohammed Shamsul Islam, Mohammed Bennamoun, and Paul Lavery. Deep learning on underwater marine object detection: A survey. In *Advanced Concepts for Intelligent Vision Systems: 18th International Conference, ACIVS 2017, Antwerp, Belgium, September 18-21, 2017, Proceedings 18*, pages 150–160. Springer, 2017.

[28] Malte Schilling, Wolfram Burgard, Katharina Muelling, Britta Wrede, and Helge Ritter. Shared Autonomy – Learning of Joint Action and Human-Robot Collaboration, 2019.

[29] Michael Fulton, Jungseok Hong, Md Jahidul Islam, and Junaed Sattar. Robotic Detection of Marine Litter Using Deep Visual Detection Models . In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5752–5758. IEEE, 2019.

[30] Jungseok Hong, Michael Fulton, and Junaed Sattar. TrashCan: A Semantically-Segmented Dataset towards Visual Detection of Marine Debris. *arXiv preprint arXiv:2007.08097*, 2020.

[31] Jungseok Hong, Michael Fulton, and Junaed Sattar. A Generative Approach Towards Improved Robotic Detection of Marine Litter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10525–10531, 2020.

[32] Jiacheng Yuan, Jungseok Hong, Junaed Sattar, and Volkan Isler. ROW-SLAM: Under-Canopy Cornfield Semantic SLAM. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2244–2250, 2022.

[33] Jungseok Hong, Suveer Garg, and Volkan Isler. Semantic Mapping with Confidence Scores through Metric Embeddings and Gaussian Process Classification.

In *2023 IEEE International Conference on Robotics and Automation (ICRA). Accepted for publication.*, 2023.

[34] M. Bernstein, R. Graham, D. Cline, J. M. Dolan, and K. Rajan. Learning-Based Event Response for Marine Robotics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3362–3367, Nov 2013.

[35] Yogesh Girdhar, Anqi Xu, Bir Bikram Dey, Malika Meghjani, Florian Shkurti, Ioannis Rekleitis, and Gregory Dudek. MARE: Marine Autonomous Robotic Explorer. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '11)*, pages 5048–5053, San Francisco, USA, September 2011.

[36] Ayoung Kim and Ryan M Eustice. Active Visual SLAM for Robotic Area Coverage: Theory and Experiment. *The International Journal of Robotics Research*, 34(4-5):457–475, 2015.

[37] Xin Yuan, José-Fernán Martínez-Ortega, José Antonio Sánchez Fernández, and Martina Eckert. AEKF-SLAM: A New Algorithm for Robotic Underwater Navigation. *Sensors*, 17(5), 2017.

[38] John J Leonard and Alexander Bahr. Autonomous underwater vehicle navigation. In *Springer Handbook of Ocean Engineering*, pages 341–358. Springer, 2016.

[39] Luke Stutters, Honghai Liu, Carl Tiltman, and David J Brown. Navigation Technologies for Autonomous Underwater Vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(4):581–589, 2008.

[40] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. AUV Navigation and Localization: A Review. *IEEE Journal of Oceanic Engineering*, 39(1):131–149, 2014.

[41] Franco Hidalgo and Thomas Bräunl. Review of underwater SLAM techniques. In *Automation, Robotics and Applications (ICARA), 2015 6th International Conference on*, pages 306–311. IEEE, 2015.

[42] Thomas H. Mace. At-sea detection of marine debris: Overview of technologies, processes, issues, and options. *Marine Pollution Bulletin*, 65(1):23–27, Jan 2012.

[43] Evan A. Howell, Steven J. Bograd, Carey Morishige, Michael P. Seki, and Jeffrey J. Polovina. On North Pacific circulation and associated marine debris concentration. *Marine Pollution Bulletin*, 65(1):16–22, January 2012.

[44] National Oceanic and Atmospheric Administration. Detecting Japan Tsunami Marine Debris at Sea: A Synthesis of Efforts and Lessons-Learned | OR&R's Marine Debris Program, Jan 2015.

[45] Zhenpeng Ge, Huahong Shi, Xuefei Mei, Zhijun Dai, and Daoji Li. Semi-automatic recognition of marine debris on beaches. *Scientific Reports*, 6:25759, May 2016.

[46] S. Kulkarni and S. Junghare. Robot based indoor autonomous trash detection algorithm using ultrasonic sensors. In *2013 International Conference on Control, Automation, Robotics and Embedded Systems (CARE)*, pages 1–5, Dec 2013.

[47] M. Valdenegro-Toro. Submerged marine debris detection with autonomous underwater vehicles. In *2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA)*, pages 1–7, Dec 2016.

[48] Non-Profit Organization. *Clear Blue Sea*. `http://clearbluesea.org`. Accessed 02-10-2018.

[49] Rachael Z. Miller. *Underwater Robots Clean Up Marine Debris on the Seafloor*. The Rozalia Project. `https://www.sea-technology.com/features/2012/1212/underwater_robots.php`. Accessed 02-10-2018.

[50] Kyra Schlining, Susan von Thun, Linda Kuhnz, Brian Schlining, Lonny Lundsten, Nancy Jacobsen Stout, Lori Chaney, and Judith Connor. Debris in the deep: Using a 22-year video annotation database to survey marine litter in Monterey Canyon, central California, USA. *Deep Sea Research Part I: Oceanographic Research Papers*, 79:96 – 105, 2013.

[51] Japan Agency for Marine Earth Science and Technology. *Deep-sea Debris Database*, 2018. `http://www.godac.jamstec.go.jp/catalog/dsdebris/e/index.html`. Accessed 02-10-2018.

[52] Michael Fulton, Jungseok Hong, and Junaed Sattar. Trash-ICRA19: A Bounding Box Labeled Dataset of Underwater Trash. https://doi.org/10.13020/x0qn-y082. Retrieved from the Data Repository for the University of Minnesota.

[53] Jose GB Derraik. The Pollution of the Marine Environment by Plastic Debris. *Marine pollution bulletin*, 44(9):842–852, 2002.

[54] GitHub - tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images. https://github.com/tzutalin/labelImg. Accessed 02-25-2019.

[55] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[56] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[57] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[58] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society.

[59] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition . *arXiv preprint arXiv:1409.1556*, 2014.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision . In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[61] Tensorflow. Tensorflow Object Detection Zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`, 2017. Accessed 2-20-2018.

[62] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox Detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[63] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[64] AlexeyAB. YOLO: How to train (to detect your custom objects). `https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects)`, 2018. Accessed 9-14-2018.

[65] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[66] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[67] Dustin Franklin. *NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge.* `https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/`. Accessed 03-01-2018.

[68] Jiawei Mo and Junaed Sattar. Extending Monocular Visual Odometry to Stereo Camera Systems by Scale optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6921–6927, Nov 2019.

[69] 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.

[70] Michael Fulton, Jungseok Hong, Md Jahidul Islam, and Junaed Sattar. Robotic Detection of Marine Litter Using Deep Visual Detection Models. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5752–5758, May 2019.

[71] Supervisely. Supervisely - Web platform for computer vision. Annotation, training and deploy. `https://supervise.ly/`, 2020. Accessed 06-30-2020.

[72] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. Accessed 10-01-2019.

[73] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv preprint arXiv:1611.05431*, 2016.

[74] Zhengwei Wang, Qi She, and Tomas E Ward. Generative Adversarial Networks: A Survey and Taxonomy. *arXiv preprint arXiv:1906.01529*, 2019.

[75] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data Augmentation Generative Adversarial Networks. *arXiv preprint arXiv:1711.04340*, 2017.

[76] 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. *arXiv preprint arXiv:1702.07392*, 2017.

[77] Cameron Fabbri, Md Jahidul Islam, and Junaed Sattar. Enhancing Underwater Imagery using Generative Adversarial Networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7159–7165. IEEE, 2018.

[78] Yang-Jie Cao, Li-Li Jia, Yong-Xia Chen, Nan Lin, Cong Yang, Bo Zhang, Zhi Liu, Xue-Xiang Li, and Hong-Hua Dai. Recent Advances of Generative Adversarial Networks in Computer Vision. *IEEE Access*, 7:14985–15006, 2018.

[79] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv preprint arXiv:1703.10593*, 2017.

[80] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[81] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*, 2014.

[82] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[83] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[84] Bin Dai and David Wipf. Diagnosing and Enhancing VAE Models. *arXiv preprint arXiv:1903.05789*, 2019.

[85] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural Photo Editing with Introspective Adversarial Networks. *arXiv preprint arXiv:1609.07093*, 2016.

[86] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein Auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

[87] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow . In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[88] 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.

[89] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a

Local Nash Equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[90] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[91] Cyril Goutte and Eric Gaussier. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[92] K. Anantharajah, ZongYuan Ge, C. McCool, S. Denman, C. Fookes, P. Corke, D. Tjondronegoro, and S. Sridharan. Local Inter-Session Variability Modelling for Object Classification. In *IEEE Winter Conference on Applications of Computer Vision*, pages 309–316, March 2014.

[93] Tianfeng Chai and Roland R Draxler. Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[94] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *nature*, 521(7553):436–444, 2015.

[95] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, page 103514, 2022.

[96] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. *International journal of computer vision*, 115:211–252, 2015.

[97] Akhil Singh, Vaibhav Jaiswal, Gaurav Joshi, Adith Sanjeeve, Shilpa Gite, and Ketan Kotecha. Neural Style Transfer: A Critical Review. *IEEE Access*, 9:131583–131613, 2021.

[98] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural Style Transfer: A Review. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3365–3385, 2020.

[99] Adrian Lopez Rodriguez and Krystian Mikolajczyk. Domain Adaptation for Object Detection via Style Consistency. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, page 232. BMVA Press, 2019.

[100] Fuxun Yu, Di Wang, Yinpeng Chen, Nikolaos Karianakis, Tong Shen, Pei Yu, Dimitrios Lymberopoulos, Sidi Lu, Weisong Shi, and Xiang Chen. SC-UDA: Style and Content Gaps aware Unsupervised Domain Adaptation for Object Detection. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1061–1070, 2022.

[101] David Kadish, Sebastian Risi, and Anders Sundnes Løvlie. Improving Object Detection in Art Images Using Only Style Transfer. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[102] Amir Ebrahimi Abdollah Amirkhani, Amir Hossein Barshooi. Enhancing the Robustness of Visual Object Tracking via Style Transfer. *Computers, Materials & Continua*, 70(1):981–997, 2022.

[103] Che-Tsung Lin, Sheng-Wei Huang, Yen-Yi Wu, and Shang-Hong Lai. GAN-Based Day-to-Night Image Style Transfer for Nighttime Vehicle Detection. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):951–963, 2021.

[104] Tong Liu, Zhaowei Chen, Yi Yang, Zehao Wu, and Haowei Li. Lane Detection in Low-light Conditions Using an Efficient Data Enhancement: Light Conditions Style Transfer. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1394–1399, 2020.

[105] S. Cygert and A. Czyzewski. Style Transfer for Detecting Vehicles with Thermal Camera. In *2019 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 218–222, 2019.

[106] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. GP-GAN: Towards Realistic High-Resolution Image Blending, 2017, 1703.07195.

[107] Georgios Georgakis, Arsalan Mousavian, Alexander C. Berg, and Jana Kosecka. Synthesizing Training Data for Object Detection in Indoor Scenes, 2017, 1702.07836.

[108] Ning Zhang, Francesco Nex, George Vosselman, and Norman Kerle. Training a Disaster Victim Detection Network for UAV Search and Rescue Using Harmonious Composite Images. *Remote Sensing*, 14(13), 2022.

[109] Renting Liu, Zhaorong Li, and Jiaya Jia. Image Partial Blur Detection and Classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[110] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[111] Aravindh Mahendran and Andrea Vedaldi. Understanding Deep Image Representations by Inverting Them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015.

[112] Shuying Liu and Weihong Deng. Very Deep Convolutional Neural Network Based Image Classification Using Small Training Sample Size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[113] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time Instance Segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019.

[114] Michael Fulton, Aditya Prabhu, and Junaed Sattar. HREyes: Design, Development, and Evaluation of a Novel Method for AUVs to Communicate Information and Gaze Direction. *Accepted for publication at the IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[115] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully Convolutional Instance-aware Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2359–2367, 2017.

[116] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[117] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path Aggregation Network for Instance Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[118] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask Scoring R-CNN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6409–6418, 2019.

[119] Sharmin Rahman, Alberto Quattrini Li, and Ioannis Rekleitis. SVIn2: An Underwater SLAM System using Sonar, Visual, Inertial, and Depth Sensor. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1861–1868. IEEE, 2019.

[120] 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.

[121] 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. In *Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[122] Michael Hoy, Alexey S Matveev, and Andrey V Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, 33(3):463–497, 2015.

[123] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding Horizon "Next-Best-View" Planner for 3D Exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468. IEEE, 2016.

[124] Michael Hoy, Alexey S Matveev, and Andrey V Savkin. Collision free cooperative navigation of multiple wheeled robots in unknown cluttered environments. *Robotics and Autonomous Systems*, 60(10):1253–1266, 2012.

[125] Oussama Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Autonomous Robot Vehicles*, pages 396–404. Springer, 1986.

[126] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[127] Petter Ögren and Naomi E Leonard. A Provably Convergent Dynamic Window Approach to Obstacle Avoidance. *IFAC Proceedings Volumes*, 35(1):115–120, 2002.

[128] Ellips Masehian and MR Amin-Naseri. A Voronoi Diagram–Visibility Graph–Potential Field Compound Algorithm for Robot Path Planning. *Journal of Robotic Systems*, 21(6):275–300, 2004.

[129] Amalia F Foka and Panos E Trahanias. Probabilistic Autonomous Robot Navigation in Dynamic Environments with Human Motion Prediction. *International Journal of Social Robotics*, 2(1):79–94, 2010.

[130] Emrah Akin Sisbot, Luis F Marin-Urias, Rachid Alami, and Thierry Simeon. A Human Aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.

[131] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J Andrew Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based Prediction for Pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3931–3936. IEEE, 2009.

[132] Minnesota Interactive Robotics and Vision Laboratory. Segmentation of Underwater Imagery Dataset. `http://irvlab.cs.umn.edu/resources`. Accessed 10-31-2020.

[133] Vladimir J Lumelsky and Alexander A Stepanov. Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape. *Algorithmica*, 2(1):403–430, 1987.

[134] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[135] Gregory Dudek, Philippe Giguere, Chris Prahacs, Shane Saunderson, Junaed Sattar, Luz-Abril Torres-Mendez, Michael Jenkin, Andrew German, Andrew Hogue, Arlene Ripsman, et al. AQUA: An Amphibious Autonomous Robot. *Computer*, 40(1), 2007.

