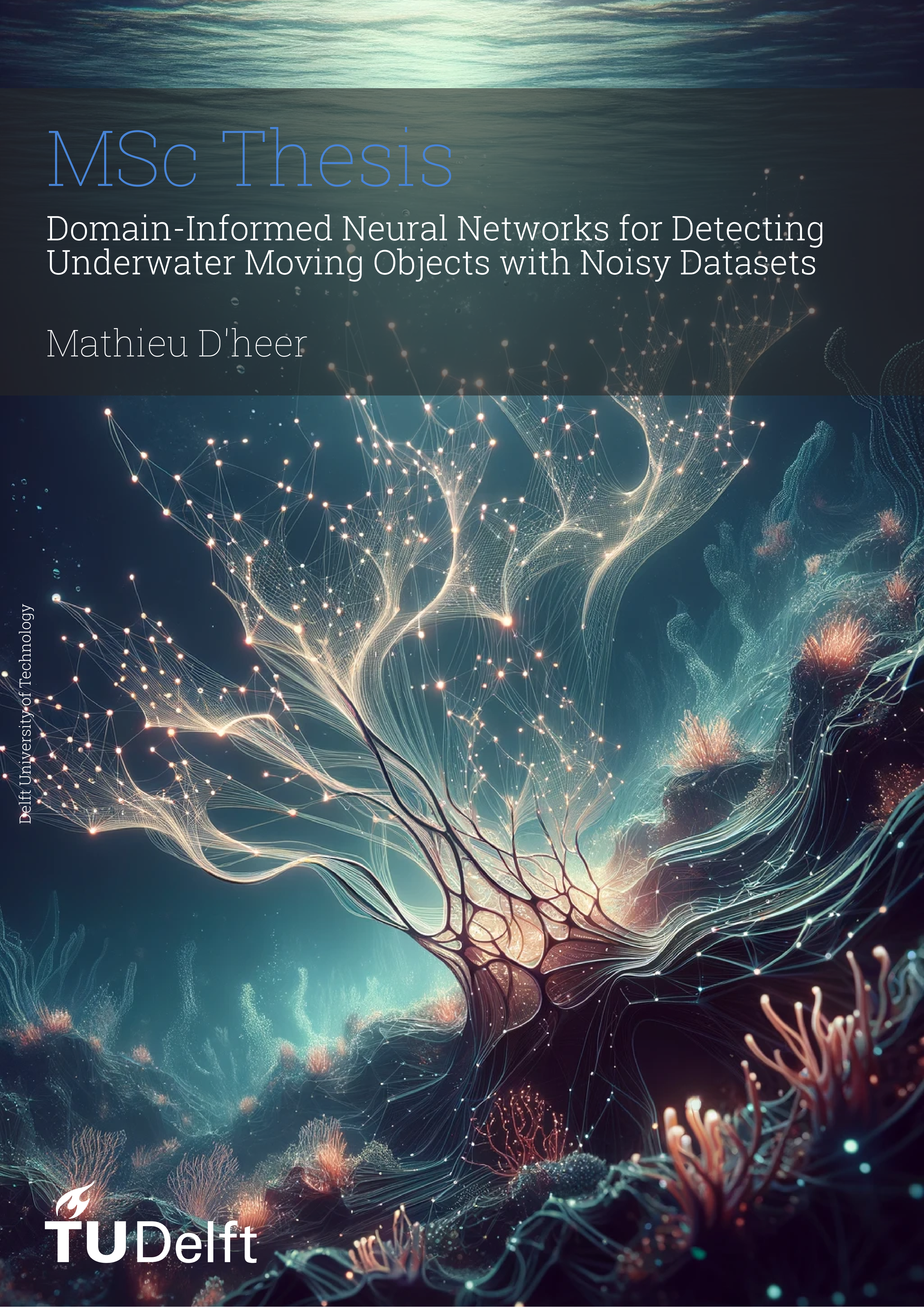


MSc Thesis

Domain-Informed Neural Networks for Detecting Underwater Moving Objects with Noisy Datasets

Mathieu D'heer

Delft University of Technology

TU Delft

MSc Thesis

Domain-Informed Neural Networks for Detecting Underwater Moving Objects with Noisy Datasets

by

Mathieu D'heer

Student number: 4557476

In partial fulfillment of the Master of Science at Delft University of Technology

Cover image generated using Dall-E.

Supervisor:	Dr. ir. A. Jamshidnejad
PhD Candidate:	Ir. A. Ilioudi
Faculty:	Faculty of Aerospace Engineering, Delft
Thesis committee:	Dr. ir. E.J.J. Smeur
	Dr. ir. A. Jamshidnejad
	Dr. ir. M. Guo

Acknowledgments

This thesis marks the end of my academic journey. As I reflect on these years, I recognize them as both rewarding and challenging. I am immensely grateful for having chosen to study Aerospace Engineering at Delft University of Technology. Despite the hard work and numerous sacrifices required, the rewards make it all worthwhile. The opportunity to forge lifelong friendships with peers from around the world, participate in a minor program in Singapore, and experience zero-gravity flight are just a few of the unique experiences that have enriched my life.

I extend my gratitude to my supervisors, Dr. Anahita Jamshidnejad and Athina Ilioudi, for their continual advice and guidance. Their flexibility with the timeline, particularly in allowing me to take a three-month break to participate in one of The Ocean Cleanup's offshore campaigns, was invaluable. Their motivation and support were crucial in helping me make consistent progress while working part-time.

To my parents, my brother, and my girlfriend Magali, who have always believed in me: I cannot thank you enough. Your unwavering support helped me persevere through the darkest and most difficult times. To all the friends I made along the way, thank you for making this journey unforgettable and filled with joy. I have found lifelong friends, and though I left home, you all helped me build a second one.

I am also thankful to my colleagues at The Ocean Cleanup for their support and understanding. I greatly appreciate the flexibility I was allowed to finish the thesis and the trust placed in my work.

I now close this chapter of my life with a smile. This journey has shaped the person I am today, and I look forward to the future with optimism. After all, it only gets easier from here, right?

Mathieu D'heer
Delft, July 2024

Part I

Scientific paper

Domain-Informed Neural Networks for Detecting Underwater Moving Objects with Noisy Datasets

M. D'heer

Delft University of Technology, 2629HS Delft, the Netherlands

Keywords: Marine pollution, neural networks, object detection, knowledge distillation, synthetic data

Marine pollution is a critical issue impacting the global community, with underwater waste a particularly daunting challenge. While autonomous detection and collection of underwater waste is highly desirable, these are extremely difficult tasks. This difficulty arises from the intrinsic complexities of the aquatic environment, including variable lighting conditions, reduced visibility, and the complex nature of water currents. This paper focuses on novel approaches for autonomous detection of underwater waste and proposes to incorporate domain knowledge to refine deep-learning-based underwater object detection techniques. More specifically, the domain knowledge is represented with models in the state space form that describe the motion of the target objects to assist the classification of objects. Moreover, optical flow is combined with a k-means clustering algorithm to extract the trajectory of the target objects from videos. These trajectories are subsequently fed into a neural network that is trained using knowledge distillation, enhanced with domain knowledge. For our experiments, a simulator is devised to facilitate the creation of a dataset for developing and testing the proposed architecture. The results of the experiments demonstrate that including domain knowledge within the object detection approach with neural networks provides numerous substantial advantages, including enhanced robustness against noisy and poorly labelled data, facilitation of semi-supervised learning, and consistent superiority in accuracy over the baseline scenario. Additionally, combining the domain knowledge with a neural network significantly increases the computational speed of object detection compared to using the standalone domain knowledge module.

Table 1: Mathematical notations

Symbol	Description
$\rho(\cdot)$	Density function
d	Depth
v_x^{obj}	True object velocity in m/s
x_i^{obj}	Horizontal position of the object in m
x_i^{rov}	Horizontal position of the ROV in m
i	Frame number
f^{s}	Frame rate in Hz
$I(\cdot,\cdot,\cdot)$	Intensity function of optical flow
x^{pixel}	Horizontal pixel location in a frame
y^{pixel}	Vertical pixel location in a frame
t	Time
v_x^{pixel}	Horizontal pixel velocity
v_y^{pixel}	Vertical pixel velocity
$\vec{c_k}$	Motion constraint vector
$\vec{a_n}$	Vector of feature parameters for component n
$\vec{m}$	Mixture model weight
$p\,(.\mid .,.,.)$	Motion constraint probability
λ^{l}	Weight of label-based loss
λ^{r}	Weight of regularization term
λ^{k}	Weight of knowledge-based loss
$L^{\text{l}}\,(.,.)$	Label-based loss function
$L^{\text{k}}\,(.,.)$	Knowledge-based loss function
$R(\cdot)$	Regularisation function
ρ^{fog}	Density of the underwater fog
ρ^{water}	Density of the water
x^{max}	Maximum x-distance object can spawn
y^{global}	y-coordinate in a real-world coordinate system
f	Focal length

fov^{v}	Vertical field of view
y^{centre}	Centre location of pixels on vertical axis
d^{cam}	Distance from the object to the camera in m
h^{pix}	Height of the image in pixels
g	Gravitational acceleration
m^{o}	Mass of the object
ρ^{water}	Water density
V^{o}	Volume of the object in m^3
$c^{\text{d},o}$	Drag coefficient
A^{o}	Surface area in m^2
v_z^{current}	Water current velocity in m/s
F^{fz}	Fish force
f^{tail}	Fishtail frequency
L^{ce}	Cross entropy loss
L^{kl}	Kullback-Leibler divergence
P^{*}	True probability distribution
P	Predicted probability distribution
P^{soft}	Softened predicted probability distribution
P^{soft}	Softened predicted teacher distribution
T	Temperature parameter
α	Weighting parameter balancing two losses

I. Introduction

While the manufacturing of disposable products has significantly increased in recent decades, the ability to sustainably manage the waste has not kept pace with the rising production. Especially in underdeveloped areas, plastics and their resulting environmental pollution have reached alarming levels, triggering the need for a global Plastics Treaty by the United Nations [1]. It is estimated that there are between 50 and 75 trillion pieces of plastic in the oceans, and this number is continuously rising [2].

Trash cleanup involves inherent difficulties and challenges, but the specific environment dictates the feasibility of the cleanup endeavours. Specifically, the removal of underwater debris is extremely resource- and time-intensive, making autonomous methods the only feasible and scalable solution. The first step in enabling these autonomous approaches is to achieve automated debris detection from captured images. This task is difficult even with the recent advancements in Artificial Intelligence (AI), such as large language and multimodal models that are specifically suited for classification (i.e., classifying an input image into the correct class) [3, 4]. In fact, the underwater environment poses various challenges compared to other object detection problems, including low contrast, uneven illumination, complex backgrounds, and high noise caused by water turbulence, marine snow, backscatter and more [5, 6, 7, 8].

The problem of underwater object detection can also be observed from an information theory perspective [9]. Each image has a limited amount of information that is encapsulated in its pixels. Convolutional Neural Networks (CNNs) are methods that extract and interpret this information [10]. The state-of-the-art neural-network-based methods, such as YOLOv8 [11], EfficientNet [12], RetinaNet [13], and Faster R-CNN [14] are highly capable of extracting and interpreting the available information in images. Enhancing performance beyond this level necessitates expanding the informational base. The most direct strategy to increase the available information entails transitioning to a video-based approach instead of an image-based one. Leveraging this additional information can be done in two ways: implicitly and explicitly. The implicit approach relies heavily on large existing datasets that allow the neural network to deduct the underlying physics and knowledge from the data. Video-based object detectors using temporal data are notoriously hard to train [15] and datasets with sufficient size for training them are hard, or even impossible, to collect. The explicit approach relies on formally defining information and integrating this information into a neural network. This method can be referred to as informed machine learning [16]. Informed machine learning leverages two sources of information: data and prior knowledge. An example is described in [17], where a neural network is trained from data and existing knowledge, i.e., it should satisfy the rule that the water density $\rho(\cdot)$ in a lake at two different depths d_1 and d_2, with $d_1 < d_2$, must follow $\rho(d_1) < \rho(d_2)$. A taxonomy of informed machine learning [16], where prior knowledge with different sources corresponding to certain representations and different integration methods are used, can be found in Fig. 1.

Considering the temporal aspects of the data and the dynamics of a moving object in aquatic environments, this paper focuses on domain-specific scientific knowledge relevant to underwater object dynamics. Moreover, this paper presents the modification of an existing Unity3D project [18], in order to generate a dataset that is tailored to the scope of this research. Subse-

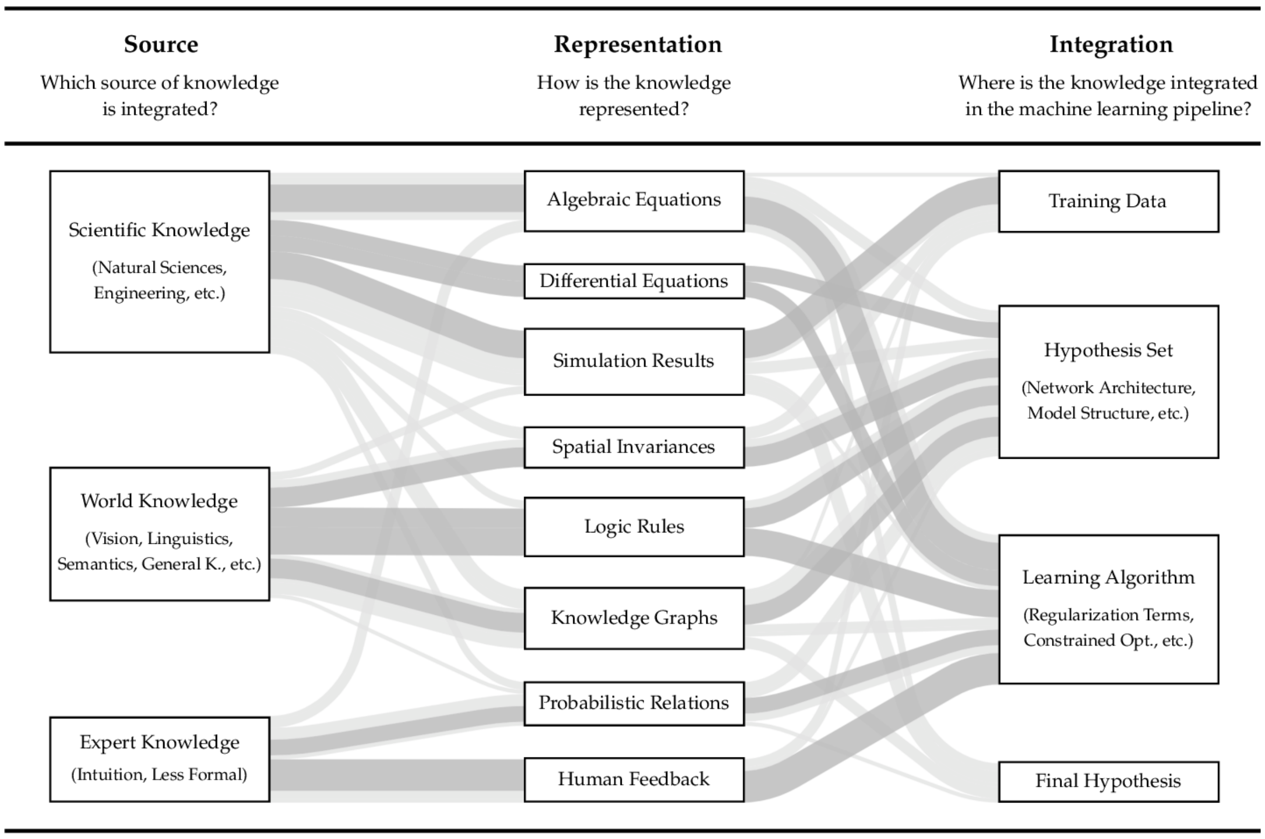

Source
Which source of knowledge is integrated?
Representation
How is the knowledge represented?
Integration
Where is the knowledge integrated in the machine learning pipeline?
Scientific Knowledge
(Natural Sciences, Engineering, etc.)
World Knowledge
(Vision, Linguistics, Semantics, General K., etc.)
Expert Knowledge
(Intuition, Less Formal)
Algebraic Equations
Differential Equations
Simulation Results
Spatial Invariances
Logic Rules
Knowledge Graphs
Probabilistic Relations
Human Feedback
Training Data
Hypothesis Set
(Network Architecture, Model Structure, etc.)
Learning Algorithm
(Regularization Terms, Constrained Opt., etc.)
Final Hypothesis

Fig. 1: Taxonomy of informed machine learning [16]

quently, an algorithm is designed to extract the object trajectories from the synthetic dataset. Two mathematical models representing the movement of the objects of the two considered class categories, i.e., trash and fish are subsequently integrated into a neural network using the principle of knowledge distillation [19]. This enables the neural network to learn from both the dataset and the domain-specific knowledge during the training phase. An analysis compares the performance of the neural network that is trained with and without the integration of domain knowledge across various scenarios and explores the benefits and limitations. Additionally, the training is performed in both fully supervised and semi-supervised settings.

The main contributions of this paper are given below:

- A synthesised Ordinary Differential Equations (ODE)-driven dataset is introduced for underwater object detection.
- A domain-informed neural network framework is developed to address the challenges of underwater object detection with limited and poor-quality data.
- The proposed architecture is trained in both a fully supervised and a semi-supervised learning setting and a comprehensive analysis of the results is presented.

Given the scope of this research, the study is delimited into two primary class categories: trash and fish. The trash class consists of two objects: a plastic bottle and a plastic beer holder. The fish class comprises a single type of fish. The proposed approaches can be generalised and expanded to more classes and different applications, including various types of trash, other marine life or even space applications that use different types of domain knowledge.

This paper is organised into four sections: Section section 2 discusses the related work. The methodology is described in section 3, covering the simulator setup and the generated dataset, the data preprocessing module, the mathematical models and their application as domain knowledge, the dataset customisation module, and the neural network along with the training process. The results, discussed in section 4, are presented in two case studies: the first one, detailed in subsection 4.1, is based on supervised learning, and the second one, discussed in subsection 4.2, is based on semi-supervised learning. The conclusion and topics for future research are given in section 5. A road map can be found in Fig. 2.

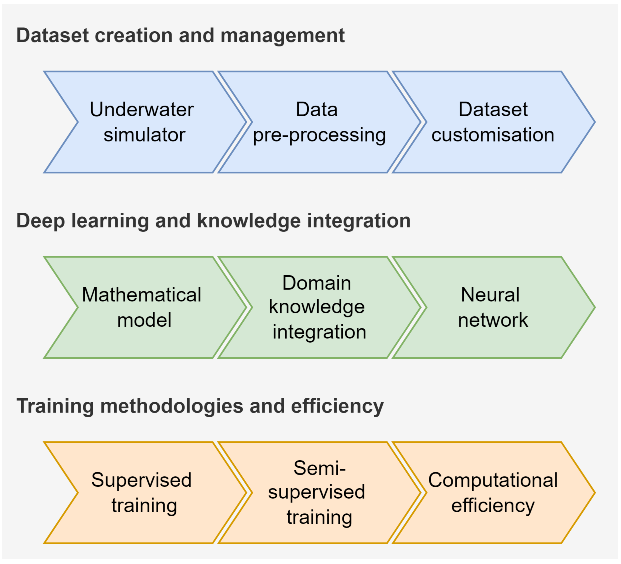


Fig. 2: Road map of the paper

II. Related Work

This section provides an overview of the existing relevant state-of-the-art knowledge on video-based underwater datasets and simulators, motion extraction techniques, neural networks, and domain knowledge integration methods.

II.i Datasets and simulators

Data is paramount in any machine learning problem. In fact, the quality of the dataset that is used in the training and validation of a machine learning method has a major effect on the performance of the resulting model [20, 21].

As this paper focuses on dynamic underwater environments, the machine learning-based classification method requires a sequentially annotated dataset including both plastic and fish. The most known underwater datasets are Deep-sea Debris Database [22], Trashcan [23], and TrashICRA19 [24]. However, these datasets provide annotations at the frame level, instead of the video level, and lack unique identifiers to track the moving objects. The velocity of a moving object can be calculated via:

$$v_x^{\mathrm{obj}} = \frac{x_{i+1}^{\mathrm{obj}} - x_i^{\mathrm{obj}}}{f^{\mathrm{s}}} - \frac{x_{i+1}^{\mathrm{rov}} - x_i^{\mathrm{rov}}}{f^{\mathrm{s}}} \tag{1}$$

with v_x^{obj} the velocity of the moving object in the horizontal direction, x_i^{obj} the location of the object across the horizontal axis, i the frame number, f^{s} the sampling rate in Hz, and x_i^{rov} the location of the Remotely Operated Vehicle (ROV) in the horizontal direction for the i^{th} frame. Note that the locations are assumed to be in a real-world coordinate system, not an image-based coordinate system. It is possible to convert the locations from one reference frame to another, but in the aforementioned datasets, neither the real-world positions nor the sampling rate is present. This makes none of the publicly available datasets usable for this research. It is not deemed feasible to generate a real-life dataset, due to the high cost and time required for setting up and performing the data collection, post-processing the data, and annotating the data.

Given the scarcity of real-world video object detection datasets, synthetic datasets are considered. After a comprehensive search for open-source visually and physically accurate simulators, the following options were identified. Note that since in fully automated detection and collection of underwater waste robots are used, simulators relevant for robotic systems were also included:

- Robot Operating System (ROS) based simulator: This is an open-source simulation tool for robotic systems that encompasses a modular structure and offers a wide range of vehicles and sensors with realistic visualizations [25].
- Gazebo-based simulator: This is a package for the Gazebo framework that supports multiple robots and intervention tasks. It allows a straightforward setup of diverse scenarios and robots [26].
- Unity-based simulator: This is an environment in the Unity landscape that enables the simulation of underwater dynamics. It facilitates the integration of various control algorithms and the training of neural networks thanks to its high visual fidelity [18].

II.ii Motion extraction techniques for images

To leverage the dynamics of the object to make predictions about its movements, the first step is to extract the motion from the gathered data. The apparent motion of the brightness patterns in an image, caused by the relative motion between the object and the camera, is called optical flow [27], which will be used in this paper to extract the position and the velocity of a moving object from images. The goal is to calculate the horizontal and vertical velocity for each pixel in the image in order to describe the movements of the object across the frames.

The most known techniques are based on the brightness consistency assumption [27, 28], which states that the apparent intensity of an object does not change across different frames. Mathematically, this assumption is given by:

$$I(x, y, t) = I(x^{\mathrm{pixel}} + \Delta x^{\mathrm{pixel}}, y^{\mathrm{pixel}} + \Delta y^{\mathrm{pixel}}, t + \Delta t) \tag{2}$$

where $I(\cdot,\cdot,\cdot)$ is the intensity function, and x^{pixel} and y^{pixel} are the locations of the object in pixel coordinates at time instant t in the, respectively, horizontal and vertical directions, and Δx, Δy, and Δt show the increment of these positions and the time. Considering a small motion, the Taylor series expansion can be used to construct the Optical flow equation, given below, where v_x^{pixel} and v_y^{pixel} are the velocities of the pixel in the, respectively, horizontal and vertical directions:

$$\frac{\partial I(x,y,t)}{\partial x} v_x^{\mathrm{pixel}} + \frac{\partial I(x,y,t)}{\partial y} v_y^{\mathrm{pixel}} + \frac{\partial I(x,y,t)}{\partial t} = 0 \tag{3}$$

To solve this equation for the two unknown variables v_x^{pixel} and v_y^{pixel}, another equation that relates these two variables is required. Different approaches have been proposed for this aim: Lucas-Kanade [28] is an approach that stands out for its speed. It leverages spare optical flow and assumes that the neighbouring points belong to the same flow. In contrast to dense optical flow, which considers all the pixels, sparse optical flow only focuses on specific features. A slightly different approach is taken in the Horn-Schunck method, where the difference in the average velocity between the neighbouring pixels is minimised [27]. Lastly, Gunnar-Farneback method utilises the dense optical flow technique, employing polynomials to represent the image and subsequently comparing two sets of these polynomials to determine the flow [29]. Despite being the slowest of the three techniques, the Gunnar-Farneback method stands out as the most accurate.

Besides the traditional methods, Gaussian mixture models [30] have also been applied to the optical flow problem. These models work based on the more general assumption that the information in the image sequence is conserved locally in space and time, in the direction of motion. The corresponding algorithm operates by clustering different layers of motion within the image. Each layer typically refers to a different object such as a car or a cyclist. The mixture model probability equation provides the conditional probability for satisfaction of the motion constraint vectors $\vec{c_k}$ for the location $\vec{x_k} = [x_k, y_k]^\top$ in the image, for the different N

flow fields with parameter vectors $\vec{a_n}$ for $n = 1, \ldots, N$. Note that $\vec{a_n}$ is the vector of all the parameters that describe the n^{th} flow field, where each flow field represents a potential motion layer in the image that captures the velocity vectors of a specific coherent motion within the image region. We have:

$$p\left(\vec{c_k} \mid \vec{x_k}, \vec{m}, \vec{a_1}, \ldots, \vec{a_N}\right) = \sum_{n=0}^{N} m_n \cdot p_n\left(\vec{c_k} \mid \vec{x_k}, \vec{a_n}\right) \tag{4}$$

In other words, (4) gives the probability that a given motion constraint vector or class (e.g. pedestrian) $\vec{c_k}$ at location $\vec{x_k}$ in the image matches the flow field described by the feature parameter vector $\vec{a_n}$ for $n = 1, \ldots, N$. The left side of (4) yields the total probability and the right side is the weighted sum of the component probabilities. Note that m_n is the mixture probability weight with a sum equal to 1. The equation is solved by utilising the Expectation-Maximisation algorithm [31].

The trend in the last few years has been to apply machine learning to improve optical flow accuracy. The first widely known machine learning-based model is FlowNet [32], where two identical models are used for both the input images and a shared correlation layer is used to match them together. The endpoint error, which is a standard metric for optical flow is used as training loss [32]. Despite the innovative nature of FlowNet, its generalizability for different datasets is limited compared to the conventional approaches, such as Lucas-Kanade [28] and Horn-Schunck [27]. FlowNet 2.0 [33] has been built based on the architecture of FlowNet 1.0, where various improvements, such as dataset training scheduling, stacking neural networks, and the addition of a neural network focused on small movements, have been added.

Another architecture for motion extraction from images is the Recurrent All-Pairs Field Transforms (RAFT) [34], which includes a feature encoder and a recurrent update mechanism. A special characteristic of RAFT is maintaining a high-resolution flow throughout the layers of the neural network. This results in considerably less missing data corresponding to small and fast-moving objects, compared to other models.

The primary drawback of using neural networks for motion extraction from images is their requirement for extensive training datasets. Although the performance of neural networks may surpass that of traditional methods, this comes at the cost of losing generalizability. Consequently, these drawbacks make the traditional optical flow methods preferable.

II.iii Neural networks

Deep learning video object detection has seen a surge in popularity over recent years. Instead of applying 2D CNNs to individual video frames, research has shown that both 3D CNNs and transformers achieve higher performance levels [35, 36]. The state-of-the-art neural networks include X3D [37], InternVideo [38], and VideoMAE V2 [39]. Despite their impressive performance, running these models on ROVs is challenging due to their high computational demands. Additionally, these neural networks are notoriously difficult to train, require extensive datasets for training and validation, and possess complex architectures, which makes it difficult to modify the architecture to embed domain-specific knowledge. Considering these challenges, this work focuses on more conventional methods for processing temporal data, including the Feed-Forward Neural Networkss (FFNNs) and Recurrent Neural Networks (RNNs).

A FFNN is a simple architecture that consists of an input layer, one or more hidden layers, and an output layer, where each layer is composed of various neurons. During the training process, the data is fed into the input layer and flows through each subsequent layer. A weighted sum of the inputs is calculated at each neuron and is passed through an activation function, which is responsible for introducing non-linearity in the neural network. At the final layer, the output tensor is resized to the number of classes of objects that should be determined by the neural network. A loss function is used to calculate the difference between the annotations and outputs of the neural network. During the training process, this loss is back-propagated through the model and an optimiser is used to update the weights, such that this loss is minimised throughout the training. The most common optimizers used for this purpose are the stochastic gradient descent methods and the Adam optimiser [40].

Similarly, an RNN deals with a layered structure, requires activation functions, and follows a training process. The main difference between RNNs and FFNNs is the presence of memory blocks, which allow the neural network to remember the previous inputs and to use this information in its predictions. The most common RNNs are the Long Short-Term Memory networks (LSTMs) [41] and the Gated Recurrent Unit networks (GRUs)[42], which address the basic architecture problems of RNNs, i.e., a vanishing or exploding gradient during the training [41, 42]. While RNNs are frequently used in time series analysis, due to their recurrent architecture, the input trajectory in the application of this paper includes a single data point. This facilitates the use of a straightforward FFNN.

II.iv Domain knowledge and deep learning integration methods

Integration of domain knowledge and neural networks is known to, generally, improve the performance, consistency with rules and principles of physics, and generalizability of neural networks [43, 44, 45]. As is given in Fig. 1, each source of knowledge that is used in training the machine learning-based models may have various representations of the information, where this representation subsequently dictates the possible methods of integration of the knowledge. For the application of this paper, the main source of information for the object detection module is the dynamic movements of the target objects that may be represented via differential equations, through numerical or computer-based simulations, or as logical rules. The corresponding information can then be integrated within a neural network via the input data, the loss function, knowledge distillation, the architecture, and the output data [46].

The integration of the domain knowledge via the input data can be done by using a modified version of the existing data for training of the neural network [47]. Another way is to construct a completely new set of data, e.g., using computer-based simulations, as additional labels in training [48]. Moreover, the existing training datasets can be enriched by modifying samples such that they exhibit specific properties, as demonstrated in [49].

Then the domain knowledge can effectively be used to select characteristics, such as rotational and Galilean symmetries, where this helps to create neural network-based models that are more robust to low-quality training sets and aligned with the underlying physical principles of the system they model.

Modifying the loss function is also a popular approach to effectively embed the domain knowledge within the training process of a neural network. The most common approach is to add a knowledge-based term within the formulation of the loss function, where this term, often called the physics-based or hybrid loss, provides information about the quality of the annotations and indicates consistencies. Furthermore, domain knowledge can be integrated into support vector machines and kernel-based approximation methods by adjusting the optimisation problem to include constraints, thereby guiding the learning process within defined regions of the input domain [50]. The different components of the loss function, including the label-based regularisation, and physics-based terms, and the problem formulation for determining $\vec{w}_f^*$ (i.e., an optimal value for the vector $\vec{w}_f$ of the weights of function $f : \mathcal{X} \to \mathcal{Y}$) for the neural network are given below, based on [16]:

$$f\left(\mathcal{X} \mid \vec{w}_f^*\right) = \arg\min_{\vec{w}_f} \left(\overbrace{\lambda^{\mathrm{l}} \sum_i L^{\mathrm{l}}\left(f\left(\xi_i \middle| \vec{w}_f\right), \zeta_i\right)}^{\text{Label-based loss}} + \overbrace{\lambda^{\mathrm{r}} R(\vec{w}_f)}^{\text{Regularisation}} + \underbrace{\lambda^{\mathrm{k}} L^{\mathrm{k}}\left(f\left(\xi_i \middle| \vec{w}_f\right), \zeta_i\right)}_{\text{Knowledge-based loss}} \right) \tag{5}$$

with $\mathcal{X}$ the input dataset and $\mathcal{Y}$ the predictions, λ^{l}, λ^{r}, and λ^{k} the weights for, respectively, the label-based loss $L^{\mathrm{l}}(.,.)$, the regularisation-based term $R(f)$ applied to the model weights, and the knowledge-based loss $L^{\mathrm{k}}(.,.)$. Moreover, $\xi_i \in \mathcal{X}$ and $\zeta_i \in \mathcal{Y}$ with $\left(f\left(\xi_i|\vec{w}_f\right), \zeta_i\right)$ the input-output pair of the neural network.

A method for embedding the domain knowledge into a neural network is knowledge distillation, which involves transferring information from a (typically large) teacher neural-network-based model to a (usually small) student neural-network-based model [19]. During the training process, the student model learns from both the training data and the teacher model. The loss function is designed such that the student model learns to mimic the probabilities of the class predictions of the teacher model. This approach not only helps in compressing the size of the student neural network-based model but also accelerates the inference time (the time it takes for a trained model to make predictions on unseen data). This is specifically advantageous for deployment in resource-constrained situations and environments, e.g., when ROVs are used. Using knowledge distillation, Z. Hu et al. [51] have in particular shown the efficacy of combining deep neural networks with domain knowledge represented via structured logical rules. Their work will serve as a foundational basis for this paper.

Domain knowledge may be integrated into a neural network through its architecture, which was done first in 1994 in [52], where the architecture of the neural network is defined by a hierarchically structured set of rules [53]. A main shortcoming of this approach is that the ability to expand to new applications beyond the predefined rules is limited. Therefore, various improvements have been proposed to this architecture [54, 55]. Recently, the trend has been shifting towards embedding symbolic knowledge, expressed as logical rules, into the architecture of the neural network [56]. The hypothesis set of a neural network is the set of all functions that the neural network chooses one function from in order to generate outputs from inputs. Integrating logical rules into the hypothesis set of a neural network in a probabilistic manner is another approach that, by assigning probabilities to the logical rules, makes these rules soft (i.e., not necessarily true, but probabilistically true) [57, 58].