[136] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[137] Tamaki Ura, Takeshi Nakatani, and Yoshiaki Nose. Terrain Based Localization Method for Wreck Observation AUV. In *OCEANS 2006*, pages 1–6. IEEE, 2006.

[138] Nathaniel Fairfield and David Wettergreen. Active localization on the ocean floor with multibeam sonar. In *OCEANS 2008*, pages 1–10. IEEE, 2008.

[139] Takeshi Nakatani, Tamaki Ura, Takashi Sakamaki, and Junichi Kojima. Terrain based localization for pinpoint observation of deep seafloors. In *OCEANS 2009-EUROPE*, pages 1–6. IEEE, 2009.

[140] Francisco Curado Teixeira, João Quintas, Pramod Maurya, and António Pascoal. Robust particle filter formulations with application to terrain-aided navigation. *International Journal of Adaptive Control and Signal Processing*, 31(4):608–651, 2017.

[141] José Melo and Aníbal Matos. Survey on advances on terrain based navigation for autonomous underwater vehicles. *Ocean Engineering*, 139:250–264, 2017.

[142] Stefan Williams and Ian Mahon. A Terrain-Aided Tracking Algorithm for Marine Systems. In *Field and Service Robotics*, pages 93–102. Springer, 2003.

[143] Deborah K Meduna, Stephen M Rock, and Rob McEwen. Low-cost terrain relative navigation for long-range AUVs. In *OCEANS 2008*, pages 1–7. IEEE, 2008.

[144] Taeyun Kim and Jinwhan Kim. Nonlinear filtering for terrain-referenced underwater navigation with an acoustic altimeter. In *OCEANS 2014-TAIPEI*, pages 1–6. IEEE, 2014.

[145] Sebastian Carreno, Philip Wilson, Pere Ridao, and Yvan Petillot. A survey on Terrain Based Navigation for AUVs. In *OCEANS 2010*, pages 1–7. IEEE, 2010.

[146] Bo He, Ke Yang, Shuai Zhao, and Yitong Wang. Underwater simultaneous localization and mapping based on EKF and point features. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 4845–4850. IEEE, 2009.

[147] Ba-Da Yoon, Ha-Nul Yoon, Sung-He Choi, and Jang-Myung Lee. Position Estimation of Underwater-Beacon Precision. In *Intelligent Autonomous Systems 12*, pages 501–508. Springer, 2013.

[148] M Karimi, M Bozorg, and AR Khayatian. A comparison of DVL/INS fusion by UKF and EKF to localize an autonomous underwater vehicle. In *Robotics and Mechatronics (ICRoM), 2013 First RSI/ISM International Conference on*, pages 62–67. IEEE, 2013.

[149] Sebastian Thrun. Particle Filters in Robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[150] Rickard Karlsson, Fredrik Gusfafsson, and Tobias Karlsson. Particle filtering and Cramer-Rao lower bound for underwater navigation. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 6, pages VI–65. IEEE, 2003.

[151] Ioannis Rekleitis. A Particle Filter Tutorial for Mobile Robot Localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, Jan. 2004.

[152] David Salmond and Neil Gordon. An introduction to particle filters. *State space and unobserved component models theory and applications*, pages 1–19, 2005.

[153] Francesco Maurelli, Angelos Mallios, David Ribas, Pere Ridao, and Yvan Petillot. Particle Filter Based AUV Localization using Imaging Sonar. *IFAC Proceedings Volumes*, 42(18):52–57, 2009.

[154] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, 2010.

[155] Thomas B Schön, Fredrik Gustafsson, and Rickard Karlsson. The Particle Filter in Practice. In Boris Rozovskii Dan Crisan, editor, *The Oxford Handbook of Nonlinear Filtering*, pages 741–767. Oxford University Press, 2011.

[156] Thomas Schon, Fredrik Gustafsson, and P-J Nordlund. Marginalized Particle Filters for Mixed Linear/Nonlinear State-Space Models. *IEEE Transactions on Signal Processing*, 53(7):2279–2289, 2005.

[157] Rickard Karlsson and Fredrik Gustafsson. Bayesian Surface and Underwater Navigation. *IEEE Transactions on Signal Processing*, 54(11):4204–4213, 2006.

[158] Eric W Weisstein. Euler angles. *https://mathworld.wolfram.com/*, 2009.

[159] Thomas B Schon, Rickard Karlsson, and Fredrik Gustafsson. The Marginalized Particle Filter in Practice. In *Aerospace Conference, 2006 IEEE*, pages 11–pp. IEEE, 2006.

[160] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, 2005.

[161] Fred Daum. Nonlinear filters: beyond the Kalman filter. *IEEE Aerospace and Electronic Systems Magazine*, 20(8):57–69, 2005.

[162] Eric A Wan and Rudolph Van Der Merwe. The Unscented Kalman Filter for Nonlinear Estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[163] Thomas B Schön, Rickard Karlsson, and Fredrik Gustafsson. The Marginalized Particle Filter–Analysis, Applications and Generalizations. In *ESAIM: Proceedings*, volume 19, pages 53–64. EDP Sciences, 2007.

[164] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An Open-Source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[165] Natural Resources Department. Lake Bathymetric Outlines, Contours, Vegetation, and DEM. `https://gisdata.mn.gov/dataset/water-lake-bathymetry`. Accessed 06-01-2018.

[166] Nathan Koenig and Andrew Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[167] Chelsey Edge, Sadman Sakib Enan, Michael Fulton, Jungseok Hong, Jiawei Mo, Kimberly Barthelemy, Hunter Bashaw, Berik Kallevig, Corey Knutson, Kevin Orpen, and Junaed Sattar. Design and Experiments with LoCO AUV: A Low Cost Open-Source Autonomous Underwater Vehicle*. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1761–1768, 2020.

[168] Kristopher Krasnosky, Christopher Roman, and David Casagrande. A bathymetric mapping and SLAM dataset with high-precision ground truth for marine robotics. *The International Journal of Robotics Research*, 41(1):12–19, 2022, https://doi.org/10.1177/02783649211044749.

[169] J. G. Bellingham and J. S. Willcox. Optimizing AUV oceanographic surveys. In *Proceedings of Symposium on Autonomous Underwater Vehicle Technology*, pages 391–398, 1996.

[170] Jimin Hwang, Neil Bose, and Shuangshuang Fan. AUV Adaptive Sampling Methods: A Review. *Applied Sciences*, 9(15), 2019.

[171] G. Dudek, P. Giguere, C. Prahacs, S. Saunderson, J. Sattar, L. Torres-Mendez, M. Jenkin, A. German, A. Hogue, A. Ripsman, J. Zacher, E. Milios, H. Liu, P. Zhang, M. Buehler, and C. Georgiades. AQUA: An Amphibious Autonomous Robot. *Computer*, 40(1):46–53, 2007.

[172] N. Miskovic, A. Pascoal, M. Bibuli, M. Caccia, J. A. Neasham, A. Birk, M. Egi, K. Grammer, A. Marroni, A. Vasilijevic, D. Nad, and Z. Vukic. CADDY project, year 3: The final validation+ trials. In *OCEANS 2017 - Aberdeen*, pages 1–5, 2017.

[173] M. J. Islam, M. Ho, and J. Sattar. Dynamic Reconfiguration of Mission Parameters in Underwater Human-Robot Collaboration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2018.

[174] Arturo Gomez Chavez, Christian A. Mueller, Tobias Doernbach, Davide Chiarella, and Andreas Birk. Robust Gesture-Based Communication for Underwater Human-Robot Interaction in the context of Search and Rescue Diver Missions. *arXiv:1810.07122 [cs]*, October 2018.

[175] Md Jahidul Islam, Marc Ho, and Junaed Sattar. Understanding human motion and gestures for underwater human–robot collaboration. *Journal of Field Robotics*, 36(5):851–873, 2019, https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21837.

[176] K. J. DeMarco, M. E. West, and A. M. Howard. Underwater human-robot communication: A case study with human divers. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3738–3743, October 2014.

[177] M. Fulton, C. Edge, and J. Sattar. Robot Communication Via Motion: Closing the Underwater Human-Robot Interaction Loop. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4660–4666, May 2019.

[178] J. Sattar and G. Dudek. Where is your dive buddy: tracking humans underwater using spatio-temporal features. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3654–3659, 2007.

[179] Md Jahidul Islam, Michael Fulton, and Junaed Sattar. Toward a Generic Diver-Following Algorithm: Balancing Robustness and Efficiency in Deep Visual Detection. *IEEE Robotics and Automation Letters*, 4(1):113–120, 2018.

[180] Michael Fulton, Chelsey Edge, and Junaed Sattar. Robot Communication Via Motion: A Study on Modalities for Robot-to-Human Communication in the Field. *ACM Transactions on Human-Robot Interaction (THRI)*, 11(2):1–40, 2022.

[181] K. Dautenhahn, M. Walters, S. Woods, K. L. Koay, C. L. Nehaniv, A. Sisbot, R. Alami, and T. Siméon. How may i serve you? a robot companion approaching a seated person in a helping context. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, HRI '06, pages 172–179. Association for Computing Machinery, 2006.

[182] Dag Sverre Syrdal, Kerstin Dautenhahn, Sarah Woods, Michael L. Walters, and Kheng Lee Koay. 'doing the right thing wrong' - personality and tolerance to uncomfortable robot approaches. In *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 183–188, 2006.

[183] M. L. Walters, D. S. Syrdal, K. L. Koay, K. Dautenhahn, and R. te Boekhorst. Human approach distances to a mechanical-looking robot with different robot voice styles. In *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, pages 707–712, 2008. ISSN: 1944-9437.

[184] L. Takayama and C. Pantofaru. Influences on proxemic behaviors in human-robot interaction. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5495–5502, 2009.

[185] Christopher Brandl, Alexander Mertens, and Christopher M. Schlick. Human-robot interaction in assisted personal services: Factors influencing distances that humans will accept between themselves and an approaching service robot. *Human Factors and Ergonomics in Manufacturing & Service Industries*, 26(6):713–727, 2016, https://onlinelibrary.wiley.com/doi/pdf/10.1002/hfm.20675.

[186] Walid Remmas, Ahmed Chemori, and Maarja Kruusmaa. Diver tracking in open waters: A low-cost approach based on visual and acoustic sensor fusion. *Journal of Field Robotics*, 38(3):494–508, 2021, https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21999.

[187] Dula Nad, Filip Mandic, and Nikola Miskovic. Using autonomous underwater vehicles for diver tracking and navigation aiding. *Journal of Marine Science and Engineering*, 8(6), 2020.

[188] Norman S. Nise. *Control Systems Engineering.* Wiley, 7 edition, 2015.

[189] Karin de Langis, Michael Fulton, and Junaed Sattar. Towards Robust Visual Diver Detection Onboard Autonomous Underwater Robots: Assessing the Effects of Models and Data. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5372–5378, 2021.

[190] NVIDIA AI IOT. trt_pose. `https://github.com/NVIDIA-AI-IOT/trt_pose`, 2019. Accessed 03-01-2020.

[191] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[192] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.

[193] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.

[194] David Gallup, Jan-Michael Frahm, Philippos Mordohai, and Marc Pollefeys. Variable baseline/resolution stereo. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[195] Margaret A McDowell, Cheryl D Fryar, and Cynthia L Ogden. Anthropometric reference data for children and adults, United States, 1988-1994; data from the third National Health and Nutrition Examination Survey (NHANES III). *Vital Health Stat 11*, 2009.

[196] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, Sep. 2004.

[197] Alphonse Bertillon. Signaletic Instructions, including the Theory and Practice of Anthropometrical Identification. *Nature*, pages 569–570, October 1896.

[198] Pawel Drozdowski, Christian Rathgeb, and Christoph Busch. Computational Workload in Biometric Identification Systems: An Overview. *IET Biometrics*, 8(6):351–368, 2019.

[199] Theerawit Wilaiprasitporn, Apiwat Ditthapron, Karis Matchaparn, Tanaboon Tongbuasirilai, Nannapas Banluesombatkul, and Ekapol Chuangsuwanich. Affective EEG-Based Person Identification Using the Deep Learning Approach. *IEEE Transactions on Cognitive and Developmental Systems*, 12(3):486–496, 2020.

[200] Yassin Kortli, Maher Jridi, Ayman Al Falou, and Mohamed Atri. Face Recognition Systems: A Survey. *Sensors*, 20(2):342, 2020.

[201] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019.

[202] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. WIDER FACE: A Face Detection Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[203] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition. In *European Conference on Computer Vision*, pages 87–102. Springer, 2016.

[204] Gwangbin Bae, Martin de La Gorce, Tadas Baltrušaitis, Charlie Hewitt, Dong Chen, Julien Valentin, Roberto Cipolla, and Jingjing Shen. DigiFace-1M: 1 Million Digital Face Images for Face Recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3526–3535, 2023.

[205] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[206] Muhammad Hassan Khan, Muhammad Shahid Farid, and Marcin Grzegorzek. Vision-based Approaches Towards Person Identification using Gait. *Computer Science Review*, 42:100432, 2021.

[207] Youya Xia and Junaed Sattar. Visual Diver Recognition for Underwater Human-Robot Collaboration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6839–6845. IEEE, 2019.

[208] Karin de Langis and Junaed Sattar. Realtime Multi-Diver Tracking and Re-identification for Underwater Human-Robot Collaboration. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11140–11146. IEEE, 2020.

[209] Yujiang Wang, Jie Shen, Stavros Petridis, and Maja Pantic. A real-time and unsupervised face Re-Identification system for Human-Robot Interaction. *Pattern Recognition Letters*, 128:559–568, 2019.

[210] Y. L. Lee, M. Y. Tseng, Y. C. Luo, D. R. Yu, and W. C. Chiu. Learning Face Recognition Unsupervisedly by Disentanglement and Self-Augmentation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3018–3024, 2020.

[211] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[212] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Representation by Joint Identification-Verification. In *Advances in Neural Information Processing Systems*, pages 1988–1996. Curran Associates, Inc., 2014.

[213] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[214] Xu Tang, Daniel K. Du, Zeqiang He, and Jingtuo Liu. PyramidBox: A Context-assisted Single Shot Face Detector. In *European Conference on Computer Vision (ECCV)*, 2018.

[215] Xudong Sun, Pengcheng Wu, and Steven CH Hoi. Face detection using deep learning: An improved faster RCNN approach. *Neurocomputing*, 299:42–50, 2018.

[216] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li. S^3FD: Single Shot Scale-Invariant Face Detector. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 192–201, 2017.

[217] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep Face Recognition. In *British Machine Vision Conference (BMVC)*, pages 41.1–41.12. BMVA Press, 2015.

[218] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A Discriminative Feature Learning Approach for Deep Face Recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016.

[219] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. CosFace: Large Margin Cosine Loss for Deep Face Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018.

[220] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. SphereFace: Deep Hypersphere Embedding for Face Recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6738–6746, 2017.

[221] L. Song, D. Gong, Z. Li, C. Liu, and W. Liu. Occlusion Robust Face Recognition Based on Mask Learning With Pairwise Differential Siamese Network. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 773–782, 2019.

[222] Ali Elmahmudi and Hassan Ugail. Deep face recognition using imperfect facial data. *Future Generation Computer Systems*, 99:213–225, 2019.

[223] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust Face Recognition via Sparse Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009.

[224] A. Singh, D. Patil, G. M. Reddy, and S. N. Omkar. Disguised Face Identification (DFI) with Facial KeyPoints Using Spatial Fusion Convolutional Network. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1648–1655, 2017.

[225] Lingxiao He, Haiqing Li, Qi Zhang, and Zhenan Sun. Dynamic Feature Learning for Partial Face Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7054–7063, 2018.

[226] Brent C Munsell, Andrew Temlyakov, Chengzheng Qu, and Song Wang. Person Identification Using Full-Body Motion and Anthropometric Biometrics From Kinect Videos. In *Computer Vision–ECCV 2012. Workshops and Demonstrations: Florence, Italy, October 7-13, 2012, Proceedings, Part III 12*, pages 91–100. Springer, 2012.

[227] Virginia Andersson and Ricardo Araujo. Person Identification Using Anthropometric and Gait Data from Kinect Sensor. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015.

[228] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[229] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. MediaPipe: A Framework for Building Perception Pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

[230] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. DeepLabCut: Markerless Pose Estimation of User-defined Body Parts with Deep Learning. *Nature Neuroscience*, 21(9):1281–1289, 2018.

[231] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep High-Resolution Representation Learning for Visual Recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

[232] Tal Hassner, Shai Harel, Eran Paz, and Roee Enbar. Effective Face Frontalization in Unconstrained Images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[233] Ellen Schwalbe. GEOMETRIC MODELLING AND CALIBRATION OF FISH-EYE LENS CAMERA SYSTEMS. In *Panoramic Photogrammetry Workshop*, volume 36, 2005.

[234] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep Convolutional Autoencoder-based Lossy Image Compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257. IEEE, 2018.

[235] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou. RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5202–5211, 2020.

[236] J. Li, Y. Wang, C. Wang, Y. Tai, J. Qian, J. Yang, C. Wang, J. Li, and F. Huang. DSFD: Dual Shot Face Detector. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5055–5064, 2019.

[237] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[238] S. Sengupta, J. Chen, C. Castillo, V. M. Patel, R. Chellappa, and D. W. Jacobs. Frontal to Profile Face Verification in the Wild. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.

[239] Libor Spacek. Facial Images: Faces94. `http://cmp.felk.cvut.cz/~spacelib/faces/faces94.html`, 2009. Accessed 10-28-2020.

[240] Hieu V Nguyen and Li Bai. Cosine Similarity Metric Learning for Face Verification. In *Asian Conference on Computer Vision*, pages 709–720. Springer, 2010.

[241] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. VGGFace2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 67–74. IEEE, 2018.

[242] Taesung Park, Alexei A Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive Learning for Unpaired Image-to-Image Translation. *arXiv preprint arXiv:2007.15651*, 2020.

[243] Claire C Gordon, Thomas Churchill, Charles E Clauser, Bruce Bradtmiller, John T McConville, Ilse Tebbetts, and Robert A Walker. Anthropometric Survey of US Army Personnel: Summary Statistics, Interim Report for 1988. Technical report, Anthropology Research Project Inc Yellow Springs OH, 1989.

[244] Liu Yang and Rong Jin. Distance Metric Learning: A Comprehensive Survey. *Michigan State University*, 2(2):4, 2006.

[245] Leif E Peterson. K-Nearest Neighbor. *Scholarpedia*, 4(2):1883, 2009.

[246] William S Noble. What Is a Support Vector Machine? *Nature Biotechnology*, 24(12):1565–1567, 2006.

[247] Laurens Van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[248] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: An Open-Source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[249] Michael Fulton, Jungseok Hong, and Junaed Sattar. Using Monocular Vision and Human Body Priors for AUVs to Autonomously Approach Divers. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1076–1082, 2022.

[250] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[251] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[252] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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. In *Advances in Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc., 2019.

[253] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The Replica Dataset: A Digital Replica of Indoor Spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[254] Sharmin Rahman, Alberto Quattrini Li, and Ioannis Rekleitis. Sonar Visual Inertial SLAM of Underwater Structures. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5190–5196. IEEE, 2018.

[255] Sharmin Rahman, Nare Karapetyan, Alberto Quattrini Li, and Ioannis Rekleitis. A Modular Sensor Suite for Underwater Reconstruction. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–6. IEEE, 2018.

[256] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary 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. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[257] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, et al. TensorFlow: A System for Large-scale Machine Learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[258] Simon J Julier and Jeffrey K Uhlmann. New extension of the Kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[259] Charles River Analytics, Inc. robot_localization. `https://github.com/cra-ros-pkg/robot_localization`. Accessed 11-01-2021.

[260] Piyush Pandey, Hemanth Narayan Dakshinamurthy, and Sierra Young. A Literature Review of Non-Herbicide, Robotic Weeding: A Decade of Progress. 2020.

[261] Philipp Lottes, Raghav Khanna, Johannes Pfeifer, Roland Siegwart, and Cyrill Stachniss. UAV-based Crop and Weed Classification for Smart Farming. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3024–3031, 2017.

[262] F Castaldi, F Pelosi, S Pascucci, and R Casa. Assessing the Potential of Images from Unmanned Aerial Vehicles (UAV) to Support Herbicide Patch Spraying in Maize. *Precision Agriculture*, 18(1):76–94, 2017.

[263] Xiaolong Wu, Stéphanie Aravecchia, Philipp Lottes, Cyrill Stachniss, and Cédric Pradalier. Robotic Weed Control Using Automated Weed and Crop Classification. *Journal of Field Robotics*, 37(2):322–340, 2020.

[264] Arun Narenthiran Sivakumar, Sahil Modi, Mateus Valverde Gasparino, Che Ellis, Andres Eduardo Baquero Velasquez, Girish Chowdhary, and Saurabh Gupta. Learned Visual Navigation for Under-Canopy Agricultural Robots. *arXiv preprint arXiv:2107.02792*, 2021.

[265] Zhongzhong Zhang, Erkan Kayacan, Benjamin Thompson, and Girish Chowdhary. High Precision Control and Deep Learning-based Corn Stand Counting Algorithms for Agricultural Robot. *Autonomous Robots*, 44(7):1289–1302, 2020.

[266] Pierre Moulon, Pascal Monasse, and Renaud Marlet. Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3248–3255, 2013.

[267] Seyyed Meghdad Hasheminasab, Tian Zhou, and Ayman Habib. GNSS/INS-Assisted Structure from Motion Strategies for UAV-Based Imagery over Mechanized Agricultural Fields. *Remote Sensing*, 12(3):351, 2020.

[268] Błażej Czupryński and Adam Strupczewski. Real-time RGBD SLAM System. In *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2015*, volume 9662, page 96622B. International Society for Optics and Photonics, 2015.

[269] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[270] Mathieu Labbé and François Michaud. RTAB-Map as an Open-source Lidar and Visual Simultaneous Localization and Mapping Library for Large-scale and Long-term Online Operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[271] Fangwei Zhong, Sheng Wang, Ziqi Zhang, and Yizhou Wang. Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1001–1010. IEEE, 2018.

[272] Zemin Wang, Qian Zhang, Jiansheng Li, Shuming Zhang, and Jingbin Liu. A Computationally Efficient Semantic SLAM Solution for Dynamic Scenes. *Remote Sensing*, 11(11):1363, 2019.

[273] Cosimo Rubino, Marco Crocco, and Alessio Del Bue. 3d Object Localisation from Multi-view Image Detections. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1281–1294, 2017.

[274] Lachlan Nicholson, Michael Milford, and Niko Sünderhauf. QuadricSLAM: Dual Quadrics from Object Detections as Landmarks in Object-oriented SLAM. *IEEE Robotics and Automation Letters*, 4(1):1–8, 2018.

[275] Kyel Ok, Katherine Liu, Kris Frey, Jonathan P How, and Nicholas Roy. Robust Object-based SLAM for High-speed Autonomous Navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 669–675. IEEE, 2019.

[276] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey, 2019, 1905.05055.

[277] Andreas Kamilaris and Francesc X. Prenafeta-Boldú. Deep Learning in Agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70–90, 2018.

[278] ASM Mahmudul Hasan, Ferdous Sohel, Dean Diepeveen, Hamid Laga, and Michael GK Jones. A Survey of Deep Learning Techniques for Weed Detection from Images. *Computers and Electronics in Agriculture*, 184:106067, 2021.

[279] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep Learning in Video Multi-object Tracking: A Survey. *Neurocomputing*, 381:61–88, 2020.

[280] Antonio Brunetti, Domenico Buongiorno, Gianpaolo Francesco Trotta, and Vitoantonio Bevilacqua. Computer Vision and Deep Learning Techniques for Pedestrian Detection and Tracking: A Survey. *Neurocomputing*, 300:17–33, 2018.

[281] Aljoša Osep, Wolfgang Mehner, Markus Mathias, and Bastian Leibe. Combined Image and World-space Tracking in Traffic Scenes. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1988–1995. IEEE, 2017.

[282] Lewis Bridgeman, Marco Volino, Jean-Yves Guillemaut, and Adrian Hilton. Multi-person 3d Pose Estimation and Tracking in Sports. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[283] Xu Liu, Steven W Chen, Shreyas Aditya, Nivedha Sivakumar, Sandeep Dcunha, Chao Qu, Camillo J Taylor, Jnaneshwar Das, and Vijay Kumar. Robust Fruit Counting: Combining Deep Learning, Tracking, and Structure from Motion. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1045–1052. IEEE, 2018.

[284] Pravakar Roy and Volkan Isler. Surveying Apple Orchards with a Monocular Vision System. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 916–921, 2016.

[285] Berthold KP Horn and Brian G Schunck. Determining Optical Flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[286] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple Online and Realtime Tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[287] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a Benchmark for Multi-target Tracking. *arXiv preprint arXiv:1504.01942*, 2015.

[288] 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*, 28:91–99, 2015.

[289] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobilenetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[290] R Revathi and M Hemalatha. Certain Approach of Object Tracking using Optical Flow Techniques. *International Journal of Computer Applications*, 53(8), 2012.

[291] Jean-Yves Bouguet et al. Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker Description of the Algorithm. *Intel corporation*, 5(1-10):4, 2001.

[292] Greg Welch, Gary Bishop, et al. An Introduction to the Kalman Filter. 1995.

[293] H. W. Kuhn and Bryn Yaw. The Hungarian Method for the Assignment Problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[294] Martin A Fischler and Robert C Bolles. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[295] John L Crassidis. Sigma-point Kalman Filtering for Integrated GPS and Inertial Navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 42(2):750–756, 2006.

[296] Sourav Garg, Niko Sünderhauf, Feras Dayoub, Douglas Morrison, Akansel Cosgun, Gustavo Carneiro, Qi Wu, Tat-Jun Chin, Ian Reid, Stephen Gould, et al. Semantics for robotic mapping, perception and interaction: A survey. *Foundations and Trends® in Robotics*, 8(1–2):1–224, 2020.

[297] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. In *Sensor devices and systems for robotics*, pages 253–276. Springer, 1989.

[298] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[299] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[300] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[301] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. Ieee, 2011.

[302] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The international journal of Robotics Research*, 31(5):647–663, 2012.

[303] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[304] Cipriano Galindo, Alessandro Saffiotti, Silvia Coradeschi, Pär Buschka, Juan-Antonio Fernandez-Madrigal, and Javier González. Multi-hierarchical semantic maps for mobile robotics. In *2005 IEEE/RSJ international conference on intelligent robots and systems*, pages 2278–2283. IEEE, 2005.

[305] Hendrik Zender, O Martínez Mozos, Patric Jensfelt, G-JM Kruijff, and Wolfram Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008.

[306] Zhixuan Wei, Weidong Chen, and Jingchuan Wang. 3d semantic map-based shared control for smart wheelchair. In *International Conference on Intelligent Robotics and Applications*, pages 41–51. Springer, 2012.

[307] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In

2017 IEEE International Conference on Robotics and automation (ICRA), pages 4628–4635. IEEE, 2017.

[308] Niko Sünderhauf, Trung T Pham, Yasir Latif, Michael Milford, and Ian Reid. Meaningful maps with object-oriented semantic mapping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5079–5085. IEEE, 2017.

[309] Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D Tardós, and Jose Maria Martinez Montiel. Towards semantic slam using a monocular camera. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 1277–1284. IEEE, 2011.

[310] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013.

[311] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1689–1696. IEEE, 2020.

[312] Fahad Lateef and Yassine Ruichek. Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 338:321–348, 2019.

[313] Simon O'Callaghan, Fabio T Ramos, and Hugh Durrant-Whyte. Contextual occupancy maps using gaussian processes. In *2009 IEEE International Conference on Robotics and Automation*, pages 1054–1060. IEEE, 2009.

[314] Simon T O'Callaghan and Fabio T Ramos. Gaussian process occupancy maps. *The International Journal of Robotics Research*, 31(1):42–62, 2012.

[315] Jinkun Wang and Brendan Englot. Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1003–1010. IEEE, 2016.

[316] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[317] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[318] Carlos Villacampa-Calvo, Bryan Zaldivar, Eduardo C Garrido-Merchán, and Daniel Hernández-Lobato. Multi-class gaussian process classification with noisy inputs. *The Journal of Machine Learning Research*, 22(1):1696–1747, 2021.

[319] Maani Ghaffari Jadidi, Lu Gan, Steven A. Parkison, Jie Li, and Ryan M. Eustice. Gaussian processes semantic map representation. *ArXiv*, abs/1707.01532, 2017.

[320] Ehsan Zobeidi, Alec Koppel, and Nikolay Atanasov. Dense incremental metric-semantic mapping via sparse gaussian process regression. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6180–6187. IEEE, 2020.