Finally, domain knowledge may be included in the output data, which typically entails applying algebraic equations to the output of the neural network and filtering out those results that do not obey the rules [59]. These rules can be the laws of physics, but also be guidelines such that the model's output does not violate any legal laws. Besides algebraic equations, this domain knowledge may be included as simulation results that are employed to validate the trained neural network [60, 61].

Knowledge distillation stands out among all these approaches, for several reasons: Firstly, knowledge distillation allows to develop a small, computationally efficient neural network with a similar performance to the larger teacher neural-network-based model. This is especially important for applications that run on resource-limited platforms, including ROVs. Additionally, the distinct nature of the teacher neural-network-based model allows for the use of various existing mathematical models, thereby enabling the incorporation and extension of state-of-the-art methods. Lastly, knowledge distillation leverages the structured knowledge that is embedded in the teacher neural network-based model, and learns from raw data. This effectively bridges any gaps that may exist in the teacher network-based model and enhances the overall robustness of the student neural network to any noise in the training data.

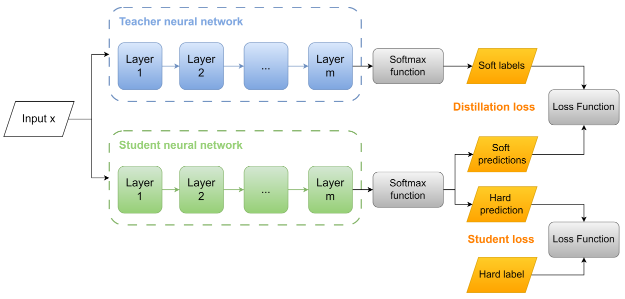


Fig. 3: Schematic of knowledge distillation

III. Methodology

This paper develops a machine learning-based model that extracts the dynamics of motion of underwater waste and fish from a series of underwater images, and that, according to these dynamics, classifies the detected object as either fish or plastic. Accordingly, this section provides a comprehensive outline of the approaches proposed and adopted, starting with the generation and curation of the dataset for training and validation of the model. We then discuss the preprocessing techniques that extract the trajectories of motion, followed by an overview of the mathematical model that captures the domain knowledge, as well as the method for integrating this domain knowledge into a neural network. It is essential to compare the performance of the neural network with and without domain knowledge integration across diverse scenarios. To facilitate this, an advanced setup for dataset generation is introduced and employed. We conclude this section with an in-depth analysis of the architecture of the proposed neural network and its training regimen.

III.i Generation and curation of data

Since no open-source dataset is available that can be used to extract the trajectories of active and passive objects, custom data via a micro-simulator will be generated. The existing simulators that focus on ROVs, lack the option to model plastic and fish in the underwater environment. Thus, custom modifications are needed to provide the desired underwater simulation environment.

First, the accessibility to both the source code of the micro-simulator and the corresponding documentation have been considered necessary. Moreover, to adopt a structured approach to the decision-making, the following list of requirements for the simulator has been established:

1. autonomously generating datasets
2. generating the required datasets in a reasonable time that meets the time requirements
3. generating an annotations file that includes a 3D vector of the position, velocity, and acceleration, as well as the coordinates of the bounding box of the detected objects corresponding to every new

trajectory
4. generating a file including the specific settings of every trajectory (i.e., the strength and direction of the water current, the direction and intensity of light, the intensity of bubbles, the intensity of particles, the transparency given as a quantity that determines the clarity of the water, and the objects in view)
5. modelling the underwater environment accurately, by including the gravity force, the buoyancy, the friction, the linear drag, and the current forces
6. creating an environment with high visual fidelity, including lighting, the water body, bubbles, and various particles
7. randomly spawning a minimum of two different objects
8. allowing to freely choose the degrees of motion of the objects
9. generating diverse scenarios with variations in the direction and intensity of lighting, the strength and direction of the water current, the water transparency, the particles, and the bubbles

An extensive investigation of different simulators brought us to the Unity Game Engine (briefly called Unity) [62], which emerges as the most suitable choice for fulfilling all the abovementioned requirements. In addition to its versatility and comprehensive documentation, which distinguish Unity from other potential simulators, Unity has shown to provide a robust setup that is capable of generating high-quality data that is flexible to be customised for object detection [18]. Next, we explain the overall architecture of the simulator that has been developed in Unity to simulate the underwater scenarios for this paper. This architecture is also represented in Fig. 4 and the id's of the block are referenced in the section below.

First, Unity simulates an ROV with a stationary camera, focusing on the underwater object avoidance. A Python module interacts with Unity in the baseline simulator and gathers data. Given that this paper is focused on detection of moving objects, rather than on the movements of the ROV itself, the baseline simulator has been extended with a custom C# module for data gathering. Since C# provides direct Unity integration, it is a more suited option than Python. The C# module autonomously produces a dataset, where the desired number of trajectories for the moving objects is provided as input to the module. The scene of the scenario and the various plugins (including Aura 2 and custom scripts) are initialised as soon as the Unity editor is launched (block 1). This then triggers the data gathering script, which generates the required number of trajectories in a loop with several iterations, where during each iteration various steps, as will be explained below, are executed (block 2).

A custom-built module, called the variable tracker (block 3.a), tracks the states of the moving objects. Two plastic objects are present, a plastic bottle, a plastic beer holder and one fish. The position, velocity in the horizontal and vertical directions and acceleration in the horizontal and vertical directions are tracked. For the environment are fog densities, bubble and particle emission and the current velocity tracked. The data generation procedure ensures a diverse dataset, by generating iterations with varying parameters for the scene (these parameters include the underwater fog strength, the particle emission rate, the intensity and position of the light, and the direction and strength of the water current) and for the moving objects (these parameters include the mass, volume, initial position and orientation), but keeping the water density, ground textures, and dimensions of the scene constant (block 3.b). Each iteration is characterised by the initialisation of the variable tracker, by generation of a random scene, and by spawning one randomly chosen object from the input list of objects (which includes fish, plastic bottle or plastic beer holder) (block 3.c).

Note that the underwater fog strength is a parameter that determines the water transparency, where a large value for this parameter indicates low transparency, which results in low visibility and affects the maximum distance that an object can be placed from the camera, before becoming undetectable for it. The particle emission rate introduces particles (called the marine snow) in the scene that impact the clarity of the images taken by the ROV. The bubble emission rate determines the amount of bubbles that are spawned in the simulation environment every second. Including these bubbles in the simulation enhances the visual fidelity of the scene. Similarly, the intensity and position of the light significantly impact the visual fidelity of the scenes, where by avoiding extreme lighting conditions more realistic scenarios are simulated. Finally, the direction and strength of the water current influence the direction and velocity of the movement of the objects. The ranges of all these parameters have been carefully chosen for the simulations to reflect realistic underwater conditions (see the details in Appendix A). At the end, by adding ocean sand and rock formations at the bottom of the scenes, the visual fidelity of the simulator has been further improved (see Fig. 5).

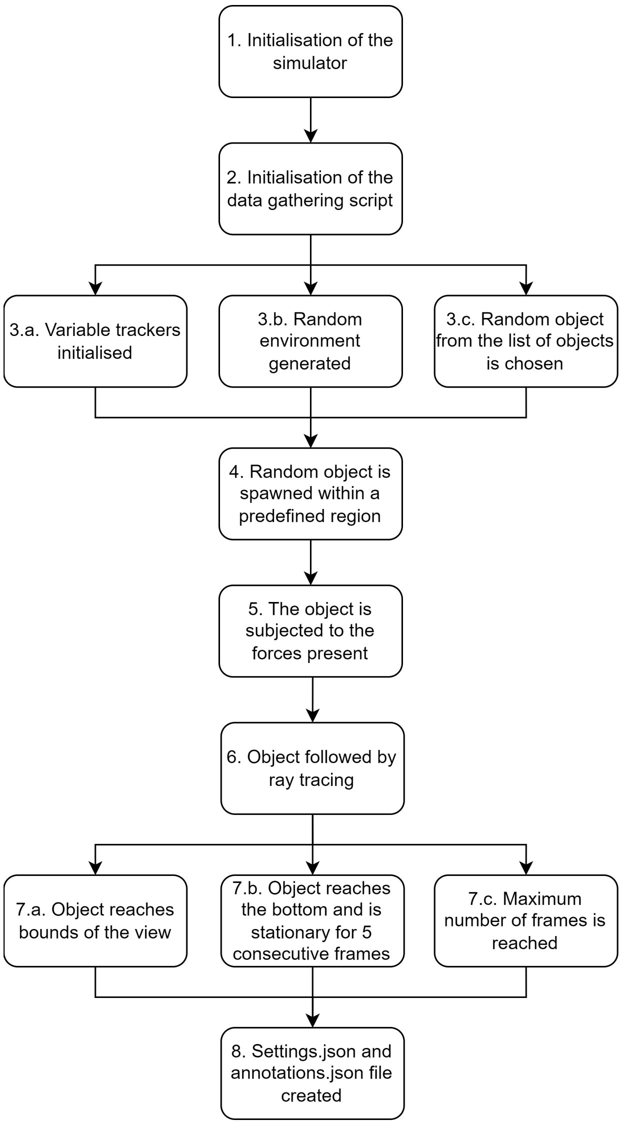
The last step in the initialisation is the spawning of either a plastic bottle, plastic six-pack holder, or a fish, where these have been illustrated in, respectively, Fig. 6a, Fig. 6b, and Fig. 6c as they appear in Unity. These objects are not only characterised by their mesh but also by their various properties, including the mass m^o, the volume V^o, the surface areas A_x^o, A_y^o, A_z^o and the drag coefficients $c_x^{\mathrm{d},o}$, $c_y^{\mathrm{d},o}$, $c_z^{\mathrm{d},o}$ in the 3 dimensions, and the density ρ^o, with $o \in \{\text{fish, bottle, holder}\}$. Moreover, for the fish, the fish force F_z^{fish} is also considered and consists of a sinusoidal force used to simulate the thrust generated by the tail. Per iteration, the mass and the volume of the plastic bottles, plastic six-pack holders, and the fish vary within a range of $\pm 10\%$ from their default values. The values for all the parameters used in the simulations are given in Appendix A.

During and across the iterations, the camera of the ROV is kept stationary at position $[0, 0.5, 0]^\top$, where the first, second, and third elements of the given coordinate correspond to, respectively, the horizontal, vertical axis, and depth directions. The camera of the ROV has a vertical field of view of 60 degrees and uses perspective projection. While the simulator is capable of generating 3D trajectories, in order to reduce the complexity, the movements of the object are restricted along the depth axis (negative x-axis). Consequently, the object moves within a 2D plane relative to the camera.

A randomisation function chooses which object is selected for the iteration (block 4). Although the movement of the object along the depth has been restricted, a different static value is selected per iteration where this value directly corresponds to the distance from the camera that the object is spawned. The upper bound of the position where the object is spawned across the horizontal axis is determined by the fog density ρ^{fog}. Given that the relationship between the fog density and the maximum distance from the camera that an object is still detectable for the camera is non-linear, the following polynomial regression is used to determine the bound for the depth coordinate of an object:

$$\begin{aligned} x^{\text{obj,max}} = 4 \times 10^{-6} \cdot \left(\rho^{\text{fog}}\right)^3 - 0.0011 \cdot \left(\rho^{\text{fog}}\right)^2 \\ +0.1048 \cdot \rho^{\text{fog}} - 5.0222 \end{aligned} \tag{6}$$

After determining the bound of the horizontal coordinate, the bounds of the vertical and depth coordinates will be calibrated. For this, the method `ViewportToWorldPoint` [63] from Unity is used to reshape the bounds. Additionally, the direction of the


1. Initialisation of the simulator
2. Initialisation of the data gathering script
3.a. Variable trackers initialised
3.b. Random environment generated
3.c. Random object from the list of objects is chosen
4. Random object is spawned within a predefined region
5. The object is subjected to the forces present
6. Object followed by ray tracing
7.a. Object reaches bounds of the view
7.b. Object reaches the bottom and is stationary for 5 consecutive frames
7.c. Maximum number of frames is reached
8. Settings.json and annotations.json file created
Fig. 4: High-level architecture of the steps within the Unity simulator for generating the underwater environment for data collection and for running our experiments

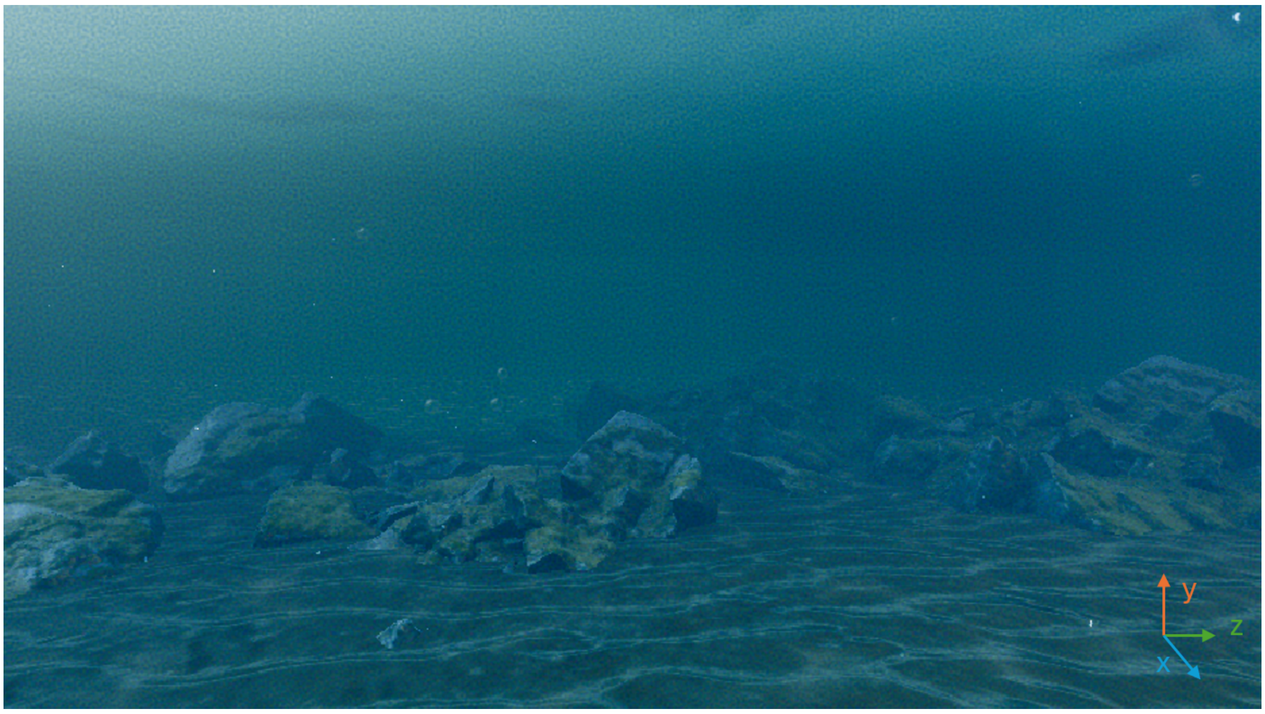


Fig. 5: Default scene of the underwater environment generated by Unity

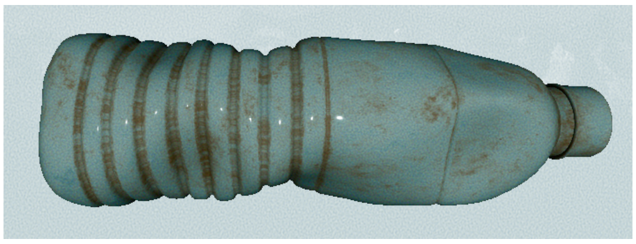
water current is considered to restrict the spawning location to either the left or right side of the horizontal plane. This consideration, along with the implementation of buffer zones, constitute the measures taken to prevent the object from being spawned at the periphery of the visible area and to ensure that the object does not immediately exit the field of view of the ROV.

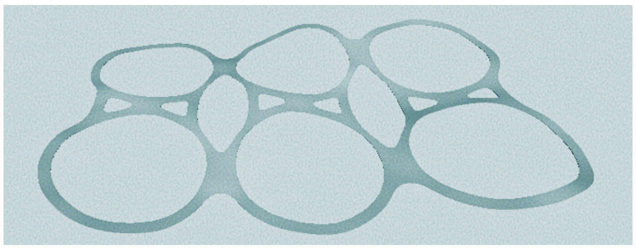
The object spawns in the scene after the initialisation steps are all complete. The Unity engine operates on a continuous cycle wherein the object states are updated, the inputs are processed, the calculations are performed, and the graphics are rendered. Each C# class within the simulation has two default methods: `Update()` and `FixedUpdate()`. The `Update()` method is invoked at every frame, but the frame rate varies with the performance of the hardware. To ensure that the calculations are executed consistently, the `FixedUpdate()` method is employed. While the Unity Physics engine applies by default the forces, such as the gravity and the collisions, more specific forces including the buoyancy and the water current require custom implementation, which has been discussed in detail in Appendix A (block 5).