[321] Eric Guerrero-Font, Francisco Bonin-Font, Miguel Martin-Abadal, Yolanda Gonzalez-Cid, and Gabriel Oliver-Codina. Sparse gaussian process for online seagrass semantic mapping. *Expert Systems with Applications*, 170:114478, 2021.

[322] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[323] Kevin Doherty, Dehann Fourie, and John Leonard. Multimodal semantic slam with probabilistic data association. In *2019 international conference on robotics and automation (ICRA)*, pages 2419–2425. IEEE, 2019.

[324] Kevin J Doherty, David P Baxter, Edward Schneeweiss, and John J Leonard. Probabilistic data association via mixture models for robust semantic slam. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1098–1104. IEEE, 2020.

[325] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[326] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017, 1703.06870.

[327] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple object tracking: A literature review. *Artificial Intelligence*, 293:103448, 2021.

[328] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era. *IEEE transactions on pattern analysis and machine intelligence*, 43(5):1578–1604, 2019.

[329] Muhammad Zubair Irshad, Thomas Kollar, Michael Laskey, Kevin Stone, and Zsolt Kira. Centersnap: Single-shot multi-object 3d shape reconstruction and categorical 6d pose and size estimation. 2022.

[330] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[331] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[332] Dimitrios Milios, Raffaello Camoriano, Pietro Michiardi, Lorenzo Rosasco, and Maurizio Filippone. Dirichlet-based gaussian processes for large-scale calibrated classification. *CoRR*, abs/1805.10915, 2018, 1805.10915.

[333] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. *CoRR*, abs/1809.11165, 2018, 1809.11165.

[334] Alexander G de G Matthews, Mark Van Der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. Gpflow: A gaussian process library using tensorflow. *J. Mach. Learn. Res.*, 18(40):1–6, 2017.

[335] GPy. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[336] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

Appendix A

Selected Simulation Results from Each Lake

We tested five unique paths per each lake and each motion. We select one case from each lake and motion type, and present our evaluations for each lakes with three types of figures: Top view, zoomed-in view, and motion estimation on x, y, z axis. For each lake, linear cases are shown in Figs. A.1 and A.2; mixed cases are shown in Figs. A.3 and A.4.

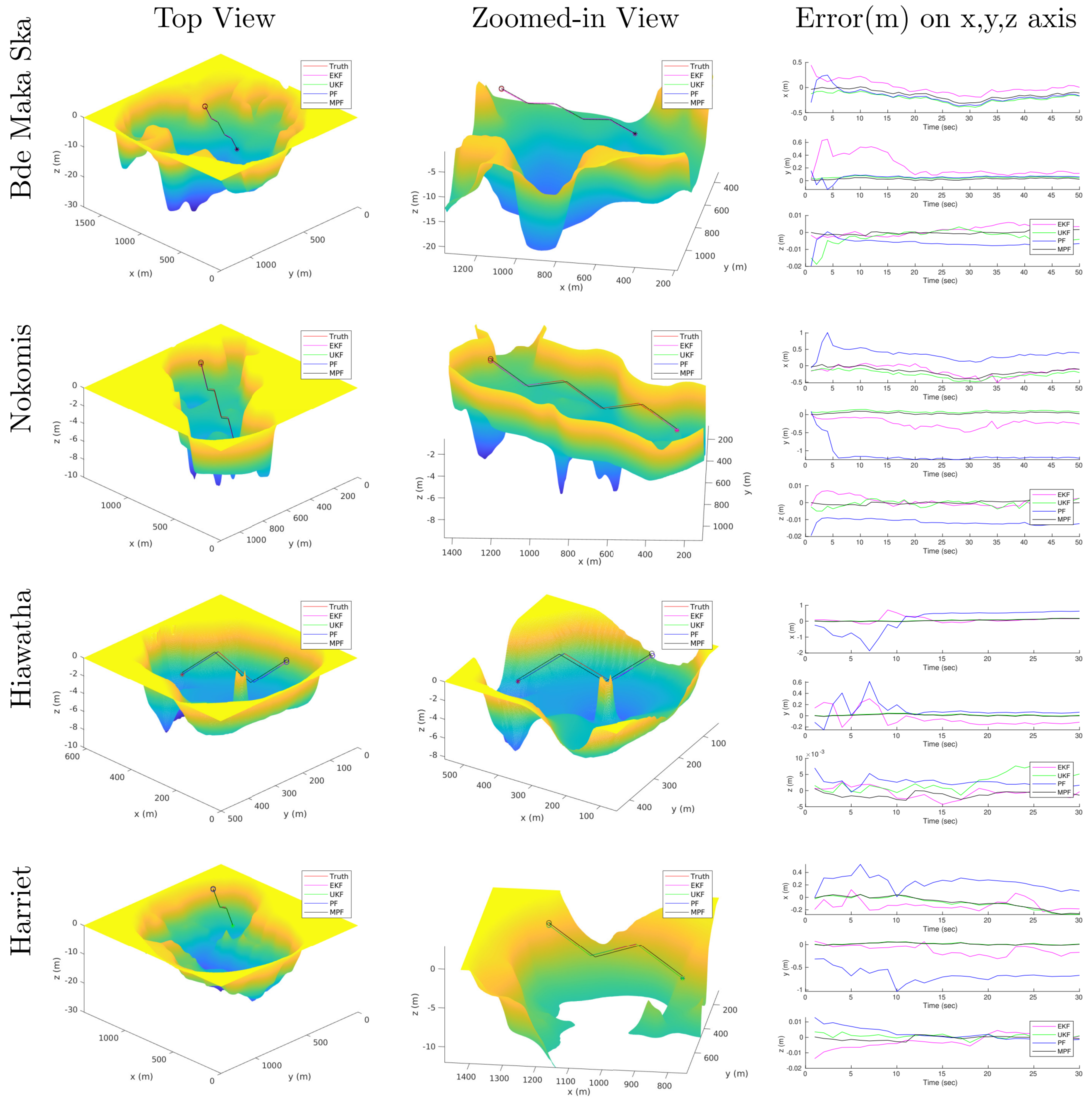


Figure A.1: Localization performance of the four algorithms for an AUV with linear motions (Best readability at 2x zoom).

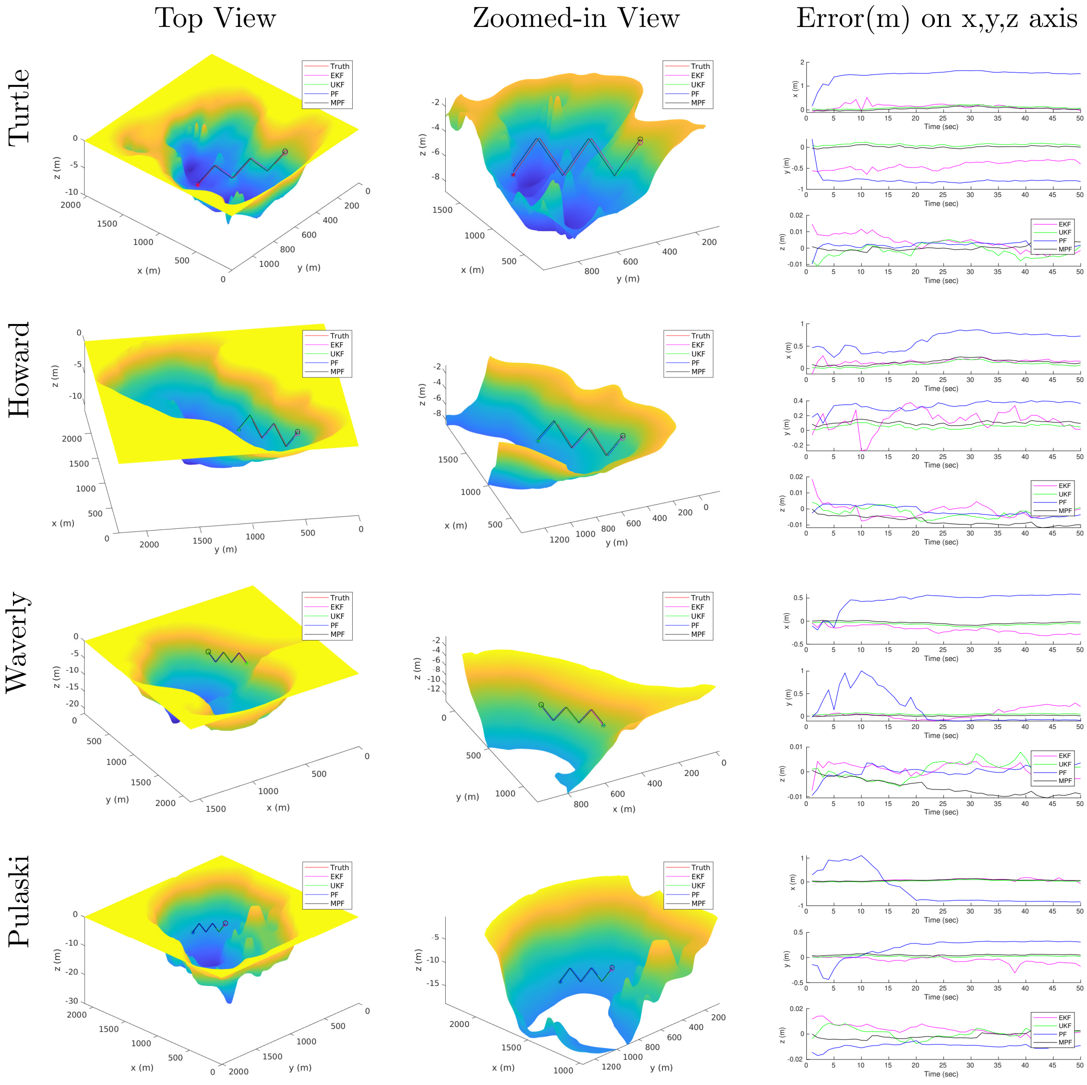


Figure A.2: Localization performance of the four algorithms for an AUV with linear motions (Best readability at 2x zoom).

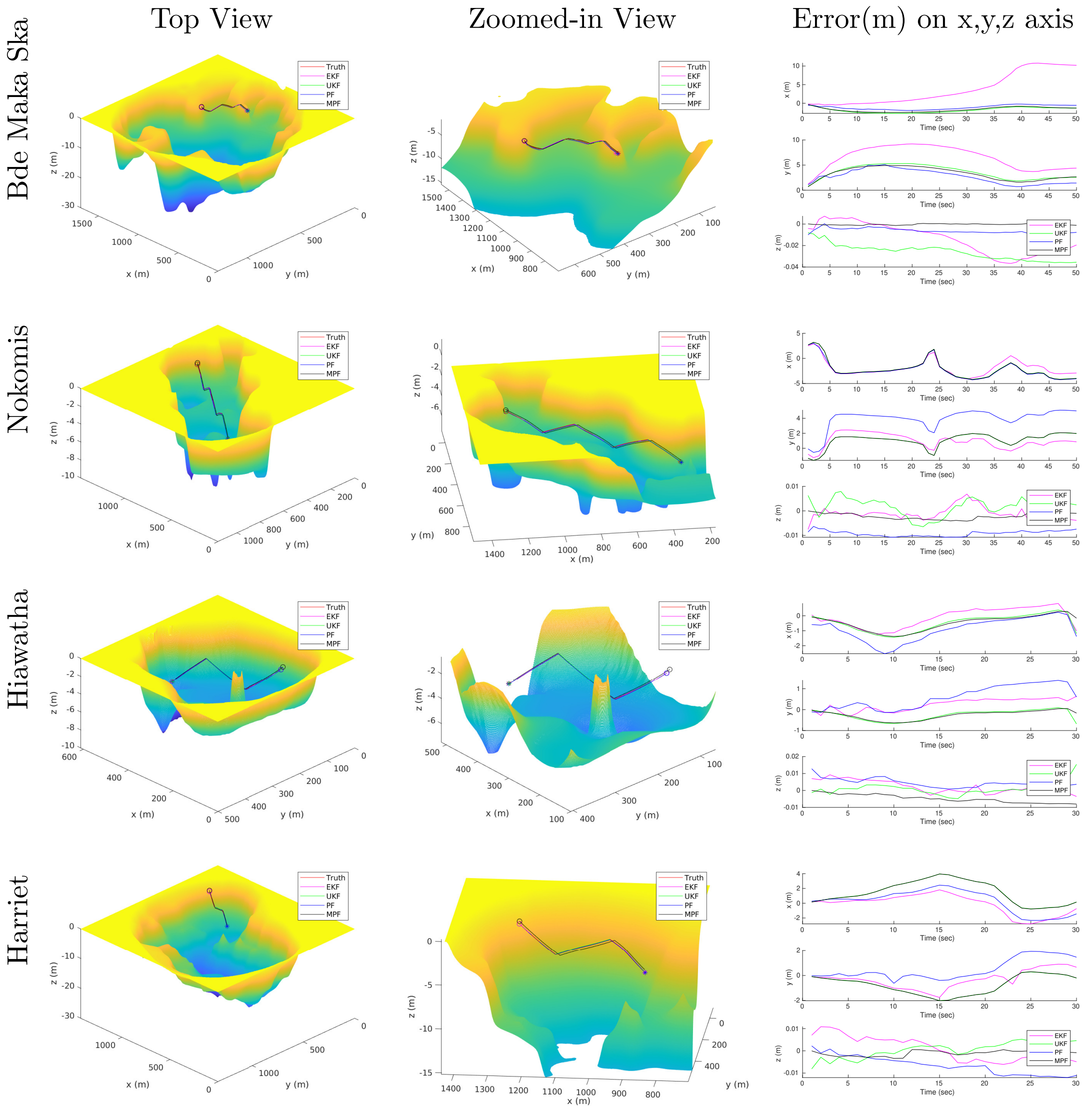


Figure A.3: Localization performance of the four algorithms for an AUV with mixed motions (Best readability at 2x zoom).

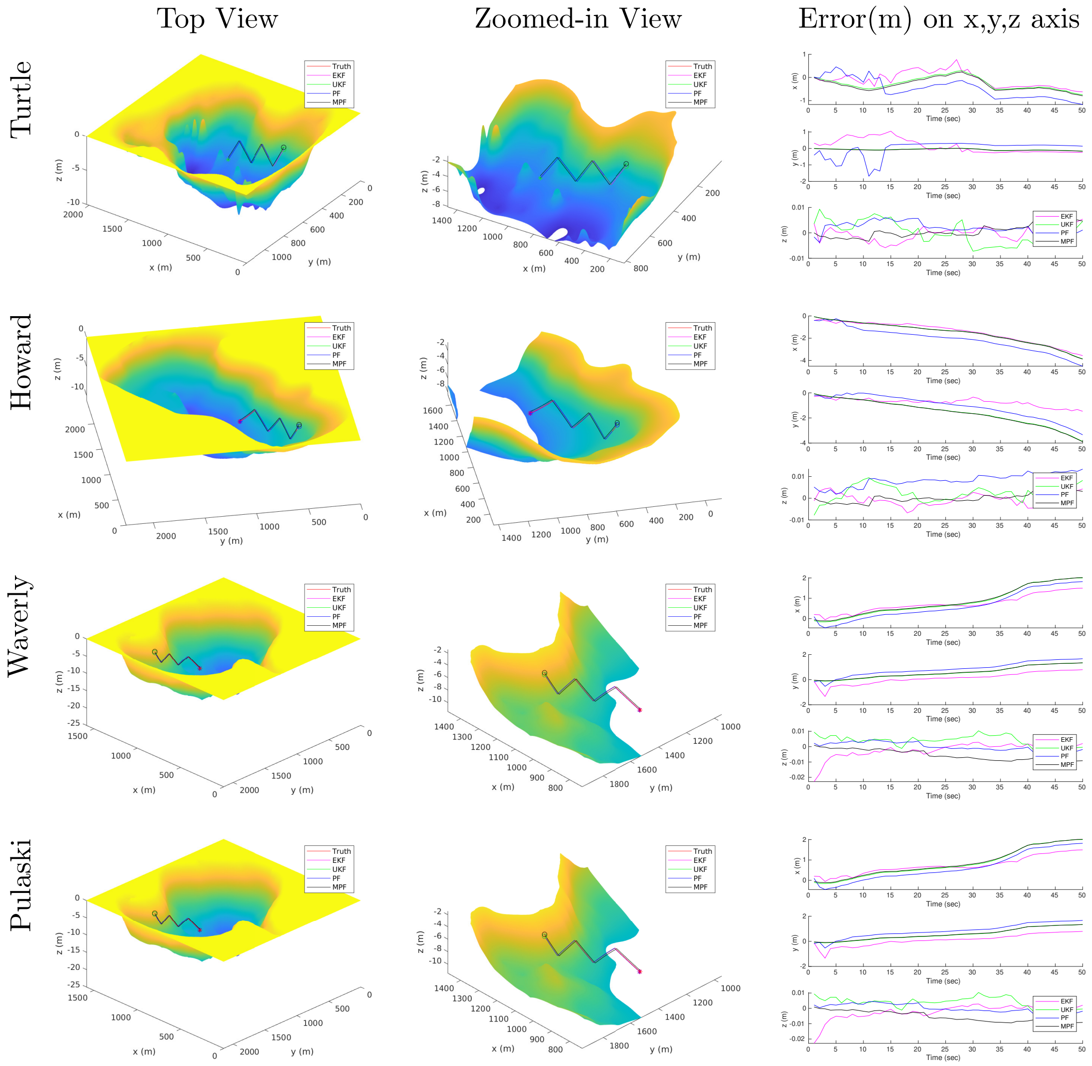


Figure A.4: Localization performance of the four algorithms for an AUV with mixed motions (Best readability at 2x zoom).

Appendix B

ROW-SLAM: Under-Canopy Cornfield Semantic SLAM

Most existing robot weeding algorithms focus on early-season weeding. During this time, since the canopy is not closed, usually a reliable GPS signal is available. Further, the robots can obtain top-down views, which are convenient for the detection and localization of weeds. However, mid-season robotic weeding still remains an open problem where the robot must operate under the canopy. During this season, detecting corn stalks and localizing them in a global coordinate frame is extremely challenging because there is very little space between the camera and the plants, and the camera motion is primarily restricted to be along the row. To overcome these challenges, we present a multi-camera system where the side camera (facing the plants) is used for detection, whereas the front and back cameras are used for motion estimation. Our proposed algorithm and evaluations are presented in detail in the following sections.

B.1 Related Work

The topic of robotic weed control [260] has been extensively studied over the last decade from various viewpoints: weed detection, field coverage, and vehicle development. Most recent weed control robots have been developed in the form of drones and ground vehicles. While drone-based approaches [261] can cover large areas, they only provide weed information to manage weeds via ground vehicles (*e.g.*, applying herbicide) [262].

Recently, Wu *et al.* [263] proposed a method using ground vehicles with downward-looking cameras to recognize weeds and remove them with end effectors. However, such an approach could fail as the crop canopies close and block the top view.

Additionally, Sivakumar *et al.* [264] use a front view camera to predict a vanishing point and corn lines for navigation under-canopy tasks. However, in the front view many details of the objects in the narrow corn row are lost due to the viewing angle. In Sec. B.4.3, we compare against their method for corn stalk plane estimation. Zhang *et al.* [265] use side view for corn stalk counting, but we show that side view also experiences frequent occlusion. To get accurate semantic features (*i.e.*, corn stalks, ground plane, corn stalk plane) from the map, we propose to combine the input from multiple views.

SfM [266] is one of the classical methods available to obtain the 3D location of the features. In our application, since there is very little space and frequent occlusion between the camera and the plants, the camera motion is also restricted. Classical SfM strategies perform poorly for estimating the motion when observing only the plants. Hasheminasab *et al.* [267] proposes to assist the SfM pipeline with GPS signals. Since GPS is not accurate enough for our application, we use SLAM odometry from the back camera and the front view corridor estimation to assist the SfM process using the side view.

For localization in outdoor environments, SLAM with RGB input [268, 269, 270] has been widely used. However, the localization accuracy of SLAM is sensitive to dynamic features. Recent works [271, 272] adopt neural networks to detect moving objects and remove their features. The remaining static features are then used to build the map. In our application scenario, most of the features above ground are susceptible to unconstrained motion due to the interference from wind or robot motion, making it challenging to detect dynamic features. So we select the ground view as input for the SLAM module to ensure the static map assumption. In Sec. B.4.4 we compare the performance using different views to support this choice.

Another aspect of obtaining semantic features is object detection. Some recent works [273, 274, 275] build offline approximation models for the objects of interest and detect them by model matching while building the map. However, such a method is unsuitable for corn stalk detection due to the lack of structured shapes in the non-uniform canopy. The dense occlusion from the leaves and the weeds makes it even more challenging to detect corn stalks by matching offline models.

Recent advances in deep learning allow accurate real-time object detection [276] and detect a wide range of objects from a single model with a large amount of data. Such advances have enabled adopting deep learning-based object detection in agricultural applications [277], and many studies [278] apply the object detection to detect weeds. However, these approaches are limited because they (1) require a large amount of weed data to train the models which is challenging to obtain, (2) can only be used for the field that existing weed information is previously known, and (3) may need a survey of a target field to collect field-specific weed information and its visual data before the deployment in a new field. In contrast, we focus on detecting corn and consider the remaining as weeds. Our approach can be applied to a broader range of fields and growing stages since corn has fewer variants compared to weeds.

To associate object detection across frames, object tracking algorithms [279] with visual sensors have been proposed to track a wide range of objects (*e.g.*, pedestrians [280], vehicles [281], sports players [282], crops [283, 284]). The tracking algorithms generally use an estimation model and data association method such as optical flow [285], Kalman filter, and Hungarian algorithm to associate detection results from the previous frame to the next frame. In our pipeline, we use Simple Online and Realtime Tracking (SORT) [286] due to its speed and accuracy, as demonstrated on the MOT15 [287] dataset.

In this work, we show that robotic weed control in mid-season corn row poses unique challenges to existing SLAM algorithms. With our system design and combining multiple existing algorithms, the results demonstrate that we can overcome the challenges.

B.2 Problem Formulation

We take as given that our environment consists of a field with corn planted in rows and have a vehicle that can traverse the field. A camera frame is rigidly attached to the vehicle and moves under the corn canopy with three cameras (one in the front-facing along the row, one on the side facing the corn stalk plane, and one on the back facing the ground as shown in Fig. B.1).

Our goal is to detect the corn stalks and localize them in the global coordinate

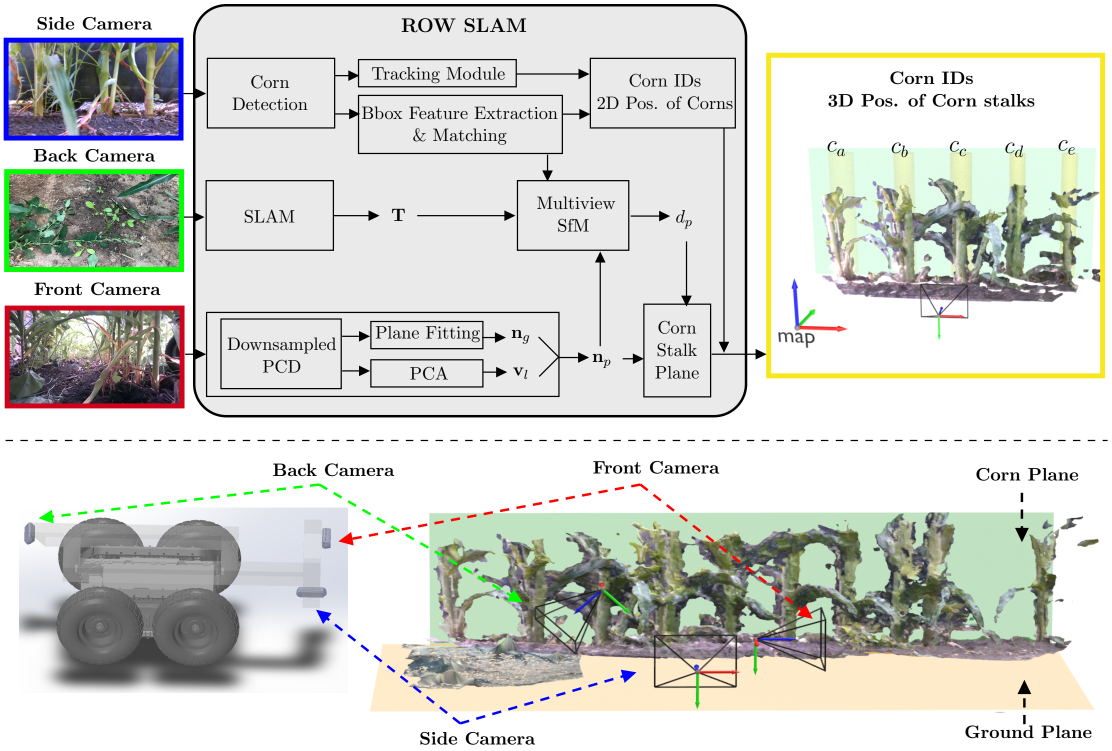


Figure B.1: (**Top**) Proposed pipeline of ROW SLAM: Our pipeline takes images from {front, back, side} cameras and yields 1) IDs for each corn (denoted with $C_{\{a,..,e\}}$ in the scene, and 2) 3D positions of the corn stalks (denoted by yellow cylinders). (**Bottom-Left**) Proposed robot design with multi-view cameras. (**Bottom-Right**) A 3D reconstruction sample of the corn field. Beige and green plane represent ground and corn stalk plane, respectively. Each camera pose is displayed separately.

frame. Therefore, we formulate this problem as a semantic SLAM problem where we aim to build a map with semantic features (corn stalks, corn stalk plane, and the ground plane). We choose corn stalks as the target semantic feature instead of weeds because, unlike the corn stalks, weeds in the field have various species and appearances. They also tend to vary across fields and regions. As a result, corn detection can be more consistent and robust than weed detection. Once corn stalks are identified, we can treat the remaining plants as weeds.

Table B.1: Summary of Notations

Notation	Description
$\mathbf{n}_p, d_p$	Normal vector and distance to the origin for the corn stalk plane (shown in Fig. B.2)
$\mathbf{n}_g$	Normal vector for the ground plane
$\mathbf{v}_l$	Intersection of the ground plane and the corn stalk plane, also the orientation of the corn row
$\mathbf{T}$	Relative transformation between sequential side camera poses

Bold uppercase letters for matrix, bold lowercase for column vector, normal ones are scalar.

B.3 System Overview

In this section, we introduce the necessary mathematical notations in Table B.1 and describe our system's hardware. We discuss the details of our approach for 3D corn stalk detection and localization next.

B.3.1 System Description

Our hardware system consists of three cameras (Fig. B.1 Bottom-Left). The front and side cameras (Intel Realsense D435) publish RGB images and depth images at 30 Hz. The back camera (Intel Realsense D435i) has a built-in IMU and is mounted at an angle facing the ground. The back camera publishes RGB and depth images at 30 Hz, gyro at 400 Hz, and accelerometer at 250 Hz. We use ROS for robot control and routing sensor data. For our experiments, we mount the system on a small 4-wheel rover from Rover Robotics.

B.3.2 Method Overview

As shown in Fig. B.1, we use RGB images from the side view camera for corn stalk detection. The detections only localize the stalks in 2D. Thus, to find the 3D position of the detected corn stalks, we also need to find the 3D pose of the corn stalk plane. We address the corn stalk plane estimation problem in two parts: (1) finding the plane

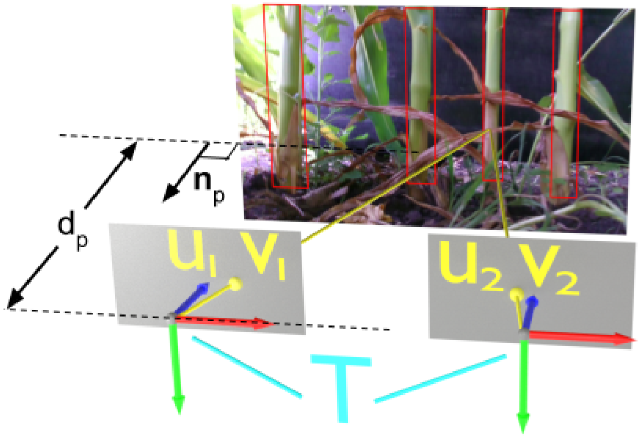


Figure B.2: Illustration for the multi-view SfM strategy. The relative camera pose (mint) is given by the SLAM module, and feature matching is masked by detected bounding boxes. $\mathbf{n}_p$ is the plane normal of the corn stalk plane, and d_p is the distance between the corn stalk plane and the left camera center.

normal direction, $\mathbf{n}_p$, and (2) finding the distance from the camera center to the plane, d_p, shown in Fig. B.2.