Finally, an iteration is terminated when one of the following three exit criteria is met:

1. The object has moved out of the view of the camera (block 7.a).
2. The velocity in the horizontal and vertical axis of the object has dropped under the threshold of moving at least 0.02 m between any two frames (block 7.b).
3. The maximum number of frames is reached (block 7.c)

Upon completion of each iteration, the following two files are generated: *settings.json* and *annotations.json* (block 8). The file *settings.json* contains a comprehensive record of the settings that have been employed in the simulation for the completed iteration. The file *annotations.json* is generated using a custom-built annotation module, which casts rays across a designated area towards the centre of gravity of the object. Each ray outputs a boolean that is true if the object is collided by the ray. An array is then created from all the rays that collide with the object and is then translated into pixel locations, facilitating the automated generation of the bounding boxes (block 6). A sketch of this approach can be found in Fig. 7. While the rays may collide with the object in two senses, i.e., box collision


(a) Plastic bottle 3D model [64]


(b) Plastic beer holder 3D model

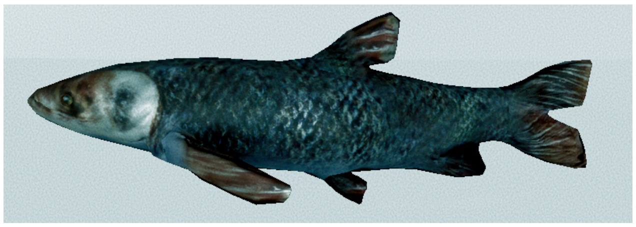

(c) Fish 3D model [65]

Fig. 6: Objects used in the simulator

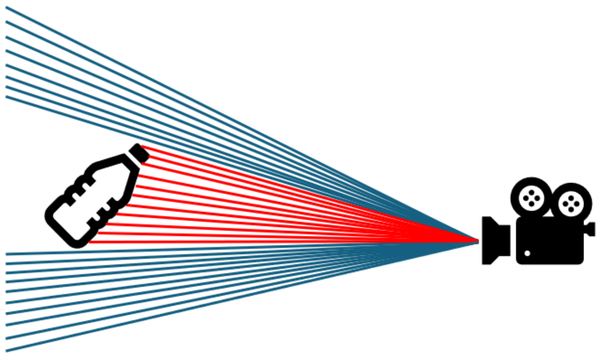


Fig. 7: Sketch of bounding box creation module. The camera projects 200 rays in both the horizontal and vertical directions. Note that the sketch only shows the 2D case.

or mesh collision, we consider the box collision (which compromises some levels of accuracy to achieve computational efficiency).

All the aforementioned steps are repeated per iteration until the desired number of trajectories of movement are generated.

III.ii Data preprocessing

As explained earlier, the simulator produces a set of trajectories for the movement of the objects, where each trajectory corresponds to a set of sequential images, a configuration file, and an annotation file. Although both the bounding boxes and the 3D positions of an object are always known to the simulator, in the real world, only the images with metadata are present. Consequently, in this section, we discuss how the patterns of movement captured from sequential images are leveraged to accurately infer and extract the real-world positions of the objects from the simulated data.

The motion extraction from sequential images is conducted through optical flow, specifically utilising the Farneback method [29]. This process is implemented using the OpenCV library [66], a widely recognised package for computer vision tasks. The input consists of two sequential images that are likely to include a moving object, and the output is a velocity magnitude plot for the object. Suppose that image 1 corresponds to time instant t_1 with the object positioned at (x_{t_1}, y_{t_1}), expressed in pixel coordinates, and image 2 corresponds to time instant t_2 with the object positioned at (x_{t_2}, y_{t_2}). Each image consists of multiple items with distinct optical flows, including the object of interest, bubbles, particles, waves, and present turbulence. If the actual velocity of the object through the water exceeds the velocity of, for example, the bubbles or the waves, then the optical flow values for the object will be larger than those for the bubbles or the waves. Note that the bubbles, partial waves, or any turbulence will be referred to as the "surroundings" of the object. Given the average frame rate of 10 fps, a small, linear motion for the object in between any two consecutive frames is assumed. We also assume that the velocity of the object exceeds that of the surroundings, thus the largest optical flow will be attributed to the object. Using these assumptions, the position of the moving object in pixel locations at time instant $\bar{t} = \frac{t_2 - t_1}{2}$ is approximated by:

$$(x_{\bar{t}}^{pix}, y_{\bar{t}}^{pix}) = \left(\frac{x_{t_2}^{pix} - x_{t_1}^{pix}}{2}, \frac{y_{t_2}^{pix} - y_{t_1}^{pix}}{2} \right) \quad (7)$$

From Fig. 8, the object silhouette is not uniformly

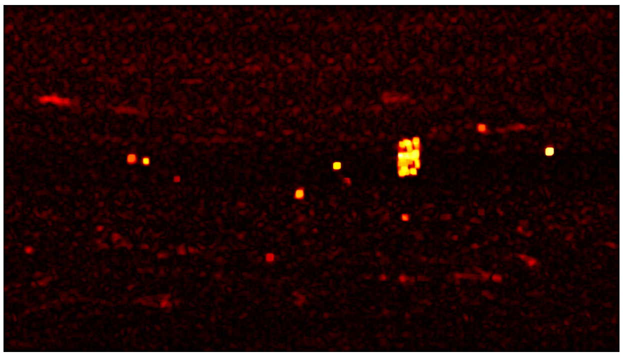


Fig. 8: Output mask of the Farneback optical flow applied to two sequential underwater images of a plastic beer holder. The more intense the colour of the pixel, the larger the optical flow is for that location.

detected, and noise is present in the mask due to the dynamic surroundings. To accurately locate the centroid of the object, a k-means clustering algorithm [67] is applied twice. Firstly, the clustering algorithm is applied to the magnitude of the optical flow values. The algorithm is designed to extract three magnitude-based clusters, from which the background (any pixels not belonging to the object) is identified if it covers more than 20% of the image. This can be either one cluster or two clusters whenever a high level of noise is present (see Appendix B for further details). Subsequently, the resulting cluster or clusters undergo a second iteration of k-means clustering. This time the emphasis is on the spatial positioning of the highest optical flow values. The objective of this phase is to determine those segments of the previously identified cluster(s) that correspond to the actual object, given that the magnitude clustering is also susceptible to noise interference, due to optical flow produced by the surroundings. Ultimately, this process isolates a maximum of six clusters (i.e., in case three magnitude and three colour clusters are identified), and the centroids are calculated per cluster. These centroids are then assessed for their proximity to each other and the clusters with centroids closer than a given threshold are merged. Finally, the cluster of the most substantial size is presumed to represent the object of interest. The centre of the resulting cluster is then extracted and expressed in the local reference frame of the image.

The motion extraction module expresses the estimated location of the moving object in pixel coordinates. In order to use differential equations corresponding to the domain knowledge, these pixel coordinates should be transformed from a local 2D reference frame into a 3D global real-world frame of reference. The simulation uses the projection view [68] to generate the footage. We have:

$$y^{\text{obj}} = \left(\left(\frac{y^{\text{pix}} - y^{\text{centre}}}{f} \right) d^{\text{cam}} - y_0 \right) \tag{8}$$

where

$$f = \frac{h^{\text{pix}}}{2 \tan \left(\frac{\text{fov}^{\text{v}}}{2} \right)} \tag{9}$$

The projection view formula to convert the y coordinate of a local reference frame in pixels to a global reference frame in meters is given by (8) (note that the same formula is used for the x-coordinate), where y^{pix} is the given y-coordinate of the moving object in pixels and y^{centre} is the y-coordinate of the centre of the image in pixels, which equals 540 pixels, as the height of the image is 1080 pixels. Moreover, d^{cam} is the distance of the object from the camera in meters that is assumed to be known. Additionally, y_0 is the offset of the coordinates of the camera that equals 0.5 m in this setup, and f is the focal length. The simulator does not provide the focal value, but this value is calculated using (9), taken from [69]. The formula uses the resolution height h^{pix} and the vertical field of view fov^{v}, which equals 60 degrees in our setup.

In order to enhance the computational efficiency, the method is militarised to enable the motion extraction and the clustering processes to be executed in parallel across the entire dataset of trajectories. These results, settings, and annotations are consolidated into a single .json file, which eliminates the need for interfacing with various files. In addition to the simulation and the optical flow-derived trajectories, to mimic the sensor noise that is typically present in real-world data, additive Gaussian noise of mean 0 and a standard deviation of 0.4 is included in the ideal trajectories. The unmodified trajectories are referred to as "ideal" trajectories.

In summary, the `parsed_data.json` file is compiled utilising the data preprocessing module to combine all the data into a single file. Furthermore, validations are implemented to ensure the data integrity and to verify compliance with the minimum length criteria of 9 frames.

III.iii Mathematical models

Given the custom data and the feasibility of modelling the underwater domain with mathematical expressions, it is chosen to represent the domain knowledge using differential equations. A mathematical model is developed per class of objects, where the focus of the modelling is on the underwater movement, while partial submersion is outside of the scope.

The mathematical model for the class of plastic is a set of differential equations describing the 2D position and velocity of a passive object with specific properties in an underwater environment. The state vector is $\mathbf{x} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \end{bmatrix}^T$ with $s_1 = y^o$, $s_2 = \dot{y}^o$, $s_3 = z^o$ and $s_4 = \dot{z}^o$ with $o \in \{\text{fish, bottle, holder}\}$, and the nonlinear system of equations of motion of the object is given by:

$$\begin{aligned}
\dot{s}_1 &= s_2,\\
\dot{s}_2 &= \frac{1}{m^o}\left(-\overbrace{m^o\cdot g}^{\text{Gravity}} + \overbrace{\rho^{\text{water}}\cdot g\cdot V^o}^{\text{Buoyancy}} \underbrace{-\frac{1}{2}\cdot \operatorname{sgn}(s_2)c_y^{\text{d},o}\cdot\rho^{\text{water}}\cdot A_y^o\cdot {s_2}^2}_{\text{Drag}}\right),\\
\dot{s}_3 &= s_4,\\
\dot{s}_4 &= \frac{1}{m^o}\left(-\frac{1}{2}\cdot\operatorname{sgn}(s_4 - v_z^{\text{current}})\cdot c_z^{\text{d},o}\cdot\rho^{\text{water}}\cdot A_z^o \cdot(s_4 - v_z^{\text{current}})^2\right)
\end{aligned} \tag{10}$$

The static parameters include the gravitational acceleration g, the density of water ρ^{water}, the drag coefficient in the y direction $c_y^{\text{d},o}$ and the drag coefficient in the z direction $c_z^{\text{d},o}$ and are assumed to be known. The other parameters are volume V^o of the object, the drag coefficients $c_y^{\text{d},o}$ and $c_z^{\text{d},o}$, the surface areas A_y^o and A_z^o with $o \in \{\text{fish, bottle, holder}\}$.

The state vector and model of the movements of a fish are derived similarly as for the plastic, since they are subject to similar forces. For the fish, however, an extra force that represents the fish propulsion force is included (the differences between the plastic model are indicated in bold). We have:

$$\begin{aligned}
\dot{s}_1 &= s_2,\\
\dot{s}_2 &= \frac{1}{m^o}\left(-\overbrace{m\cdot g}^{\text{Gravity}} + \overbrace{\rho^{\text{water}}\cdot g\cdot V^o}^{\text{Buoyancy}} \underbrace{-\operatorname{sgn}(s_2)\frac{1}{2}\cdot c_y^{\text{d},o}\cdot\rho^{\text{water}}\cdot A_y^o\cdot {s_2}^2}_{\text{Drag}}\right),\\
\dot{s}_3 &= s_4,\\
\dot{s}_4 &= \frac{1}{m^o}\left(\overbrace{\mathbf{F^{fz}}\cdot\mathbf{u_2}+\mathbf{F^{fz}}}^{\text{Fish force}} - \operatorname{sgn}(s_4 - v_z^{\text{current}})\cdot\frac{1}{2}\cdot c_z^{\text{d},o} \cdot\rho^{\text{water}}\cdot A_z^o\cdot(s_4 - v_z^{\text{current}})^2\right)\\
\mathbf{u_2} &= \sin(\mathbf{t}\cdot\mathbf{f}^{\text{tail}})
\end{aligned} \tag{11}$$

Specifically, the equation for $\dot{z}$ incorporates a "fish force" F^{fz} and a tail frequency f^{tail} . More details are given in Appendix C.

From both (10) and (11), solving the differential equations requires the values for the following parameters: the volume V^o of the object, the drag coefficients $c_y^{\text{d},o}$ and $c_z^{\text{d},o}$, the surface areas A_y^o and A_z^o, and the fish force F^{f_z} with $o \in \{\text{fish, bottle, holder}\}$. The exact values for these parameters are known within the simulator, but determining these parameters in the real world is very hard, and model identification is used to estimate those parameters. Thus, we also performed model identification based on an initialisation of the parameters. Numerical integration is then performed to simulate a trajectory using the backward differentiation formulation and a quasi-constant step scheme [70] of 0.1 seconds. These parameters are identified to account for the discrepancies between the observed trajectories and the simulated ones, that occur due to small differences between the mathematical models and the simulation. A minimisation algorithm is employed to iteratively update the parameters until the desired conversion rate is obtained.

The system identification module is set up such that the volume V^o of the object is estimated using the system identification module, while other parameters are derived from the simulation. Note that the volume of the object is chosen to be identified, due to its significant impact on the buoyancy, which affects the differential equations governing the dynamics of the vertical motions of the object.

Both models (10) and (11) have been validated using data collected from simulations. The results of the validation are detailed in Appendix C.

III.iv Domain knowledge integration

The concept of knowledge distillation [51] is used to integrate the domain knowledge (provided via the mathematical models (10) and (11)) during the training phase of the neural network. Knowledge distillation is typically applied to scenarios where a small-sized neural network should replace a large-sized neural network with similar performance. The smaller neural network is called the student and the large neural network is called the teacher. During the training process, the student not only learns from the data, but also from the teacher, by calculating the loss consisting of a weighted combination of the cross-entropy (CE) loss, $L^{\text{ce}}(\cdot)$, and the Kullback-Leibler (KL) divergence loss, $L^{\text{kl}}(\cdot)$, where the corresponding formulas are given below:

$$L^{\text{ce}}\left(P^* \mid P\right) = -\sum_{i=0}^{2} P_i^* \cdot \log P_i \tag{12}$$

$$L^{\text{kl}}(P^{\text{soft}} \mid Q^{\text{soft}}) = \sum_{i=0}^{2} P_i^{\text{soft}} \log \frac{P_i^{\text{soft}}}{Q_i^{\text{soft}}} \tag{13}$$

where P^* represents the true probability distribution for the class labels, P denotes the probability distribution predicted by the neural network, and Q is the predicted probability distribution determined by the teacher model. Moreover, P_i^*, P_i, P_i^{soft}, and Q_i^{soft} specify the probabilities of the i-th class in the true distribution, the predicted distribution, the softened predicted distribution, and the softened target distribution, respectively. Finally, P^{soft} equals P/T with T called the temperature parameter, a hyperparameter that controls the softness of the probability distribution of the model's logits, making the distribution more uniform as T increases and more peaked as T decreases. A value larger than 1 for T softens the distribution, by spreading the probabilities more evenly across the different classes.

The teacher model in our proposed training architecture is represented by the domain knowledge module. The domain knowledge module employs the mathematical models (10) and (11)) to generate the ideal trajectories of movement per class, based on the initial conditions given for these input trajectories, including the initial positions and velocities, and the object's parameters. Each of these trajectories is then compared to the input trajectory that is generated by the simulator and is assessed based on their similarities. A variety of similarity metrics for the trajectories exist in literature, where the following metrics have been considered in this paper:

- Pearson correlation [71]
- Spearman correlation [72]
- Discrete Hausdorff distance [73]
- Mean average distance (MAD)
- Discrete Frechet distance (DFD) [74]
- Dynamic time warping (DTW) [75]
- Mean Squared Error (MSE)

Each of these metrics is assessed based on two criteria: accuracy and the alignment between the expected and the observed probability distribution. The accuracy reflects the effectiveness of the metric in predicting the correct class. The probability distribution pertains to the relative difference between the two classes. The probabilities must accurately reflect the match between the input trajectory and the expected trajectory generated by the mathematical models. The input trajectory can be one of three different types:

- **Ideal:** The trajectory coordinates are directly obtained from the simulator.
- **Noisy:** The trajectory coordinates from the simulator are used but are multiplied by Gaussian noise.
- **Optical Flow:** The trajectories are extracted using the motion extraction module.

The probabilities are expected to be centred around 0.5 if the trajectory data is of a low quality. In contrast, for high-quality trajectory data, the probabilities are expected to be more widely distributed towards the extremes. It is essential that these probabilities accurately represent the expected distribution to ensure that the maximum amount of information can be encapsulated within the neural network.

The accuracy is constructed by calculating the trajectory similarity between the input trajectory and by the mathematical models for y^o, z^o, v_y^o, v_z^o with $o \in$ {fish, bottle, holder} and by determining the weighted sum of all these similarities. Depending on the metric considered, the largest or smallest value of the metric indicates the highest similarity (for instance the higher the mean squared error, the lower the similarity, while a higher Pearson coefficient indicates a higher similarity). The class with the highest probability is taken as the prediction for the domain knowledge module and is compared with the ground truth to compute the accuracy.

This accuracy is evaluated across different data types. It is found that the mean squared error yields the highest accuracy. The comparison is illustrated in Fig. 9, with additional details available in Appendix D. The figure is constructed by testing each of the metrics on the full dataset of 2500 samples. The anticipated trend is observed, with optical flow data proving to be the most challenging, followed by Gaussian noise and ideal data being the easiest.

For optical flow, which is suited for data of low quality, the probabilities are expected to be centred around 0.5. In the case of Gaussian noise, the distribution is expected to be more uniform, with a higher concentration of values at the extremes. For ideal data, this trend is anticipated to be even more pronounced. In Fig. 9 the probability distributions for each of the metrics for the optical flow case are presented. It shows that the MSE adheres to the expected distribution, while the mean absolute distance (MAD) places more values at the extremes. The other metrics also generally follow the expected distribution pattern. A different distribution is observed in Fig. 11. As desired, the probabilities for the MSE and MAD metrics show a greater presence at the extremes. This pattern is not evident in the rest of the similarity metrics. Based on these observations, it is concluded that the MSE is the best similarity metric, thanks to its high accuracy and the alignment of its probability distribution with the desired trend.

The output of the domain knowledge module is aimed to have a similar behaviour as the logits of the typical teacher model. The softmax function, given below, is then used to convert the raw logits to probabilities:

$$\sigma(\vec{z})_i = \frac{\exp(h_i)}{\sum_{j=1}^{K} e^{h_j}} \tag{14}$$

with $\vec{z} = [h_i, h_2, ..., h_K]$ the input vector of logits, $\vec{z}_i$ the output of the softmax function for the i-th element in the input vector $\vec{z}$, K the number of classes and h_i the i-th element of the input vector. The logits are divided by the temperature parameter before the softmax operation to soften the final probabilities. The total loss function becomes a weighted sum of the CE loss L^{ce} and the KL divergence loss L^{kl} given by:

$$\begin{aligned} &L^{\mathrm{tot}}(P^*, P, P^{\mathrm{soft}}, Q^{\mathrm{soft}}) = \\ &\quad (1-\alpha)\, L^{\mathrm{ce}}(P^* \mid P) + \alpha T^2 L^{\mathrm{kl}}(P^{\mathrm{soft}} \mid Q^{\mathrm{soft}}) \end{aligned} \tag{15}$$

with α the parameter that determines the impact of both the CE loss and the KL divergence loss on the total loss. The KL divergence loss is multiplied by T^2 to ensure that the gradients remain appropriately scaled, maintaining the balance between the contributions of the CE and KL divergence losses [19]. The temperature parameter T and the balancing parameter α are two hyperparameters that are tuned to obtain the highest accuracy. The knowledge distillation approach allows the domain knowledge module to effectively regularise the training, by leveraging its information on the quality of each trajectory.

III.v Dataset customisation

Data constitutes the fundamental basis for machine learning problems. The generation of a sufficiently large dataset usually involves challenges due to the significant time and financial resources required. Typically, a dataset is divided into training, validation, and testing subsets, with a distribution of 70%, 20%, and 10%, respectively. It is important that this distribution is executed cautiously; otherwise, the derived metrics may inaccurately reflect the performance of the neural network. In some applications, the available data may be insufficient to ensure the representation of all possible scenarios across each data split. Investigating how an architecture deals with these suboptimal datasets can be a tedious analysis. A data split generation module is designed for this paper to facilitate this analysis.

The dataset generator module uses a dataset configuration file as input and the path to the *parsed_data.json* file described earlier in subsection 3.1 Besides choosing the number of samples for the training, validation, and test set, the following steps are important for ensuring that the evaluation process is thorough and robust against any noise present in the subsets:

- Utilisation of k-fold cross-validation[76, 77].
- Creation of multiple datasets, each with a reduction in the number of training samples.
- Generation of datasets based on specific criteria.

The first step involves k-fold cross-validation, which decreases the bias in the model's results on the test and validation sets, and prevents overfitting. The custom-written k-fold cross-validation module works by allo-

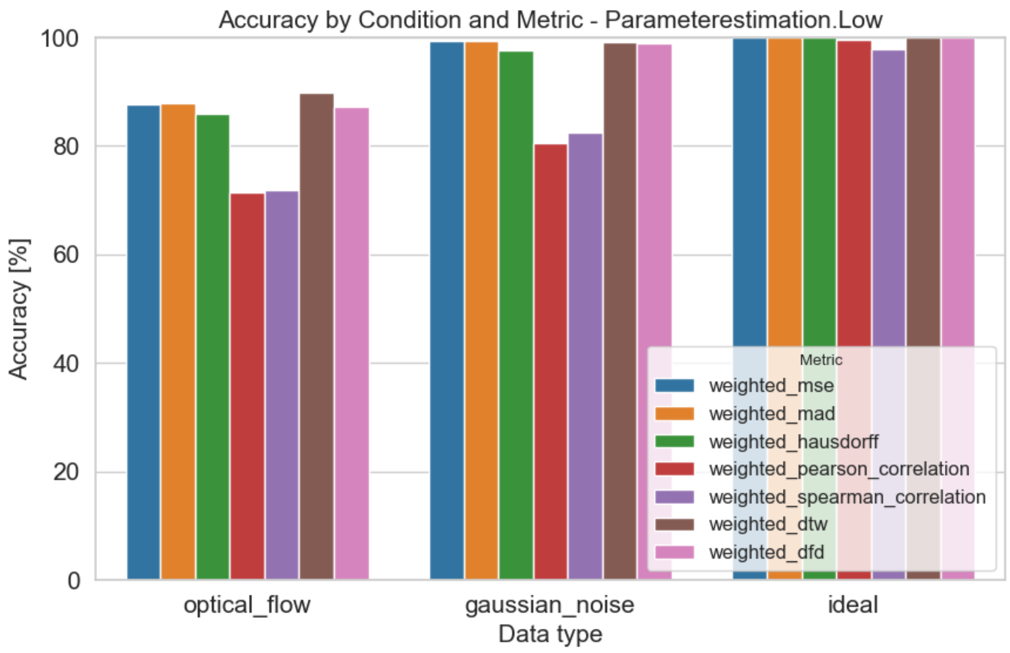


Fig. 9: Comparison between all the similarity metrics for the three different data types (ideal refers to the datasets generated via the simulator, Gaussian noise is the ideal data augmented with Gaussian noise and optical flow is the trajectory extracted by the motion extraction module).

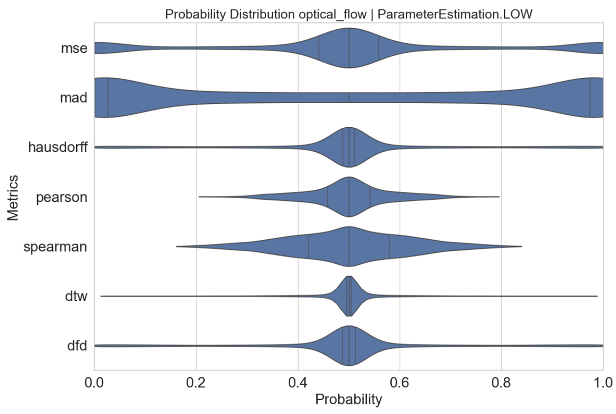


Fig. 10: Probability distribution for all the similarity metrics for the optical flow data. All the metrics are weighted and abbreviated for plotting.

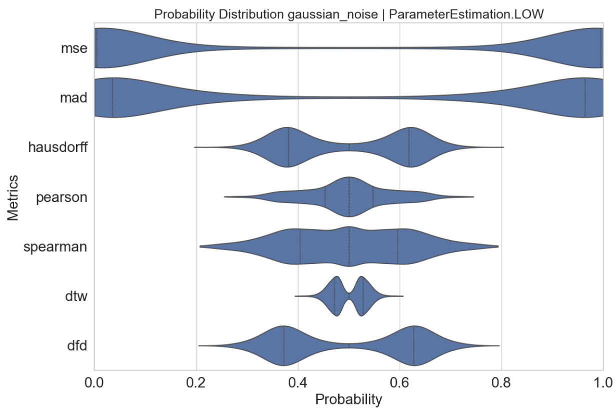


Fig. 11: Probability distribution for all the similarity metrics for the Gaussian noise data. All the metrics are weighted and are abbreviated for plotting.

cating a specific number of samples to each k-fold, ensuring that this number is a multiple of the number of samples in the training, validation, and test subsets. Firstly, the algorithm extracts the required data points for each data split. Subsequently, each of the data points is divided into a specific k-fold. Depending on the number of k-folds, the assigned k-folds are shuffled between the data splits. For instance, in one configuration, folds 1, 2, and 3 might be allocated to the training set, folds 4, 5, and 6 to the validation set, and folds 7, 8, and 9 to the test set. A subsequent distribution could assign folds 3, 7, and 9 to training, folds 1, 4, and 8 to validation, and folds 2, 5, and 6 to testing. This approach ensures that the trajectories within each fold remain consistent, while the folds themselves are shuffled across the data splits. In the end, the variability and robustness against noise in the data of the evaluation process is increased.

The second step entails generating multiple datasets by progressively increasing the number of training samples. Each training file is created using a mapping between the data samples and their respective data split. Each iteration increases the number of data samples in the training set by a fixed number. This approach makes investigating the impact of the dataset size on the results a trivial task.

The last step involves filtering the datasets based on specific properties, such as keeping the current below a certain threshold. Using this approach, the training and validation datasets will exclusively consist of data with lower current values, while the test dataset comprises only high current values. This stratification allows for the evaluation of the generalizability of the model. Note that if the property-based filtering is applied, the k-fold methodology cannot be utilised due to the inherent contradiction that a single trajectory cannot simultaneously possess both high and low current values.

III.vi Neural network and training

The neural network can be trained both with and without the knowledge distillation approach, by switching between learning from both data and domain knowledge to learning only from data.

The goal of the neural network is to classify the extracted trajectory of the target objects. A simple feed-forward neural network is chosen with one input layer, two hidden layers, and an output layer. The hidden layers are fully connected and each is followed by a Rectified Linear Unit (ReLU) activation function, which introduces non-linearity into the neural network and facilitates the ability of the neural network to learn more complex functions. The input layer is configured with a fixed size of 1000 neurons, and the first and second hidden layers include respectively, 128 and 64 neurons. Reducing the number of neurons across the layers is to distil the most salient features of the data as it progresses throughout the neural network. Finally, the output layer maps the last hidden layer to the different classes of objects (see Fig. 12).

A custom dataset class is used to parse and iterate through the data to ensure a streamlined training process. The training data for the neural network consists of an input trajectory transformed into a tensor of the format $[y^o, z^o, v_y^o, v_z^o, 0, 0, ...]$. Zero padding is utilised to meet the fixed input requirement with variable trajectory lengths. During training without domain knowledge, α (see (15)) is equal to zero and training relies solely on the CE loss function. The Adam optimiser [78] is used to solve the optimisation problem (15). If the loss value increases for 5 consecutive epochs, the training procedure is terminated due to overfitting.

When domain knowledge is included in the training procedure, both the trajectory of the detected object and the settings used to generate the trajectory (such as the distance of the object from the camera) are fed into the domain knowledge module, which then calculates the dynamics of motion based on the mathematical model for both classes, plastic and fish. The estimated dynamics are then compared with the input trajectory, and the logits are calculated and undergo a softening process, being divided by a temperature parameter. Utilising these adjusted logits, the difference between the probability distributions of the neural network and of the mathematical model is quantified using the KL divergence loss. The total loss is then back-propagated through the model in order to find the gradients and to optimise the weights of the neural network.

At the end of each training epoch, the performance of the neural network is evaluated using the validation dataset. The highest accuracy is tracked throughout the training process. The final step is to apply the trained neural network that shows the highest accuracy for the validation set to the test dataset. A schematic overview of the entire process is illustrated in Fig. 13. More information is given in Appendix E.

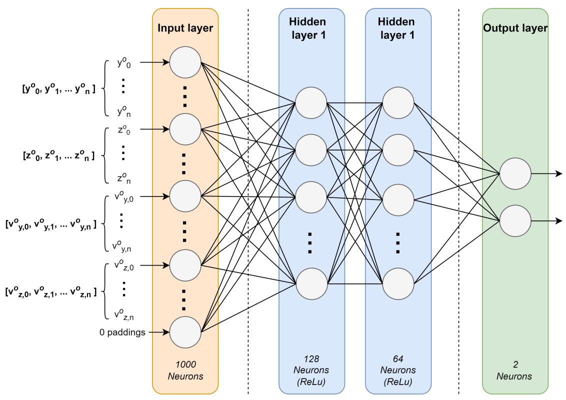


Fig. 12: Architecture of the neural network used to classify the underwater moving objects

IV. Case studies

This section presents various case studies that were conducted to evaluate the proposed underwater object detection model. A comprehensive overview of various scenarios is provided, distinguishing cases where the incorporation of domain knowledge proves beneficial and cases where the implementation of domain knowledge may not be justified. Two principal categories of scenarios are considered: those involving supervised learning and those involving semi-supervised learning. Per scenario, three distinct cases (with regards to data types for training, validation, and testing) were considered, as will be explained below:

1. Base data: training, validation, and testing were all conducted on ideal data (i.e., data that is generated by the simulator without the addition of any noise). This case represents the conventional approach in machine learning where all the sets contain the same type of data.
2. Low-quality data: training and validation used optical flow data, whereas testing was performed on ideal data. This case has been designed to evaluate the robustness of training on trajectories that are extracted using a non-perfect module.
3. Noisy data: training and validation were conducted on ideal data that is augmented with Gaussian noise, whereas testing was performed on ideal data. This case simulated situations when sensor noise is present in the training dataset.

The three cases mentioned above (i..e, base data, low-quality data, and noisy data) assess the impact of different data qualities on the performance of various object detection models, including a neural network with and without incorporated domain knowledge. The aim is to explore the potential advantages of incorporating domain-specific knowledge into the training procedure of a neural network, for various conditions. In addition to investigating the effects of the quality of the datasets, each case is also evaluated based on the dependability of the sample size of the training set. This evaluation is conducted by training two neural networks six times, with the training set size increasing from 100 to 600 samples in increments of 100. Both the validation and test datasets consistently hold 400 samples each.

To facilitate a robust assessment, a 10-fold cross-validation method is employed, with each fold containing 100 samples. Consequently, a total of 60 training runs per learning method are performed. A predefined seed is used to initialise the random neural network weights, ensuring that both the neural network with the domain knowledge module and the one without it start from the same initial conditions. Moreover, these neural networks are trained, validated, and tested on identical samples in order to maintain consistency across the experiments. The learning rate is maintained constant throughout the training process and is tuned per case of data type. The parameters of the mathematical models are estimated during the first epoch and are reused in subsequent epochs in order to improve the computational efficiency. This approach is viable because the mathematical models are static and remain unchanged during the training period.

The training of each neural network involves tuning the various hyperparameters of the model to optimise its performance. For this discussion, the batch size is kept equal to 1, and α in (15) is equal to 0.5. The temperature parameter in (15) and the learning rate are fine-tuned.

Lastly, the robustness of the supervised and semi-supervised learning scenarios against low-quality annotations is examined. This is because annotations that are predominantly generated by humans are prone to errors. Thus, the resilience of both training processes is evaluated against incorrectly labelled data. Based on the existing literature [79, 80] and our experience, it is deemed that incorrectly labelling 20% of the samples in the training dataset is a realistic scenario.