We use the front and back camera to assist with the estimation of the corn stalk plane $\mathbf{n}_p$ and d_p. With the RGB-D input from the front view camera, we estimate the ground plane normal direction $\mathbf{n}_g$ and the direction of the corn line $\mathbf{v}_l$. The corn stalk plane orientation is found by the cross product of these two vectors.

$$\mathbf{n}_p = \mathbf{n}_g \times \mathbf{v}_l \tag{B.1}$$

Then, we use the corn stalk plane normal $\mathbf{n}_p$, combined with side view object detection results and SLAM odometry $\mathbf{T}$ as the input for the Multiview SfM module to estimate d_p. Before going through the details of this module, we first introduce the detection and tracking.

B.3.3 Corn Stalk Detection

We implement our corn stalk detection model using Faster R-CNN [288] with MobileNet V3 [289] as the backbone and Fully Connected Network (FCN) as a head. We select MobileNet to provide fast inference, thus preventing the deep detection model from being a bottleneck in our pipeline. The backbone can be replaced with a larger network such as Resnet-18, -34, -50 [88] to improve the accuracy of the model. We train our model with pre-trained weights using the COCO dataset [65], and refine further using our corn stalk dataset.

B.3.4 Corn Stalk Tracking

With the outputs from the detection model, we implement the corn stalk tracking pipeline using (1) optical flow with centroids [290], and (2) Simple Online and Realtime Tracking (SORT) [286].

Optical Flow

We build the corn stalk tracking algorithm by applying optical flow and the corn detection results. The model detects bounding boxes for each corn stalk every 200 frames, and the centroids of the bounding boxes are calculated. After we obtain the centroids from the output of the detection model at frame X, the iterative Lucas-Kanade method with pyramids [291] is used to track each centroid using optical flow from frame $\{X + 1\}$ to $\{X + 199\}$.

SORT

SORT performs four actions for each input frame: detection, estimation, data association, and update tracking identities. The detections from Faster R-CNN are propagated to the next frame using a linear constant velocity model. The results are also utilized to update the target state with the Kalman filter [292]. When a new detection is obtained from the next frame, the Hungarian algorithm [293] is used to associate detections from the previous frame to the current frame using the intersection-over-union (IoU) metric. When the IoU metric is below a predefined threshold, then new identification of the detection is created.

B.3.5 Corn Stalk Plane Estimation

Plane Normal To get the ground plane pose, we first use the front view RGB-D input to compute the point cloud of the scene, followed by color thresholding and RANSAC [294] plane fitting to extract the ground plane. The points are usually denser when closer to the camera center, so we downsample the ground plane inlier points before applying principal component analysis (PCA). Since the corn row is narrow ($\approx 70cm$) and the corn line is parallel to the corn row direction, the two eigenvectors with the largest and smallest eigenvalues from PCA indicate the directions of the corn line $\mathbf{v}_l$ and the ground plane normal $\mathbf{n}_g$, respectively. The corn stalk plane normal is then computed by the cross-product of these two vectors.

Plane Distance Across different frames, the motion of the features on the corn stalks can be modeled by planar homography. However, instead of doing the full SfM purely on the side view input, we find it much more stable if we use the ground features to estimate the camera trajectory. We thus use the images from the back camera, providing a predominantly ground-plane view to avoid the unstable features from the corn leaves and weeds. With these images, we use the off-the-shelf RTAB-Map SLAM [270] package to perform visual SLAM and use the Sigma Point Kalman Filter [295] to fuse the SLAM odometry with IMU input. The resulting odometry runs at 200 Hz. After calibrating the back view and side view cameras, we can use the camera trajectory as is. Combining this with the estimated corn stalk plane normal, we formulate plane distance estimation as the following least square problem over re-projection error (shown in Fig. B.1 as the Multiview SfM block):

Given two side view RGB frames I_1, I_2, the relative transformation $\mathbf{T}_1^2$ between their camera poses (where $\mathbf{R}$ and $\mathbf{c}$ is the rotation and translation component) and the corn stalk plane normal $\mathbf{n}_p$, we want to estimate the plane distance d_p, as shown in Fig. B.2.

We first run Faster R-CNN on I_1, I_2 to get bounding boxes for detected corn stalks. Next, we compute SIFT features f_1, f_2 for I_1, I_2 only within the regions defined by the bounding boxes, and then use cross-matching and ratio test to find good matches between f_1, f_2. For each matched pair, let (u_1, v_1) and (u_2, v_2) be the corresponding pixel coordinates, $\mathbf{K}$ and λ_1 respectively be the camera intrinsic matrix and the depth for the first feature.

Letting $\mathbf{x}_1 = \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix}^T$, the 3D position of f_1 can be expressed as: $\lambda_1 \mathbf{K}^{-1} \mathbf{x}_1$.

Since we assume the feature lies in the corn stalk plane, according to the plane equation

$$\lambda_1 \mathbf{n}_p^T \mathbf{K}^{-1} \mathbf{x}_1 + d_p = 0 \tag{B.2}$$

We can rewrite λ_1 in terms of d_p:

$$\lambda_1 = -d_p / \mathbf{n}_p^T \mathbf{K}^{-1} \mathbf{x}_1 \tag{B.3}$$

After applying the rotation $\mathbf{R}$ and translation $\mathbf{c}$, the projection of the 3D position of f_1 in the second camera frame can be expressed as:

$$\lambda_1 \mathbf{K} \mathbf{R} \mathbf{K}^{-1} \mathbf{x}_1 + \mathbf{K}\mathbf{c} \tag{B.4}$$

If we substitute Eq. B.3 in Eq. B.4 we can rewrite Eq. B.4 as follows:

$$-d_p \begin{bmatrix} l_0 & l_1 & l_2 \end{bmatrix} + \begin{bmatrix} s_0 & s_1 & s_2 \end{bmatrix} \tag{B.5}$$

where,

$$\begin{bmatrix} l_0 & l_1 & l_2 \end{bmatrix}^T = \frac{-1}{\mathbf{n}_p^T \mathbf{K}^{-1} \mathbf{x}_1} \mathbf{K} \mathbf{R} \mathbf{K}^{-1} \mathbf{x}_1 \tag{B.6}$$

$$\begin{bmatrix} s_0 & s_1 & s_2 \end{bmatrix}^T = \mathbf{K}\mathbf{c} \tag{B.7}$$

The projected pixel position of f_1 in the second camera (u_2', v_2') is:

$$\begin{cases} u_2' = (-d_p l_0 + s_0)/(-d_p l_2 + s_2) \\ v_2' = (-d_p l_1 + s_1)/(-d_p l_2 + s_2) \end{cases} \tag{B.8}$$

We can linearize the re-projection error $(u_2 - u_2', v_2 - v_2')$ with respect to d_p so it can be estimated by least squares and RANSAC. However, notice that the least square robustness is sensitive to the second term $\mathbf{Kc}$ in Eq. B.4. So when applying this algorithm, we need to make sure the translation scale is above a threshold to ensure robust triangulation.

B.3.6 3D Localization

The 3D position of a corn stalk is obtained by the projection of the bounding box centroid onto the corn stalk plane. To show that it is more robust than directly using RGB-D input, we compare against RANSAC Plane Fitting in Sec. B.4.4. We also use the tracking module to link the 3D corn stalk positions with corn IDs. Such association across multiple frames allows us to reject outliers and false positives, making the localization more robust.

B.4 Experiments and Results

We present our data collection procedures, followed by the evaluation metrics for our method. Then we introduce the baseline methods we compare against.

B.4.1 Data Collection

The rover is controlled by a human operator to drive at around $0.3m/s$. To provide additional control of imaging conditions, we used a wheeled arch platform to go over the rows and provide cover for the rover. In the future, we are planning to combine the rover and the cover into a single robotic platform. Our collected dataset includes 8 different corn rows where 4 of them are from conventional fields, and the other 4 are from organic fields. We collected data across different growing stages throughout the mid-season (between V5 stage and V12 stage, late June to early August in Minnesota, USA).

B.4.2 Evaluation Metrics

We perform offline evaluations using the dataset collected in two different aspects, detection accuracy and localization. For evaluating the corn detection module, we label 763 images and separate them into 563 training images (235 images from conventional fields and 328 images from organic fields) and 200 test images (100 images from each field type). Training and test dataset images for each field type are picked from different rows to reduce the correlation between the datasets and provide an accurate measure of the generalization capability of our network. For localization under the canopy, it is difficult to get high-accuracy GPS signals reliably. So we manually measure the distance between neighboring corn stalks and use the corn stalks as landmarks along a straight line to evaluate the localization accuracy. We first line fit on the camera trajectory and project predicted target positions to the fitted line to reduce their dimension to one. To remove the duplicated predictions for a single target, we compute its nearest

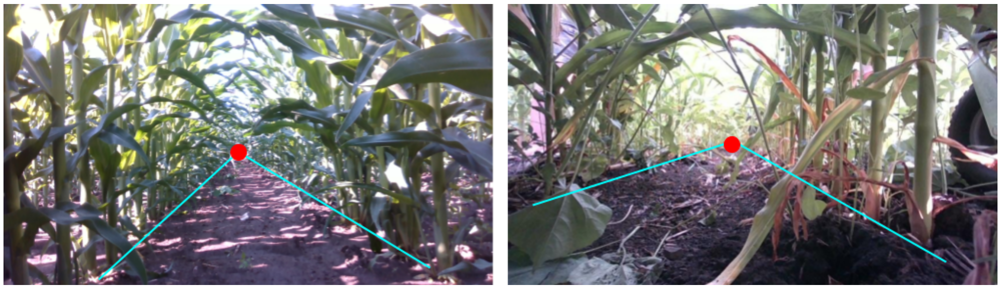


Figure B.3: Corridor projection of the corn row. Two corn lines (blue) are parallel and intersects at the vanishing point (red). Due to occlusion, the corn lines are usually not clearly visible.

neighbor from the world measurements for each predicted corn stalk position. The distance error ϵ_1 is computed as the mean absolute error of the neighboring corn stalks distance between prediction and real-world measurements. Finally, we measure the tracking variance through re-projection error ϵ_2. After accumulating the 3D position measurements for each corn stalk, we compute the mean position of the centroids and re-project them back to each frame. The re-projection error is computed by the mean pixel error between the centroids and the centroids of the bounding boxes proposed by the tracking module across all the frames.

B.4.3 Baselines

SLAM Input As the SLAM pipeline is sensitive to moving objects in the scene, we compare the robustness and accuracy of SLAM while using different input views. Specifically, we replace the SLAM input in Fig. B.1 with the front camera view or side camera view and re-run the Semantic SLAM pipeline.

Corridor Prediction The corridor prediction method replaces the corn stalk plane prediction module in Fig. B.2. It is also based on the planar approximation of the corn row. Since the corn stalk planes on both sides are parallel, their intersection line with the ground plane must also be parallel. Therefore by projective geometry, they intersect in the image plane at the vanishing point (Fig. B.3). After obtaining the 3D pose of the ground plane, the vanishing point can be used to get $\mathbf{v}_l$. The corn stalk plane normal $\mathbf{n}_p$ can be computed by the cross product of $\mathbf{v}_l$ and $\mathbf{n}_g$. Then, d_p can be obtained by the 3D position of any point along the corn line since it is also on the ground plane.

It is known that color thresholding yields unreliable results [264] for vanishing line

Table B.2: Faster RCNN Corn Detection Accuracy (%)

	Conventional field	**Organic field**
AP	26.2	47.8
AP_{50}	**83.1**	**89.1**
AP_{75}	9.3	46.6

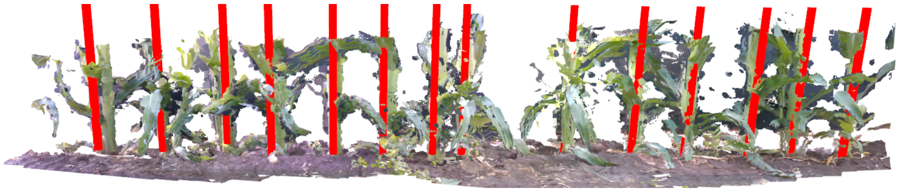


Figure B.4: The reconstructed point cloud of a corn row and the estimated corn stalk position (red cylinders) from ROW SLAM.

detection. Thus, we combine the ResNet34 head with a 3-layer multi-layer perceptron to predict the vanishing point pixel location and two slopes for the corn line. We do not predict the full parameters for the two lines so that we can enforce the projective geometry.

RANSAC Plane Fitting The RANSAC plane fitting method replaces the multi-view SfM module in Fig. B.2 for the estimation of d_p. After obtaining the side plane normal $\mathbf{n}_p$, one way to estimate d_p is by direct observation of the 3D scene in the side view. Therefore to support our claim on the low observability of the corn stalk plane, we present plane fitting in the side view as another baseline method. The detected object bounding boxes are used to find candidate points in the corn stalk plane. We only need the 3D position of one selected point with the plane normal to compute d_p. Then RANSAC is applied to make the estimation robust.

Optical Flow Tracking To ensure high localization accuracy, we use SORT as our tracking module which runs detection at every frame and uses the Kalman Filter to make the tracking more robust. In this baseline, we replace the tracking module with optical flow tracking. Unlike SORT, the optical flow tracking method will re-initialize target id every k frame. In our case, we choose $k = 200$.

Table B.3: Performance of Corn Stalk Localization (ϵ_1: Metric Error, ϵ_2: Centroid Re-Projection Error)

	Conventional field		**Organic field**	
	ϵ_1 (cm)	ϵ_2	ϵ_1 (cm)	ϵ_2
our approach	**1.8**	**5.6**	**3.5**	**10.4**
corridor prediction	8.5	19.3	9.7	27.8
front-view SLAM	2.9	8.2	6.2	13.3
side-view SLAM	4.5	16.8	7.1	15.5
RANSAC plane fitting	3.7	9.5	9.3	24.2
optical flow tracking	3.3	11.7	4.6	17.9

B.4.4 Results

We fine-tune Faster R-CNN for 100 epochs using the COCO-pretrained weight., and Table B.2 shows evaluation results for each field. It tracks objects at $\approx$ 11 FPS on a laptop (Intel i7-8850H, Quadro P3200, and 32GB RAM). The organic field case has higher Average Precision (AP) values since about 60% of the training dataset is from organic fields. AP_{50} (AP at IoU=.50) has the highest accuracy for both cases, and it is because irregular shapes of the corn tend to have relatively low IoU values for many detections while they are still detected as corn.

We compute the metric localization accuracy by comparing predicted target positions against the manually measured distance between corn stalks. We also evaluate the tracking variance through the re-projection error of the centroid. In Table B.3 we summarize the metric error ϵ_1 and centroid re-projection error ϵ_2 using the data from both conventional field and the more challenging organic field.