IV.i Supervised learning

The initial setup that was investigated included supervised learning, which encompasses the three previously discussed cases: base data, low-quality data, and

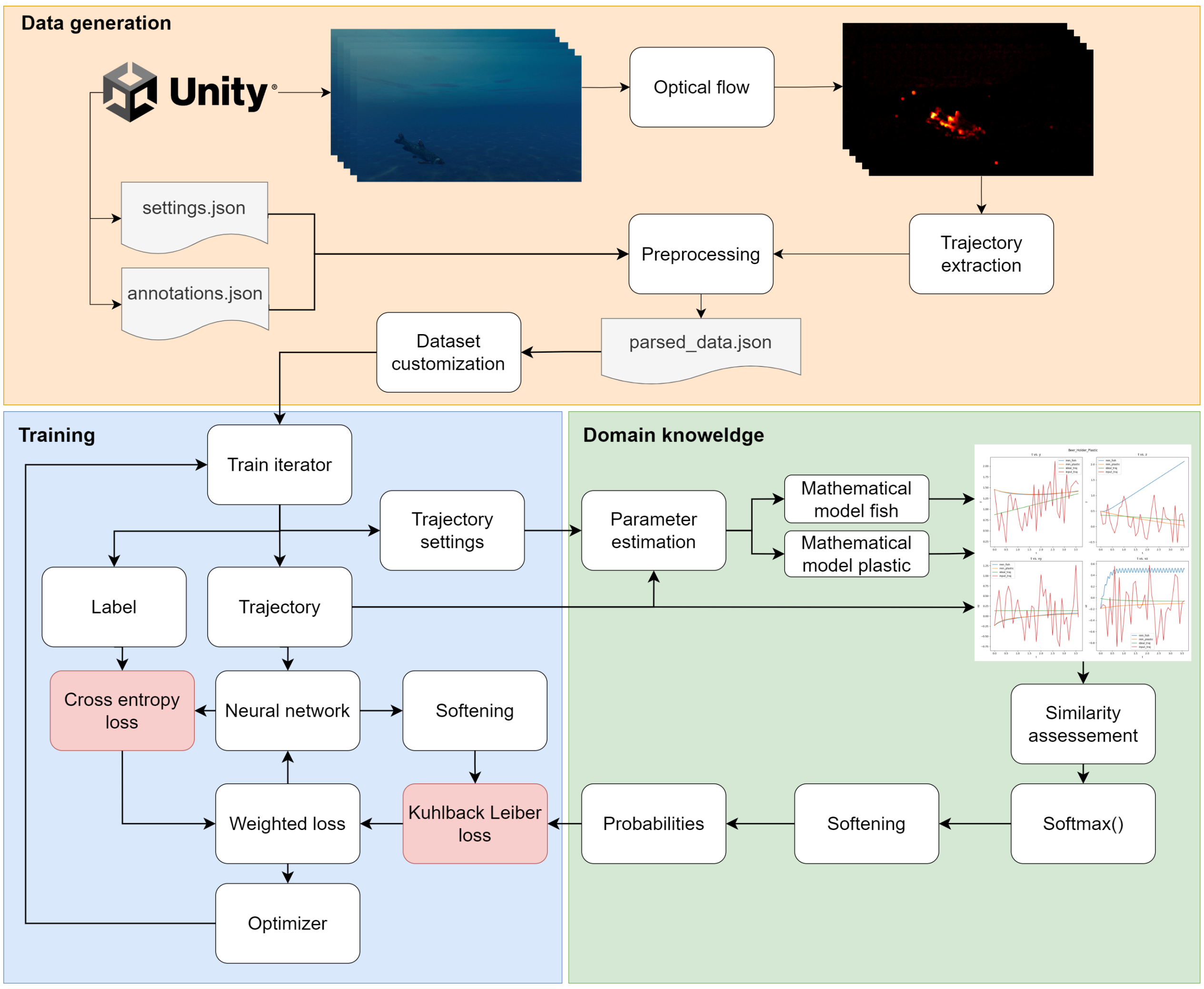


Fig. 13: Overview of the proposed architecture for training the classifying neural network

noisy data. For each case, the accuracy of the neural networks, the impact of the size of the training dataset on the performance of the trained model, and the consequence of using a poorly labelled dataset for training were examined.

The accuracy values obtained for the trained models for the base data case (with training, validating, and testing processes done on ideal data) with a decreasing number of training samples are illustrated in Fig. 14a. It shows that increasing the number of data points for training the models yields a higher performance for both models. As the mathematical model is static, the number of data points does not impact its performance. The accuracy graphs for both models are very similar, indicating that the domain knowledge is not impacting the learning significantly. Since the real-life data considered for this case is generated by the mathematical model, as expected the mathematical model shows the best performance among all. With a lower learning rate of 10^{-4}, the trend of the accuracy graph is further accentuated where both curves for the two neural network models almost fully coincide. This behaviour can be explained by analysing the probability distribution. The mathematical models closely approximate the ideal trajectory, positioning the probabilities at the extremes of the spectrum. Consequently, the information conveyed by these probabilities is identical to that contained in the labels, that meaning the domain knowledge does not contribute additional insight in this case.

The accuracy values obtained for the trained models for the low-quality data case are illustrated in Fig. 14c when using a temperature parameter value of 0.5. A temperature parameter below 1 hardens the probability distribution, effectively increasing the sharpness. A temperature value higher than 1 softens the distribution by bringing the probabilities closer to each other. As previously demonstrated, the probability distribution for optical flow is centred around 0.5. Utilising a temperature parameter value larger than 1 will soften the distribution (bringing the probabilities closer to each other), which dilutes the embedded information and slows down the learning process. Conversely, a temperature of 0.5 hardens the distribution and amplifies the information, thereby enhancing the learning speed. In comparison to the results presented in Fig. 14a, for the low-quality data case the neural network with domain-knowledge-infused learning consistently outperforms the neural network without domain knowledge, with an average improvement of 2%. This is because the domain knowledge module instructs the neural network with domain knowledge on which trajectories to learn or disregard.

Finally, the accuracy values for the trained models in the noisy data case are shown Fig. 14e, where the observed trend is consistent with the previous cases. In other words, the inclusion of domain knowledge consistently leads to a superior performance for the trained neural network. The domain knowledge module effectively prevents the neural network from overfitting to the Gaussian noise. In contrast, the absence of domain knowledge is marked by a (significant) performance decline at 600 data points, indicating overfitting. A temperature parameter value of 3 has been used. As the probabilities for the Gaussian noise are centred on the extremes, a temperature parameter value of 3 ensures softening the probability distributions and broadens the range of values.

The three cases explained above showed that the inclusion of domain knowledge will consistently improve the performance of the trained neural network, particularly for low-quality and noisy data. Similar accuracy results for lower numbers of samples for the training, indicate the potential of the neural network with domain knowledge for modelling cases where limited data is available.

Next, the robustness of both neural networks in situations where 20% of the training dataset has been mislabelled is discussed. The accuracy values for the trained models with 20% mislabelled base data is presented in Fig. 14b. In this scenario, the domain knowledge module demonstrates a distinct advantage. From 200 data points onwards, the neural network with domain knowledge consistently outperforms the neural network that does not incorporate domain knowledge. On average, a 2% improvement in performance is observed that diminishes with larger training datasets. Via providing insights into the quality of the data, the domain knowledge module acts as a regularisation in the explained situation. This enables the neural network to effectively identify and disregard incorrectly labelled trajectories. Consequently, incorporating domain knowledge enables the neural network to maintain accuracy even when trained on poorly annotated datasets, avoiding the accuracy loss that would occur if domain knowledge were not used.

The impact of introducing erroneous labels to the low-quality data case is depicted in Fig. 14d. It is evident that the incorporation of domain knowledge consistently increases the accuracy of the trained model across all training sample sizes. Although a slight reduction in the maximum accuracy is observed, compared to the scenarios without erroneous labels, the performance of the neural network with domain knowledge remains approximately 2-5% higher than that of the neural network that is trained without domain knowledge included.

Finally, the accuracy values for the models trained with noisy data including 20% mislabelled data are shown in Fig. 14f. A less consistent increase in the performance of the models is observed, with a notable decline at 400 data points. Moreover, while the neural network with domain knowledge shows a lower accuracy at 100 data points, it outperforms the neural network without domain knowledge at all other data sizes. In general, the performance across different training set sizes fluctuates significantly. These fluctuations are attributed to the high learning rate employed. The diminished performance at low data point counts is due to the domain knowledge module optimising for two objectives: label accuracy and alignment with domain knowledge. This dual optimisation, especially when combined with a high learning rate, becomes particularly challenging with a limited amount of data.

In summary, when utilising supervised learning with training data that is different from the testing data, the integration of domain knowledge into the training procedure of a neural network enhances the accuracy of the trained model and improves its robustness against poorly annotated datasets. Moreover, the temperature parameter value and the learning rate are crucial hyperparameters that must be carefully selected to optimise the performance of the neural network.

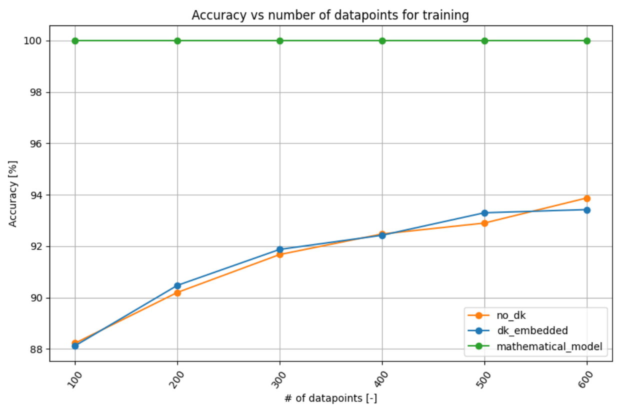
IV.ii Semi-supervised learning

In this part of the case study, we investigate whether the inclusion of domain knowledge into the training procedure of a neural network will similarly to the supervised learning, also facilitate the use of semi-supervised learning. Hence, an additional 400 unlabelled data points of the same type as the labelled training dataset were incorporated into the training procedure, with a value of α set to 1. This setup effectively eliminates the CE loss, in order to solely use the KL divergence loss for the 400 unlabelled trajectories. The value of α is set back to 0.5 for the other data points. The analysis proceeds with the same cases as before, now augmented with the inclusion of unlabelled data.

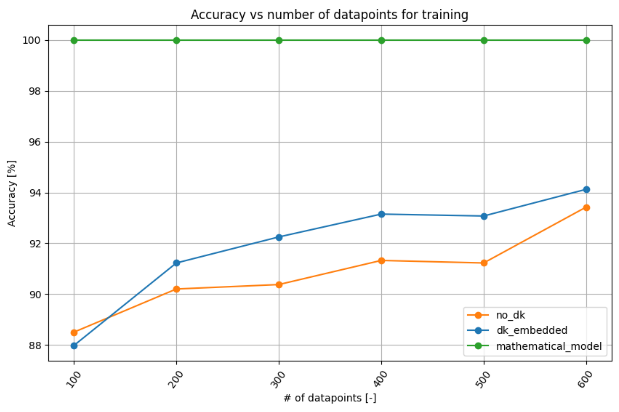
The accuracy values for the trained models when the base data case was used for training are shown in Fig. 15a. The optimal temperature parameter value was found to be 1. With the introduction of additional unlabelled data points, a lower learning rate was needed to yield an improved performance. From the figure, the knowledge-infused training procedure significantly outperforms the training procedure without domain knowledge, particularly with lower numbers of data samples. The out-performance diminishes when larger numbers of data points are used for the training. In fact, it is anticipated that the performance of the training process without domain knowledge will converge when more data becomes available. At the 400-trajectory mark, the performance of the neural network-trained with domain knowledge stabilises. This indicates that any further increase in the number of data points does not enhance the performance of the model. This also emphasises the advantage of employing semi-supervised learning together with domain knowledge, especially in scenarios where obtaining annotations comes with a large financial cost or is impossible in practice.

The accuracy values for the trained models when low-quality data is used for training are shown in Fig. 15c. In this case, a different trend is observed: A consistent out-performance by the neural network that is trained including domain knowledge. The neural network trained without the inclusion of the domain knowledge does not ever catch up, while the neural network with domain knowledge outperforms it for 5-7%. This consistent advantage suggests that the low-quality (optical flow) data is significantly more complex to learn unsupervised, compared to the base data case. It is expected that, with a sufficiently large number of data, the trend mirrors that of the base data case, albeit at a slower rate. These results were obtained using a temperature parameter value of 0.5, which is consistent with the value used in the supervised learning.

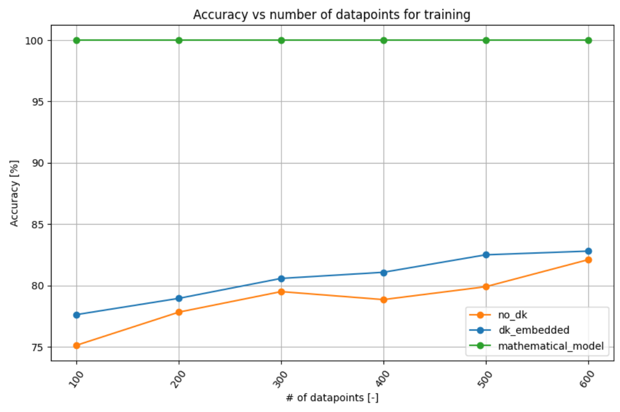
Finally, the accuracy values for the trained models when noisy data is used for training are illustrated in Fig. 15e. This case yet shows a different trend compared to the above two cases. The signature element of consistent out-performance of the neural network with domain knowledge is present, particularly for lower numbers of data points. Interestingly, at 400 data points —a threshold previously noted in the base data case as well— the training process that incorporates domain knowledge begins to learn the Gaussian noise characteristics. This is indicated by the decrease in the accuracy. Between 100 and 400 data points, the accuracy peaks for this process.



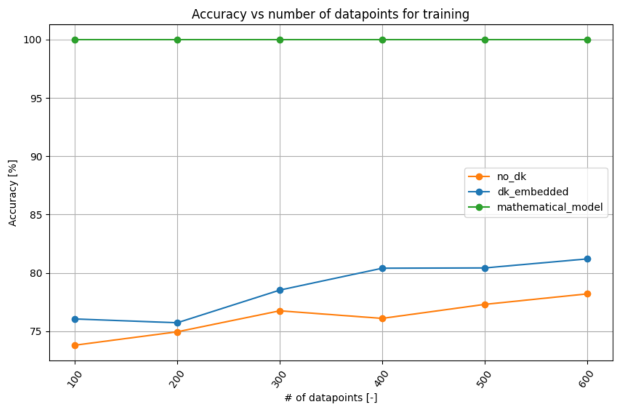
(a) Base data case: The accuracy when training, validating, and testing the two neural networks (with and without domain knowledge) on ideal data.



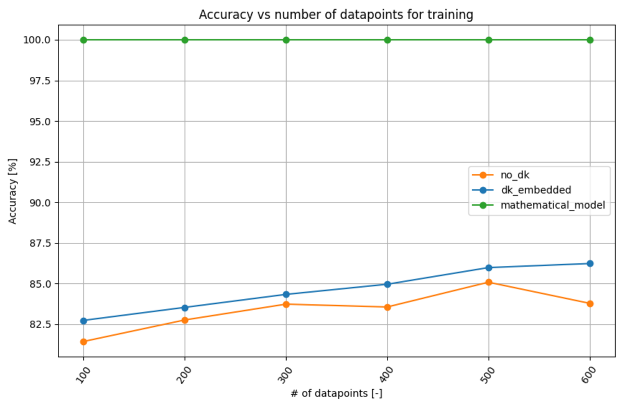
(b) Base data case: The accuracy when training, validating, and testing the two neural networks (with and without the domain knowledge) on ideal data with 20% of the samples mislabelled.



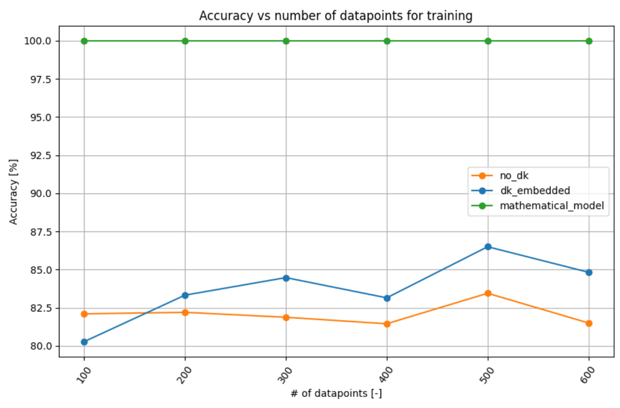
(c) Low-quality data case: The accuracy when training the two neural networks (with and without the domain knowledge) on optical flow data, validating them on optical flow data, and testing them on ideal data.



(d) Low-quality data case: The graph is constructed by training on optical flow data with 20% of the samples mislabelled, validating on optical flow data and testing on ideal data.



(e) Noisy data case: The accuracy when training the two neural networks (with and without the domain knowledge) on data including Gaussian noise, validating them on data including Gaussian noise, and testing on ideal data.



(f) Noisy data case: The accuracy values when training the two neural networks (with and without the domain knowledge) on data including Gaussian noise with 20% of the samples mislabelled, validating them on data including Gaussian noise, and testing them on ideal data.