Our method outperforms all the baseline methods, which demonstrates its accuracy and robustness. As a comparison, the corridor prediction method shows fragile estimations of d_p. Due to the challenge of accurate estimation for $\vec{v_l}$ and projective geometry, the error of d_p is significantly impacted by the angular error of the vanishing line in the front view. As for the view selection in SLAM, we notice that the front-view SLAM achieves comparable accuracy with our method during a windless day in the conventional field. However, in general, the comparison between our ROW SLAM and front/side view SLAM shows that the static features are critical for accurate under-canopy localization in the corn rows. For the RANSAC plane fitting method, the performance drop indicates the challenge in directly observing the corn stalk plane from a single view. Lastly, we observe greater centroid position drift in image plane with optical flow tracking than SORT, which affects the localization accuracy and variance, proving the importance of robust visual tracking.

B.5 Conclusions

This paper presents our effort to address the under-canopy corn stalk detection and localization problem within narrow corn rows by proposing ROW SLAM and a multi-view camera system mounted on a ground vehicle. ROW SLAM combines existing algorithms (*i.e.*, object tracking, SLAM, SfM) and applies several geometric methods to accurately estimate the three planes that bound the row as well as individual plants. Our results demonstrate ROW SLAM yields an accurate map containing corn stalk positions and their IDs where existing SLAM algorithms fail. Future work will add an end-effector to our existing system to remove weeds physically. Furthermore, since the current approach suffers from drift over the long run, we are developing our algorithm to minimize the drift by using corn stalks as landmarks with an offline map.

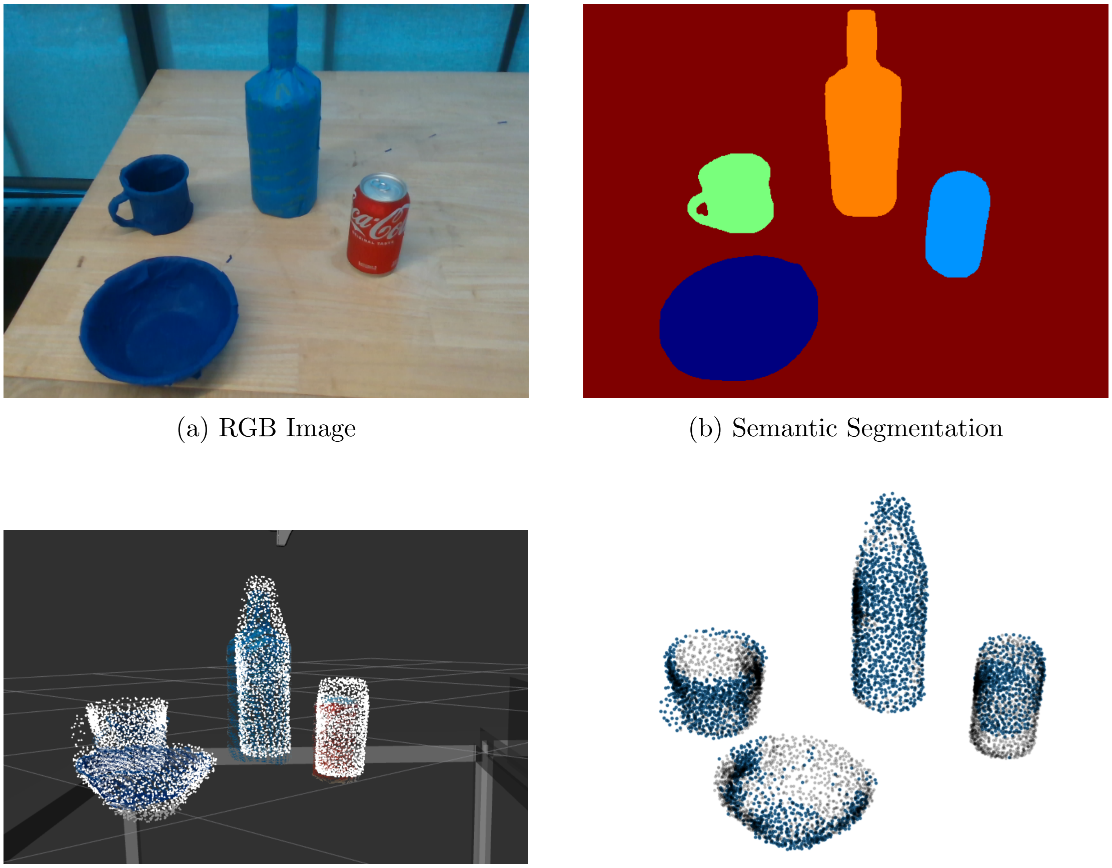
Appendix C

Semantic Mapping with Confidence Scores through Metric Embeddings and Gaussian Process Classification

Recent advances in robotic mapping enable robots to use both semantic and geometric understanding of their surroundings to perform complex tasks. Current methods are optimized for reconstruction quality, but they do not provide a measure of how certain they are of their outputs. Therefore, algorithms that use these maps do not have a way of assessing how much they can trust the outputs. We present a mapping approach that unifies semantic information and shape completion inferred from RGBD images and computes confidence scores for its predictions. We use a Gaussian Process (GP) classification model to merge confidence scores (if available) for the given information. The details of the proposed method and results are presented in the following sections.

C.1 Related Work

Many researchers have studied robotic mapping [296] over several decades. Seminal techniques such as occupancy grid-based representations [297, 298, 299], Simultaneous



(a) RGB Image

(b) Semantic Segmentation

(c) Shape Completion

(d) Uncertainty Evaluation

Figure C.1: An example scenario of real-world experiments: (a) An RGB image only provides a partial view of the scene, (b) Semantic labels for observed objects, (c) Predicted complete shapes (white color), (d) Complete shapes are filtered with confidence scores. Predictions from (c) are colored in gray/black, and uncertain parts of the complete shapes are marked with blue spheres.

Localization And Mapping (SLAM) [300] have been introduced in the last few decades. Geometric maps resulting from such methods allow robots to navigate their environments accurately, but some early works [297, 298] use a fixed grid size. However, memory consumption from such methods becomes a bottleneck when robots need finer resolution maps or are deployed in large-scale environments. Hornung *et al.* [299] introduced Octomap which uses an Octree data structure and voxel representation to address these issues. Other representations have also been explored to address the memory bottleneck. Newcombe *et al.* [301] proposed KinectFusion which uses a Truncated Signed

Distance Field (TSDF) representation to build a map as a part of their SLAM method. Henry *et al.* [302] presented RGB-D Mapping with surfel [303] representation. These are powerful geometric representations, but do not store semantic information.

To allow robots to perform higher-level tasks (*e.g.,* navigate and grasp a target object), researchers proposed approaches to create a map containing both semantic and spatial information of given environments (*i.e.,* semantic mapping [304, 305, 306, 307, 308] and semantic SLAM [309, 310, 311]). Semantic mapping approaches [310, 308] generally add semantic information on top of the existing mapping framework without considering the dependency between spatial and semantic information.

Recent advances in deep learning models [69, 312] provide accurate scene understanding from an RGB image. Merging the semantic information of images from different views for mapping has not received much attention. The merging process needs to be robust to both drastic view changes and the lack of generalization of deep neural network approaches. To handle such challenging cases, it is necessary to consider both geometric and semantic features during the merging process.

Researchers [313, 314, 315] use Gaussian Process (GP) [316] to model the underlying data correlation for the occupancy mapping problem. They formulate occupancy mapping as a regression problem and infer a map with GP. Their results [314, 315] show better mapping performance with GP than standard occupancy mapping [317], which assumes independence between neighboring cells. The success of GP in occupancy mapping motivated several efforts to apply it for semantic mapping. Unlike GP regression, GP multi-class classification requires approximate techniques for computing the latent variables' posterior distribution [318]. This is one of the challenges for adopting GP classification, and researchers reformulate the multi-class problem as multiple binary classifiers [319] or regression problem [320, 321].

Recently, there has been increasing interest [322, 323, 324] in applying a probabilistic approach to merge semantic information for mapping. Bowman *et al.* [322] associate metric and semantic class probability to formulate an optimization problem for SLAM. Doherty *et al.* [323, 324] considers uncertainty when fusing semantic and geometric information. For these approaches, uncertainty is provided with information from different sources. In addition, they are tested with large-scale and long-trajectory datasets (*e.g.,* KITTI [325]), which have relatively similar camera view angles across frames.

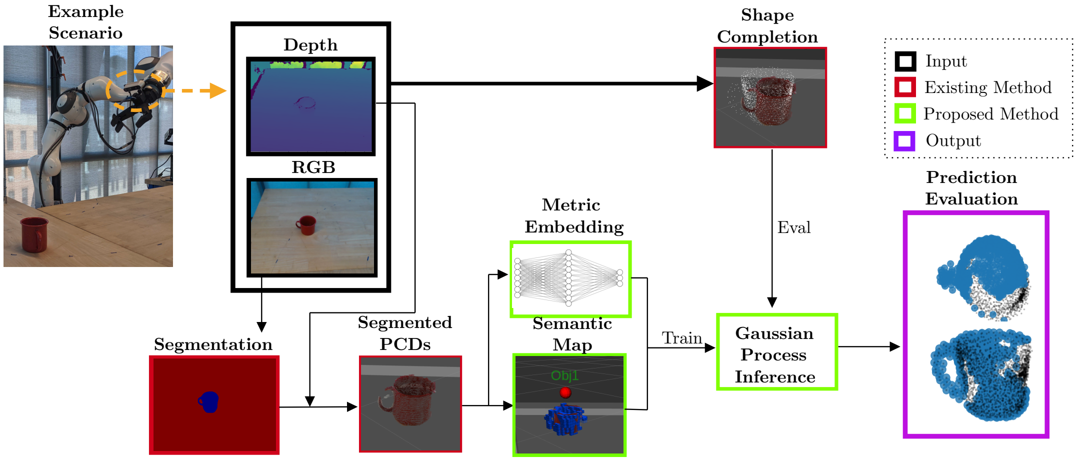


Figure C.2: Proposed Pipeline with an example scenario: our pipeline takes RGBD images and predicts objects' complete shapes with relevant confidence scores. Each component of the pipeline (segmentation, metric embedding, semantic map, GP model, shape completion, prediction evaluation) is presented with example results. The predicted mug pointclouds are colored in black. The predicted points with lower confidence scores than the desired confidence score are marked in blue.

In our approach, we address how to evaluate the uncertainty of predictions with a metric GP model. Additionally, we merge information from multiple sources using the uncertainty and present combined information with its uncertainty. Furthermore, we focus on mapping environments for robotic arms. Such environments are usually small-scale and may have a limited number of views due to physical constraints (*e.g.,* collision, space, etc.).

C.2 Methodology

Our proposed pipeline consists of four main components: Semantic Map module, Shape Completion module, Metric Embedding neural network module, and Gaussian Process (GP) Inference module. The pipeline takes RGBD images as inputs (Fig. C.2) and makes predictions for semantic labels and shape completion from each view. The confidence scores for semantic labels are given by the Mask R-CNN, and the ones for shape completion are computed by the metric GP model. The outputs from the pipeline have predictions for unseen parts of the objects, their confidence scores, and semantic labels. The final confidence scores are used to filter predictions, and the filtered point

Table C.1: Attributes stored in each voxel of a semantic map

No.	Attribute
1	x, y, z
2	R, G, B
3	Semantic Class (Mode of the histogram)
4	Confidence Score
5	Object ID

can be used to improve the map. In each section below, we discuss the details of each component.

C.2.1 Semantic Map

We design a semantic map module to extract geometric information from the inputs (RGBD images) and maintain relevant semantic information such as *object category, instance ID,* and *a confidence score* (Table C.1). We select Octomap [299] as our starting point to build the module since it is efficient and flexible due to Octree data structure. From the inputs, RGB images are used by Segmentation Network (MaskRCNN [326]) to predict instance segmentation for each pixel of the RGB images and obtain confidence scores of the prediction. By combining the segmentation outputs and depth images, we obtain segmented pointclouds. The pointclouds are then fed to: 1) Semantic mapping module, and 2) Metric embedding network (refer to Section C.2.3).

While our main focus is on the uncertainty evaluation of predictions, we develop a basic tracker for a semantic map to aid semantic information retrieval. With the tracker, the semantic map can maintain consistent object IDs over observed objects unless they are heavily cluttered. This is useful functionality to recover the objects' semantic information if a robot's motion causes drastic camera view changes (*e.g.,* no overlapping observations between two sequential views). Object tracking is a well-studied problem. Existing image-based object trackers [327] generally rely on overlapping information between different views, and they lose tracking information if the views do not share any visual cue. We propose a 3D geometry-based tracker to handle this issue. When we observe an object from each view, we calculate a centroid of the object's pointclouds.

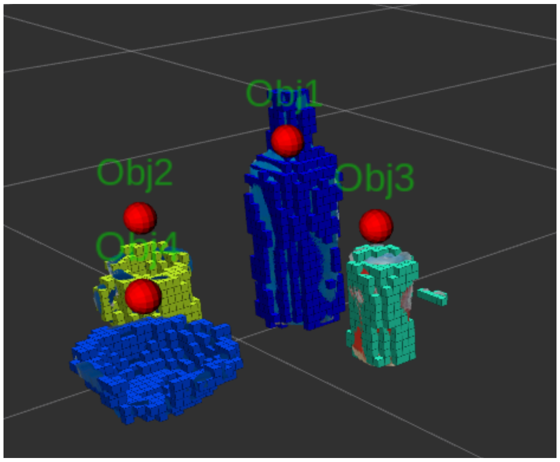


Figure C.3: A semantic map of the example scene introduced in Fig. C.1a. Each object's semantic labels are represented with different colors, and object trackers (red spheres) are visualized with their IDs.

We then use the Euclidean distance between the new centroid and existing centroids to update the set of centroids (*i.e.,* delete, append, and modify centroids). We then assign an object ID to each centroid in the set and store this information in our semantic map along with the semantic information in each voxel. In addition, we maintain a histogram for each semantic class in the voxel map to address temporary false prediction caused by a misclassification or noise from a camera.

The functionalities introduced above enable us to query the information at the object level from a semantic map (Fig. C.3). The information includes *semantic label, confidence score*, and *object ID* for every voxel belonging to any observed object. Although a semantic map contains information about the observed pointclouds, it cannot reason any unseen parts of the objects. To address such limits, we use a shape completion module and predict unseen parts of the objects.