Fig. 14: Supervised learning: Comparison between the mathematical model and the neural networks trained with and without domain knowledge. The accuracy shown is based on the test set. Each row corresponds to a different type of data. The first column corresponds to training data with no errors in the labelling and the second column corresponds to training data with 20% mislabelled. The orange, blue, and green curves correspond to, respectively, the neural network without domain knowledge, the neural network with domain knowledge, and the domain knowledge module alone.

Conversely, the training process without domain knowledge reaches its peak of accuracy only at 400 data points. It is concluded that with larger numbers of data points, some of the noise is learned by the neural network with domain knowledge. This effect could be minimised again with a dynamic value for α that is adapted to balance the CE loss and KL divergence during the training process. A temperature parameter value of 3 is used, as for the supervised learning.

In summary, knowledge-infused training significantly boosts the accuracy for semi-supervised learning, especially with fewer training samples. This improvement is evident not only in the noisy data case, but also in the base data case, demonstrating the efficacy of semi-supervised learning including domain knowledge across these various conditions.

Next, similarly to the supervised learning, we evaluated the robustness of the training with semi-supervised learning with and without domain knowledge when 20% mislabelled data was included in the training dataset. Fig. 15b shows the accuracy of the neural networks trained with and without domain knowledge on the base data, where 20% of the data has been wrongly labelled. A similar pattern for the accuracies as in the case of non-erroneous data is observed, with an enhanced performance per data point level and a plateau reached at 400 data points. The increased discrepancy –compared to the base data case– in the performance of the neural networks with knowledge-infused and standard training that is observed for larger numbers of data points, is attributed to the presence of the data with erroneous labels.

The obtained accuracies for the neural networks for the low-quality data, with 20% of the data being labelled wrongly, are illustrated in Fig. 15d. The same hyperparameters as for the low-quality data case without erroneous labels have been used. A similar trend as for the low-quality data case without the erroneous labels is observed, but with a smaller difference in the accuracies for both neural networks. This trend, which does not align with the conclusions drawn from the supervised learning, can be elucidated by examining the loss function, which is a linear combination of the CE loss and the KL divergence loss (see (15)). During the initial phase of training on the 400 unlabelled samples, the neural network trained with domain knowledge only minimises the KL divergence loss. In the subsequent phase, the CE loss, which may include errors, is much larger than the KL divergence loss, and thus becomes more influential in the total loss. Consequently, the optimisation process prioritises the CE loss. This issue can be addressed by introducing α as an adaptable parameter, to ensure that the KL divergence loss maintains its effectiveness in the total loss. However, implementing a varying α is considered beyond the scope of this research.

Finally, when 20% of the noisy data is mislabelled, the accuracy graphs are obtained as shown in Fig. 15f. A similar trend as for the noisy data without wrong labelling is again observed, while at the 400 data points mark, the performance accuracy of the neural network that is trained without domain knowledge experiences a significant drop. This indicates that with a larger number of samples, erroneous elements are impacting further the learning of this neural network. This drop does not occur in the performance accuracy of the neural network that is trained including domain knowledge. It is noteworthy that a temperature parameter value of 2 yields the best results in this case. Additionally, the knowledge-guided learning process requires an average of 12 epochs to train, whereas the training process without domain knowledge requires 25 epochs. This reveals another advantage of combining semi-supervised learning with noisy data and domain knowledge for situations where the long training time should be avoided, such as in online learning.

In summary, this section demonstrated that integrating domain knowledge allows for semi-supervised learning to significantly enhance the robustness of a neural network against low-quality and noisy data, especially including erroneous labels. The analysis across various cases and scenarios showed that domain knowledge-infused training consistently outperforms training that excludes domain knowledge, particularly with lower numbers of data points and with imperfect annotations. The analysis of the impact of the number of data points on the performance accuracy revealed that increasing the number of data points does not necessarily enhance the performance. Additionally, it was found that a subset of data points, combined with augmented training, can achieve similar performance accuracy as when larger datasets are used in training processes without domain knowledge included. This finding is significant for applications where obtaining labelled data is particularly challenging or impossible.

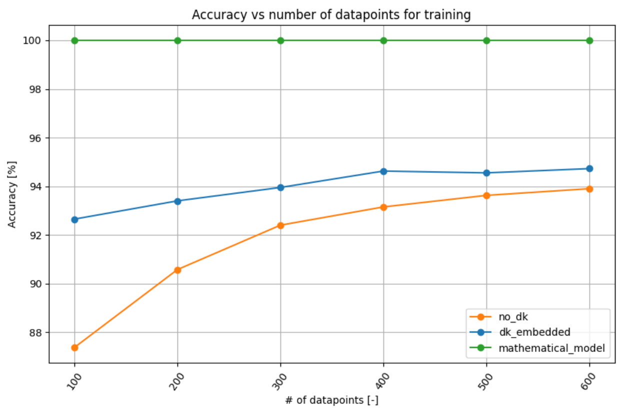
IV.iii Computational efficiency

While the standalone domain knowledge module exhibits high accuracy, it is crucial to consider the intended final application, i.e., the use of (ROVs) for autonomous underwater cleanup operations. These ROVs are constrained by limited power and on-board computational resources, where these necessitate computationally efficient algorithms for online object detection. The domain knowledge module is composed of nonlinear differential equations and a parameter estimation module, both of which are computationally intensive. Furthermore, the computation time scales almost linearly with any new class that is added to the domain knowledge module, making it not a scalable solution. In contrast, this problem is not observed with the neural network. The domain knowledge has a run time of 5.0381 seconds while the neural network has a run time of 0.0032 seconds, hence a difference of 15267 fold difference. The experiments are performed on NVIDIA GeForce RTX 3090. The neural network operates orders of magnitude faster than the domain knowledge module, making it the only feasible option for real-time on-board ROV applications.

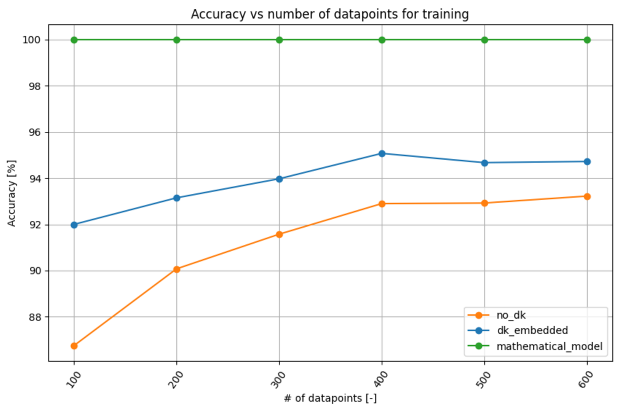
IV.iv Recommendations for future research

Based on the results presented and discussed in this case study, the following recommendations are made for future research on enabling autonomous trash cleanup.

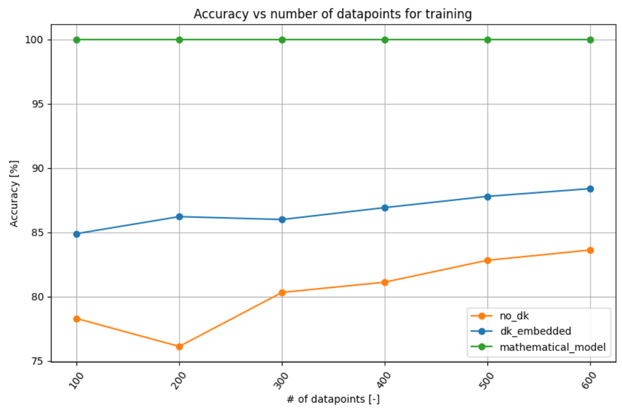
- The current simulator may be enhanced in terms of dynamics and visual fidelity of the simulated camera. This will allow for validation with real-life experiments and comparison with state-of-the-art neural networks.
- The trajectory extraction module consisting of optical flow and clustering may be improved. Ideally, these components will be enhanced using a Kalman filter to predict the subsequent positions of the moving objects and to verify these predictions using the extraction module. The proposed pipeline should also be augmented with state-of-the-art object detection neural networks that have superior performance for static objects.
- Making the hyperparameters α, temperature, and learning rate adaptable throughout the training of the knowledge-infused learning is recommended for enhancing the performance of the resulting model [81].
- While the current setup and codebase of the project are designed to support only two classes of moving objects (i.e., plastic and fish), future de-



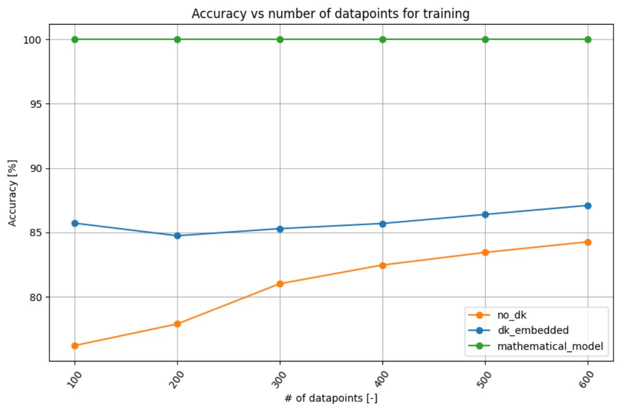
(a) Base data case: The accuracy when training, validating, and testing the two neural networks (with and without domain knowledge) on ideal data.



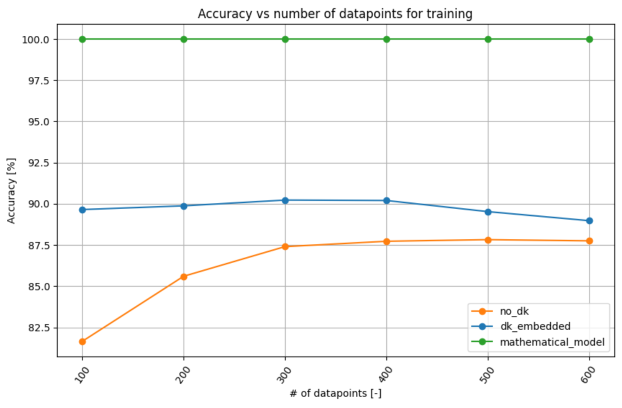
(b) Base data case: The accuracy when training, validating, and testing the two neural networks (with and without the domain knowledge) on ideal data with 20% of the samples mislabelled.



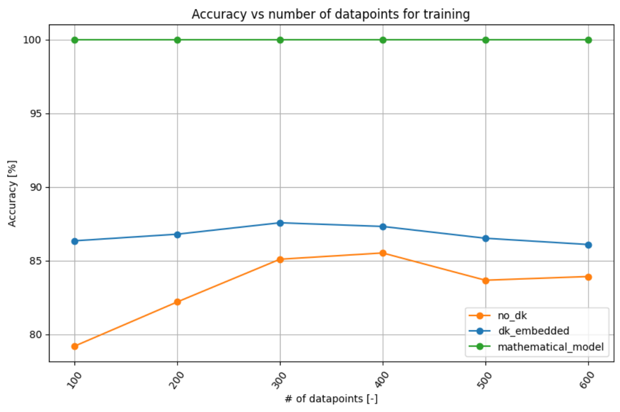
(c) Low-quality data case: The accuracy when training the two neural networks (with and without the domain knowledge) on optical flow data, validating them on optical flow data, and testing them on ideal data.



(d) Low-quality data case: The graph is constructed by training on optical flow data with 20% of the samples mislabelled, validating on optical flow data and testing on ideal data.



(e) Noisy data case: The accuracy when training the two neural networks (with and without the domain knowledge) on data including Gaussian noise, validating them on data including Gaussian noise, and testing on ideal data.



(f) Noisy data case: The accuracy values when training the two neural networks (with and without the domain knowledge) on data including Gaussian noise with 20% of the samples mislabelled, validating them on data including Gaussian noise, and testing them on ideal data.

Fig. 15: Semi-supervised learning: Comparison between the mathematical model and the neural networks trained with and without domain knowledge. The accuracy shown is based on the test set. Each row corresponds to a different type of data. The first column corresponds to training data with no errors in the labelling and the second column corresponds to training data with 20% mislabelled. The orange, blue, and green curves correspond to, respectively, the neural network without domain knowledge, the neural network with domain knowledge, and the domain knowledge module alone.

velopments should expand this to a broader range of objects.

- For real-world applications, the mathematical model for fish and plastic motions should be augmented to include additional forces, such as the rotational drag and the wave-induced motion, in order to enhance the accuracy and realism of the models.

V. Conclusion

Trash pollution in the rivers and oceans has become a global problem. While autonomous cleanup represents the most viable solution to this problem, the performance of current underwater trash detection methods falls short of enabling full-scale autonomous operations. This research introduces a novel pipeline that has been developed to enhance frame-by-frame object detection by incorporating dynamic domain knowledge into the training procedure of the classifying neural network and by basing the predictions solely on the object's trajectory, eliminating the need for CNNs. This paper was focused on embedding domain knowledge into the training procedure of a trajectory classifier and on investigating the performance of the resulting model across various cases and dataset types. In order to validate the hypothesis that domain knowledge improves the detection performance, a simulator was developed in Unity to ensure the generation of high-quality datasets. A trajectory extraction module was also designed using optical flow and a clustering algorithm as the first step of the proposed pipeline. Subsequently, two mathematical models were formulated to provide the foundation for the domain knowledge module. The probability distribution of the domain knowledge module was computed using the mean squared error in conjunction with the softmax function. This knowledge-infused training process was then compared with a standard training process that did not include domain knowledge across multiple scenarios. The findings revealed that training with domain knowledge notably enhances the robustness of the resulting neural network model against low-quality and noisy data, particularly when such datasets involve wrongly labelled data. Moreover, the inclusion of domain knowledge facilitated semi-supervised learning and consistently surpassed in performance accuracy the model that was trained solely based on data. In particular, an improvement of 10% in the accuracy on the test set was observed in scenarios where only a limited number of annotations were available.

VI. Acknowledgements

The author would like to express sincere gratitude to Piotr Szleg and Paweł Barczyk for their generosity in licensing their simulator for this project. This research would not have been possible without their support.

References

[1] L. Parker, "The world's plastic pollution crisis, explained," *National Geograhic*, February 2024.

[2] M. Fava, "Ocean plastic pollution: An overview of data and statistics," May 2022. Accessed January 27, 2023.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pp. 1877–1901, 2020.

[4] R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, *et al.*, "Gemini: A family of highly capable multimodal models," 2023. arXiv preprint arXiv:2312.11805.

[5] C. Fu, R. Liu, X. Fan, P. Chen, H. Fu, W. Yuan, M. Zhu, and Z. Luo, "Rethinking general underwater object detection: Datasets, challenges, and solutions," *Neurocomputing*, vol. 517, pp. 243–256, 2023.

[6] R. A. Dakhil and A. R. H. Khayeat, "Review on deep learning techniques for underwater object detection," in *Data Science and Machine Learning*, Academy and Industry Research Collaboration Center (AIRCC), 9 2022.

[7] S. Xu, M. Zhang, W. Song, H. Mei, Q. He, and A. Liotta, "A systematic review and analysis of deep learning-based underwater object detection," *Neurocomputing*, vol. 527, pp. 204–232, 2023.

[8] A. Jesus, C. Zito, C. Tortorici, E. Roura, and G. De Masi, "Underwater object classification and detection: first results and open challenges," in *OCEANS 2022 - Chennai*, IEEE, Feb. 2022.

[9] R. M. Gray, *Entropy and Information Theory.* New York: Springer-Verlag, first edition corrected ed., 2023. Corrected first Edition copyright 2011 by Robert M. Gray.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[11] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8." https://github.com/ultralytics/ultralytics, 2023. Version 8.0.0.

[12] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 2019.

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, 2017.

[14] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[15] D. K. Prasad, C. K. Prasath, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Challenges in video based object detection in maritime scenario using computer vision," *International Journal of Computer and Information Engineering*, vol. 11, pp. 31–36, 2017.

[16] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker, "Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems,"

IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 1, pp. 614–633, 2023.

[17] A. Karpatne, W. Watkins, J. S. Read, and V. Kumar, *Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling*, ch. 15, pp. 353–372. Taylor & Francis, 2022.

[18] P. Szleg, P. Barczyk, B. Maruszczak, S. Zielinski, and E. Szymańska, "Simulation environment for underwater vehicles testing and training in unity3d," in *Annual Meeting of the IEEE Industry Applications Society*, 2022.

[19] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, vol. abs/1503.02531, 2015.

[20] H. Chen, J. Chen, and J. Ding, "Data evaluation and enhancement for quality improvement of machine learning," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 831–847, 2021.

[21] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," *International Journal on Advances in Software*, vol. 10, pp. 1–20, 07 2017.