C.2.2 Shape Completion

Shape completion is a task to predict the shape of a given object when only partial observations are available for the object. It has gained increasing attention in recent years [328]. However, most of the previous studies focus on predicting an object's shape in an object frame. This object frame-based shape completion requires an additional pose estimation of the object for robot manipulation. This is an issue when an object's

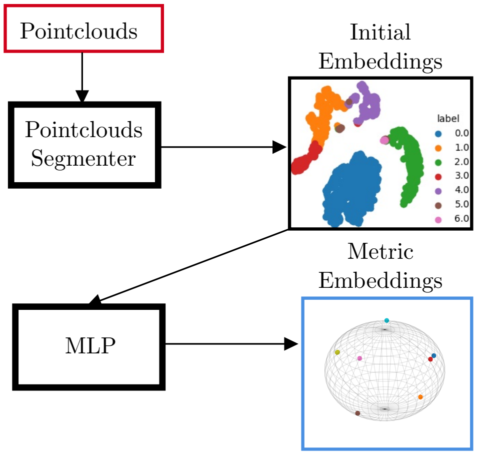


Figure C.4: Our metric learning method: It takes pointclouds from partial views as inputs (red) and outputs learned metric embeddings (blue).

pose is not available. To avoid this issue, we choose CenterSnap [329] to complete an object's shape in a camera frame, which can then be transformed to the world frame with a known robot pose. The outputs of this network are the 3D shape reconstruction for all the objects in the current scene and their categorical 6 degree-of-freedom poses and size estimates. The network is trained on the NOCS dataset [330] that contains six object classes. The input to the network is an RGB and depth image. The output is shape completed pointclouds and bounding boxes in the camera frame. Using the camera extrinsics previously calculated via the robot, we transform these into the world frame, which is defined at the base of the robot arm. While this module can predict unseen parts of objects, a way to evaluate each predicted point does not yet exist. To fill such gap, we introduce metric embedding for a GP classification model.

C.2.3 Metric Embedding

Embeddings refer to intermediate results that we can obtain from neural networks before they make predictions at the last layer. The intermediate results are often called *embeddings* since they *embed* some information; *semantics* and *geometric* information in our case. To obtain such embeddings for the segmented pointclouds ((x, y, z) coordinates), we use two neural networks sequentially (Fig. C.4): 1) DGCNN [331], and 2) Multilayer Perceptron (MLP) network. First, we feed segmented pointclouds to DGCNN and take the $7D$ embeddings from the last dense layer of DGCNN. Next,

Table C.2: Two types of inputs for the GP model

Type	Train X	Train Y	Test X
Classic GP	x, y, z	Semantic Label	x, y, z
Metric GP	Metric Embedding	Semantic Label	x, y, z

we pass the embeddings to the MLP network that we designed to enforce embeddings to be in a unit-sphere metric space. The MLP consists of four dense layers $((7 \times 250), (250 \times 100), (100 \times 50), (50 \times 3))$, and is trained with A-Softmax loss [220]. This loss forces the MLP to generate embeddings on a unit-sphere surface upon training the network. As a result, we yield $3D$ metric embeddings for segmented pointclouds from this step. Free space does not have any geometry, so its embedding cannot be learned. Therefore, we assign a zero embedding to free space explicitly. To make objects' embeddings equidistant from the free space embedding, we constrain them to lie on a unit-sphere.

C.2.4 Gaussian Process

With the metric embeddings of the segmented pointclouds, we select the GP-based model for the prediction since GP [316] is well-known for its ability to capture the underlying structure from training data points and use the structure to make predictions for test data points. Particularly, we choose the Dirichlet GP classification model [332] to make predictions from our embeddings. This model solves a classification problem by transforming class labels with Dirichlet distribution. With the metric embedding as training points, we call this GP model *metric GP*. However, its computational cost often becomes a bottleneck as the size of the data increases. A few libraries [333, 334, 335] have been introduced to alleviate this problem by optimizing the framework. Among them, we use GPytorch [333], an open-source library that allows us to run fast GP inference on GPUs using Pytorch [252] and CUDA. In addition, it provides recent advanced GP models.

With the selected GP classification model, we present two approaches with two types of inputs (Table C.2): 1) Classic GP, and 2) Metric GP. For the classic GP, we use pointclouds (x, y, z coordinates of each point) to construct the covariance matrices with a kernel. This approach relies on the geometric distances between training and test points to make inferences, which may not be able to exploit underlying semantic information from the training data.

The GP model can reason underlying data features better if the training points embeds semantic information. This motivated us to construct the GP model using training points obtained in a learning metric embedding space, which is a unit-sphere metric space in our case.

Once we obtain the metric embeddings for the training points, we append embeddings to relevant points. As a result, the training data becomes 6-dimensional (x, y, z, e_1, e_2, e_3). Eq. C.1 shows the covariance matrix K for the GP model.

$$K = \begin{pmatrix} K(X,X) & K(X,X_*)_*^T \\ K(X,X_*) & K(X_*,X_*) \end{pmatrix} \tag{C.1}$$

where X is the training points, and X_* is the test points.

For our GP model training, we construct the $K(X,X)$ matrix using the metric embeddings (e_1, e_2, e_3) of the training points. For the inference, we calculate $K(X,X_*)$ with $3D$ coordinates of both training and test points. Due to the fact we do not have any prior information (*i.e.,* colors, cluster shapes) about the test points, which are unknown except for their coordinates, we use the 3D coordinates of the training points to find the correlation between the training and test points via $K(X,X_*)$. Similarly, we compute $K(X_*,X_*)$ with $3D$ coordinates of the test points.

When the GP model makes inferences, it provides variance (σ) along with predictions. We use the variance to output the raw confidence score (s in Eq. C.2) of each prediction by measuring the variance reduction. Then we normalize s to obtain the score $\tilde{s}$ ranging between 0 and 1.

$$s = \frac{1-\sigma}{1-\sigma_{min}} \tag{C.2}$$

$$\tilde{s} = \frac{s - s_{min}}{s_{max} - s_{min}} \tag{C.3}$$

To compute our final prediction confidence scores for shape completion parts and their semantics, we multiply the normalized confidence score (Eq. C.3) by the confidence score given by the segmenter.

C.3 Experiments and Results

C.3.1 System Overview

We set up our experiments in both simulation and real-world settings to evaluate our approach both quantitatively and qualitatively. Our simulator is built using Pybullet [336]. For real experiments, we evaluate our approach with a Panda Arm robot, which has 7 degrees-of-freedom. We mount a Realsense D-435 (RGBD) camera on the wrist of the robot. For both the simulation and real setup, we use Rviz to visualize the outputs from our algorithms. We use open source libraries in ROS2 Foxy with Ubuntu 20.04 OS. The underlying communication layer is ROS2, allows for seamless communication between the robot (both in simulation and real), our algorithm, and visualization modules.

C.3.2 Evaluation Metrics

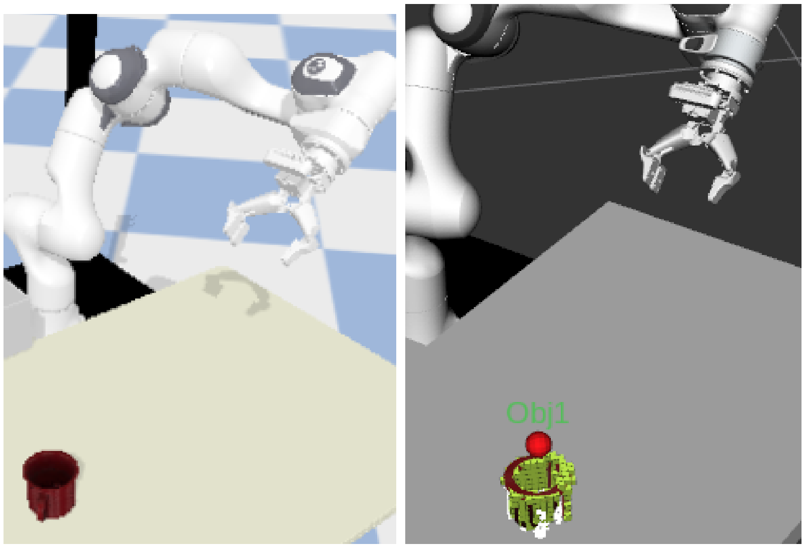
To evaluate our approach, we build a confusion matrix to evaluate the prediction accuracy. The matrix has four elements, and they are defined as follows for our experiments:

- True Positive (TP): Points predicted by GP as part of the object that lie on the ground truth object mesh.

- True Negative (TN): Points predicted by GP as free space that doe not lie on the ground truth object mesh.

- False Positive (FP): Points predicted by GP as part of the object that do not lie on the ground truth object mesh.

- False Negative (FN): Points predicted by GP as free space but lie on the ground truth object mesh.



(a) Simulation (b) Visualization

Figure C.5: An example scenario of our simulation experiments: (a) A mug on the table rendered with Pybullet, (b) Visualization results in Rviz: It shows the mug pointclouds and semantic map with its object tracker.

C.3.3 Baselines

Due to sensor noise and lighting conditions, it is generally infeasible to obtain ground truth pointclouds for objects in a real-world experiment setup. We ran our baseline experiments in simulation to evaluate our method with ground truth pointclouds sampled from known object meshes as shown in Fig. C.5. For the simulation experiments, we use two types of objects (*e.g.,* bowl and mug) while for the real robot evaluation we added multiple objects in the scene including a bottle and a can.

We designed our experiments to compare the performance of our proposed pipeline where we apply the metric GP model and the classic GP model. The metric GP model uses metric embeddings of the observed pointclouds to train the model, and the classic GP takes the pointclouds without any further processing. Additionally, the pipeline is tested with and without the shape completion (SC) module for both the GP models to see if the performance of each GP model relies on the shape completion module's presence. For the case without the SC module, we sample test points for inference randomly in the object bounding box. For the other case, we use points from the SC module as test points. In total, we ran four runs for a mug and bowl, respectively.

Table C.3: The results of baseline experiments for mug and bowl in Simulation

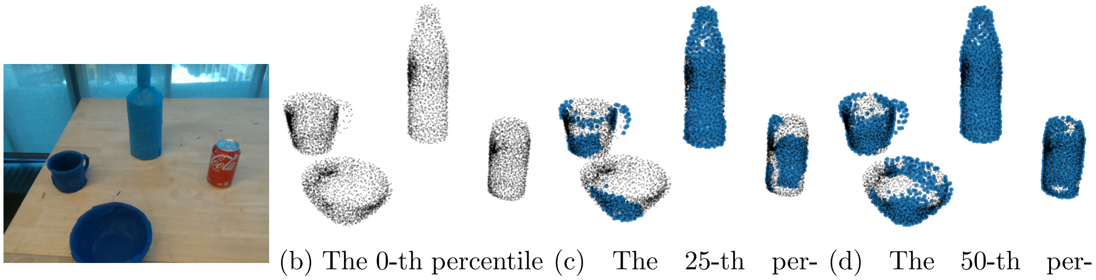
		TP	TN	FP	FN	Acc.
Mug	Classic GP with SC	0.066	0.793	0.125	0.016	0.345
	Metric GP with SC (Ours)	0.161	0.711	0.083	0.045	**0.659**
	Classic GP without SC	0.356	0.139	0.414	0.091	0.462
	Metric GP without SC	0.404	0.161	0.345	0.090	**0.539**
		TP	TN	FP	FN	Acc.
Bowl	Classic GP with SC	0.265	0.024	0.598	0.113	0.307
	Metric GP with SC (Ours)	0.406	0.012	0.502	0.080	**0.447**
	Classic GP without SC	0.246	0.206	0.477	0.071	0.340
	Metric GP without SC	0.249	0.244	0.445	0.062	**0.358**

C.3.4 Results

For our experiments, we trained the Mask R-CNN with the NOCS real and camera datasets with 10 epochs on NVIDIA 1080 TI GPU. Additionally, we trained our metric embedding model with $30,000$ scenes where each scene has $2,048$ data points on NVIDIA 3090 GPU. Pretrained weights are used for the shape completion network(CenterSnap). Lastly, we built both GP models with $3,600$ training points and queried $1,000$ test points from them using Matern Kernel.

The results of the baseline are shown in Table C.3. The confusion matrix results for each case are normalized as all evaluations are done on 1000 points. The results demonstrate that our method outperforms all the other baseline experiments for both mug and bowl classes. We noticed that the shape completion (SC) module improved the performance of our method regardless of the choice of the GP model. Additionally, the mug class shows higher accuracy in general than the bowl class. The mug model in our simulation has a more similar shape to the mug model in the NOCS dataset compared to the bowl case.

We also qualitatively evaluate our proposed approach in a real-world environment. Fig. C.6 shows the results, where the blue-colored points represent the points with lower confidence scores than the threshold confidence score. In other words, any point that has a lower confidence score than the desired confidence score is colored blue. From the observed scene, our approach predicted shape completion and semantic information. Based on the observed prior (Fig. C.6a), each point in the prediction has different



(a) An RGB Image

(b) The 0-th percentile confidence score

(c) The 25-th percentile confidence score

(d) The 50-th percentile confidence score

Figure C.6: An example of real-world evaluation on the robotic platform. The original predicted shape completion points are colored in black. The predicted points with lower confidence scores than the desired confidence score are marked in blue. As the desired confidence score increases from (b) to (d), the number of uncertain points are increasing visually (*e.g.* the size of the blue region grows).

confidence scores. The complete shape points are then filtered with the $n-$th percentile confidence scores (*i.e.,* threshold confidence score) in Fig. C.6b-C.6d. As the threshold confidence becomes higher, we observed that the number of uncertain predicted points is increasing. Observed points from the RGB image tend to be more certain points. Also, the bottle class has more uncertain points compared to other classes since its predicted complete shape is quite different from the shape of the bottle we used for the experiment. This shows the difference between the object model predicted from the SC module and the observed model can degrade the performance of our approach. As our qualitative result shows, the capability to filter predictions with confidence scores will give a robot to adjust the threshold confidence score based on its task for mapping its environment. flexibility to a robot to decide how much predicted information it will use for mapping. In other words, the robot can use higher confidence scores to filter the prediction with higher-risk tasks while lower confidence scores can be used for lower-risk tasks.

C.4 Conclusions

This paper presents our novel mapping method that unifies semantic information and shape completion predicted from partial-view RGBD images and calculates confidence scores for its predictions. Our metric Gaussian Process (GP) classification model merges confidence scores (if available) for the given information. A novel aspect of our method is to transfer sensor measurements to a learned metric space for training the GP model. From this metric GP model, we can measure the uncertainty of the unified information and make predictions more accurately than a classic GP model. The results demonstrate that our approach successfully merges the given information (*e.g.,* semantic information and shape completion) from partial views and computes their confidence scores. Future work will include building a grasping algorithm that relies on our modules and demonstrate its robustness to occlusions in the scene. In the long term, we plan to focus on generalizing our approach so that it can incorporate a larger number of sensor inputs.