[22] J. A. for Marine-Earth Science and T. (JAMSTEC), "Deep-sea debris database." (Accessed on 02/03/2023).

[23] J. Hong, M. Fulton, and J. Sattar, "Trashcan: A semantically-segmented dataset towards visual detection of marine debris," *CoRR*, vol. abs/2007.08097, 2020.

[24] M. Fulton, J. Hong, M. J. Islam, and J. Sattar, "Robotic detection of marine litter using deep visual detection models," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5752–5758, IEEE, 2019.

[25] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *International Conference on Intelligent Robots and Systems*, (Vilamoura, Algarve, Portugal), 10 2012.

[26] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, pp. 1–8, 2016.

[27] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981.

[28] B. Chiang and J. Bohg, "Lecture notes cs231a: Computer vision, from 3d reconstruction to recognition: Optical and scene flow," February 2022.

[29] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis* (J. Bigun and T. Gustavsson, eds.), (Berlin, Heidelberg), pp. 363–370, Springer Berlin Heidelberg, 2003.

[30] A. Jepson and M. Black, "Mixture models for optical flow computation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 760–761, 1993.

[31] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[32] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, 2015.

[33] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2462–2470, IEEE, 2017.

[34] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 402–419, Springer, 2020.

[35] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4489–4497, 2015.

[36] Z. Tang, Y. Zhao, Y. Wen, and M. Liu, "A survey on backbones for deep video action recognition," *IEEE Transactions on Image Processing*, vol. 33, pp. 525–540, 2024.

[37] C. Feichtenhofer, "X3d: Expanding architectures for efficient video recognition," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 200–210, 2020.

[38] Y. Wang, K. Li, Y. Li, Y. He, B. Huang, Z. Zhao, H. Zhang, J. Xu, Y. Liu, Z. Wang, S. Xing, G. Chen, J. Pan, J. Yu, Y. Wang, L. Wang, and Y. Qiao, "Internvideo: General video foundation models via generative and discriminative learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[39] L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang, and Y. Qiao, "Videomae v2: Scaling video masked autoencoders with dual masking," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14549–14560, 2023.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[41] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[42] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, 2014.

[43] F. Fernández de la Mata, A. Gijón, M. Molina-Solana, and J. Gómez-Romero, "Physics-informed neural networks for data-driven simulation: Advantages, limitations, and opportunities," *Physica A: Statistical Mechanics and its Applications*, vol. 610, p. 128415, 2023.

[44] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[45] C. Anitescu, B. İsmail Ateş, and T. Rabczuk, *Physics-Informed Neural Networks: Theory and Applications*, pp. 179–218. Cham: Springer International Publishing, 2023.

[46] T. Dash, S. Chitlangia, A. Ahuja, and A. Srinivasan, "A review of some techniques for inclusion of domain-knowledge into deep neural networks," *Sci Rep*, vol. 12, p. 1040, 2022.

[47] T. M. Deist, A. Patti, Z. Wang, D. Krane, T. Sorenson, and D. Craft, "Simulation-assisted machine learning," *Bioinformatics*, vol. 35, pp. 4072 – 4080, 2018.

[48] J. Pfrommer, C. Zimmerling, J. Liu, L. Kärger, F. Henning, and J. Beyerer, "Optimisation of manufacturing process parameters using deep neural networks as surrogate models," *Procedia CIRP*, vol. 72, pp. 426–431, 2018.

[49] J.-L. Wu, H. Xiao, and E. Paterson, "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework," *Physical Review Fluids*, vol. 3, July 2018.

[50] G. Fung, O. L. Mangasarian, and J. W. Shavlik, "Knowledge-based support vector machine classifiers," *Procedia Computer Science*, 2002.

[51] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing, "Harnessing deep neural networks with logic rules," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (K. Erk and N. A. Smith, eds.), (Berlin, Germany), pp. 2410–2420, Association for Computational Linguistics, Aug. 2016.

[52] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1, pp. 119–165, 1994.

[53] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks," in *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, AAAI'90, p. 861–866, AAAI Press, 1990.

[54] J. Fletcher and Z. Obradovic, "Combining prior symbolic knowledge and constructive neural network learning," *Connection Science*, vol. 5, pp. 365–375, 1993.

[55] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, pp. 71–101, 1993.

[56] Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, and H. Soh, "Semantically-regularized logic graph embeddings," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 4233–4243, Curran Associates, Inc., 2019.

[57] M. Schiegg, M. Neumann, and K. Kersting, "Markov logic mixtures of gaussian processes: Towards machines reading regression data," *Proceedings of Machine Learning Research*, vol. 22, pp. 1223–1231, 2012.

[58] A. Kimmig, S. H. Bach, M. Broecheler, B. Huang, and L. Getoor, "A short introduction to probabilistic soft logic," in *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, pp. 1–9, Curran Associates, Inc., 2012.

[59] R. N. King, O. Hennigh, A. T. Mohan, and M. Chertkov, "From deep to physics-informed learning of turbulence: Diagnostics," in *71st Annual Meeting of the APS Division of Fluid Dynamics*, 2018.

[60] G. Hautier, C. C. Fischer, A. Jain, T. Mueller, and G. Ceder, "Finding nature's missing ternary oxide compounds using machine learning and density functional theory.," *ChemInform*, vol. 41, 2010.

[61] Y. Du, Z. Liu, H. Basevi, A. Leonardis, B. Freeman, J. B. Tenenbaum, and J. Wu, "Learning to exploit stability for 3d scene parsing," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4145–4155, Curran Associates, Inc., 2018.

[62] Unity Technologies, *Unity Game Engine*. Unity Technologies, San Francisco, CA, 2024. Version 2024.1.

[63] Unity Technologies, "Camera.viewporttoworldpoint," 2024. Accessed: 2024-06-06.

[64] RoutineStudio, "Plastic water bottle." https://sketchfab.com/3d-models/plastic-water-bottle-731efe2635c9472c9c1e4fdb1f8fbd13, 2020. Accessed: 15/01/2023.

[65] Froggreen, "Fish." https://sketchfab.com/3d-models/fish-ae9089d355d244aebd9abee4da7d35af, 2017. Accessed: 20/01/2023.

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

[67] OpenCV, "K-means clustering," 2023. Accessed: 2024-05-21.

[68] S. Mallick, "Geometry of image formation." LearnOpenCV, OpenCV, 2020. Accessed: 2024-04-21.

[69] G. Hollows and N. James, "Understanding focal length and field of view." https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/, 2023. Accessed: 2023-05-23.

[70] L. F. Shampine and M. W. Reichelt, "The matlab ode suite," *SIAM Journal on Scientific Computing*, vol. 18, pp. 1–22, Jan 1997.

[71] K. Pearson, "Vii. note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240 – 242, 1895.

[72] C. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 100, no. 3/4, pp. 441–471, 1987.

[73] F. Hausdorff, *Grundzüge der Mengenlehre*. Leipzig, Germany: Veit & Comp., 1914.

[74] H. Hahn, "Sur quelques points du calcul fonctionnel," *Monatshefte für Mathematik und Physik*, vol. 19, pp. A47–A48, 1908.

[75] H. Sakoe, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, pp. 159–165, 1978.

[76] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[77] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-fold cross validation in prediction error estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 569–575, 2010.

[78] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[79] M. Staats, M. Thamm, and B. Rosenow, "Reevaluating loss functions: Enhancing robustness to label noise in deep learning models," *ArXiv*, vol. abs/2306.05497, 2023.

[80] S. Bai, S. Zhou, Z. Qin, L. Wang, and N. Zheng, "Robust noisy label learning via two-stream sample distillation," *arXiv preprint arXiv:2404.10499*, 2024.

[81] Z. Li, X. Li, L. Yang, B. Zhao, R. Song, L. Luo, J. Li, and J. Yang, "Curriculum temperature for knowledge distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 1504–1512, 2023.

[82] G. Li, H. Liu, U. K. Müller, C. J. Voesenek, and J. L. van Leeuwen, "Fishes regulate tail-beat kinematics to minimize speed-specific cost of transport," *Proceedings of the Royal Society B: Biological Sciences*, vol. 288, 2021.

Appendix A: Unity simulator setup

This appendix consists of an in-depth discussion about the physics implemented in the simulator alongside a comprehensive listing of the parameter values utilised for data generation.

The scene is generated using various parameters. An overview can be found in Table 2.

Table 2: Unity scene parameter ranges

Unity Simulator parameters			
Name	**Unit**	**Minimum value**	**Maximum value**
Underwater fog strength	-	6	120
Particle emission rate	#/s	3	10
Bubble emission rate	#/s	0	3
Water current z	m/s	-0.3	0.3
Light intensity	lx	0.8	1.8
Light angle x	deg	60	120
Light angle y	deg	0	360
Distance from the camera	m	1	6
Spawn position x	m	-3	-6
Spawn position y	m	-1.8	-1.2
Spawn position z	m	-6	6
Spawn rotation y	deg	0	360
Margin	-	0.25	0.75

As mentioned in the main paper, the `FixedUpdate()` method is used to update the physics. Two custom forces were added: buoyancy force and drag force. The buoyancy force formula can be found in (16).

$$f^{\text{buoy}} = \rho^{\text{fluid}} \cdot g \cdot V^o \tag{16}$$

With ρ^{fluid} equal to 997 for the density of water, g the gravitational acceleration constant equal to 9.81 m/s^2, and V^o the volume.

The simulator offers two methods for simulating gravity: it can either treat gravity as a point source at the object's centre of mass or discretize the gravitational force across the object's entire surface based on voxel count. Additionally, to simulate minor disturbances commonly found in underwater environments, the simulator incorporates Perlin noise. This technique generates a pseudo-random pattern of floating-point values that modify the gravitational force, effectively simulating environmental disturbances.

Note that the buoyancy formula in (16) uses volume V^o. Accurately calculating an object's volume based on its mesh can be challenging. As mentioned before, the simulator opts for a box collider instead of a mesh collider. This collider outlines the object with a box. This suffices for many objects but in some cases, such as for the plastic beer holder, it considerably overestimates the volume. To address such discrepancy, a volume scaling parameter is introduced, allowing for more accurate buoyancy calculations. Specific parameters for various objects are detailed in Table 3, Table 4, and Table 5. In each `FixedUpdate()` step, the buoyancy force is calculated and added to the object.

The second force implemented is the drag force. This simulator only considers linear drag and models angular drag using constant damping. The drag force is caused by the difference in speed between the fluid v^{fluid} and the object v^{obj}. In typical applications, the following relation holds $v^{\text{fluid}} << v^{\text{obj}}$. In the underwater domain with slow-moving objects, the velocity of the fluid has a large effect, especially on passive objects. The formula for the drag force alongside one axis is given in (17).

$$f_i^{\text{drag}} = \frac{1}{2} \cdot \rho^{\text{fluid}} \cdot v_i^2 \cdot c_i^{\text{d},o} \cdot A_i^o \tag{17}$$

with $i \in \{x, y, z\}$ and $o \in \{\text{bottle}, \text{holder}, \text{fish}\}$. The value for ρ^{fluid} is the same as before, v_i^2 the difference in fluid and object speed squared, $c_i^{\text{d},o}$ the drag coefficient, and A_i^o is the surface area. The values used to solve the equations are given in Table 3, Table 4, and Table 5. Note that the surface areas are based on the box collider discussed before. For each `FixedUpdate()` step, the speed of the current is subtracted from the velocity of the object and the calculated force is added to the centre of mass of the object.

Table 3: Unity fish parameters

Fish parameters			
Name	**Symbol**	**Unit**	**Value**
Mass	m^{fish}	kg	3.6
Drag coefficient x	$c_x^{\text{d, fish}}$	#/s	0.5
Drag coefficient y	$c_y^{\text{d,fish}}$	#/s	0.5
Drag coefficient z	$c_z^{\text{d,fish}}$	m/s	0.5
Area x	A_x^{fish}	m^2	0.1003
Area y	A_y^{fish}	m^2	0.0898
Area z	A_z^{fish}	m^2	0.0323
Damping coefficient	ζ^{fish}	-	2e-5
Volume scaler	-	-	0.1876
Swim force	F_z^f	-	7
Probability against current swimming	p	-	0.7

Table 4: Unity plastic bottle parameters

Plastic bottle parameters			
Name	**Symbol**	**Unit**	**Value**
Mass	m^{bottle}	kg	0.78
Drag coefficient x	$c_x^{\text{d, bottle}}$	#/s	0.82
Drag coefficient y	$c_y^{\text{d, bottle}}$	#/s	0.82
Drag coefficient z	$c_z^{\text{d, bottle}}$	m/s	1.03
Area x	A_x^{bottle}	m^2	0.0152
Area y	A_y^{bottle}	m^2	0.0138
Area z	A_z^{bottle}	m^2	0.0050
Damping coefficient	ζ^{bottle}	-	2e-6
Volume scaler	-	-	0.8914

Table 5: Unity plastic beer holder parameters

Plastic beer holder parameters			
Name	**Symbol**	**Unit**	**Value**
Mass	m^{holder}	kg	0.3
Drag coefficient x	$c_x^{\text{d, holder}}$	#/s	0.83
Drag coefficient y	$c_y^{\text{d, holder}}$	#/s	0.88
Drag coefficient z	$c_z^{\text{d, holder}}$	m/s	1.17
Area x	A_x^{holder}	m^2	0.0092
Area y	A_y^{holder}	m^2	0.0834
Area z	A_z^{holder}	m^2	0.0139

Table 5 continued from previous page

Damping coefficient	ζ^{holder}	-	2e-6
Volume scaler	-	-	0.11

Each run of the simulator outputs a series of images displaying the movement of the object over time. Some examples of the plastic bottle, plastic beer holder and fish can be found in Fig. 16, Fig. 17 and Fig. 18 respectively.

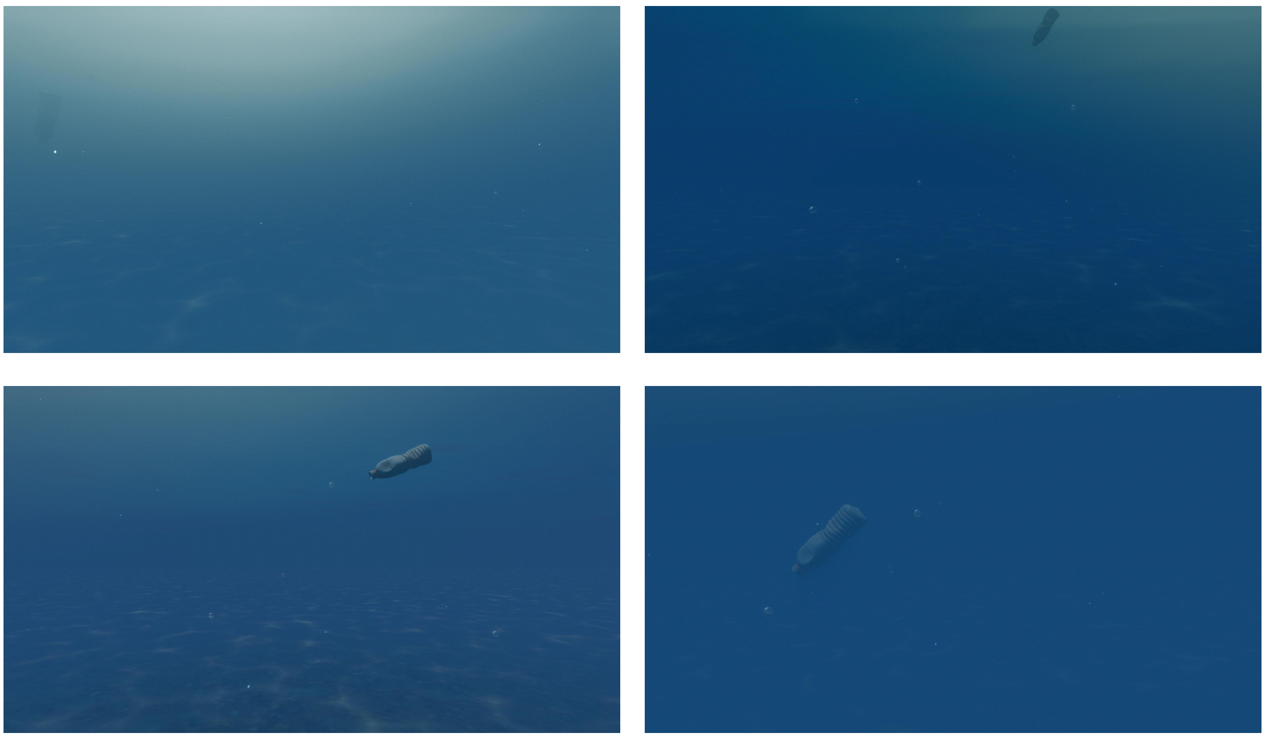


Fig. 16: Example images of the plastic bottle inside the simulation.

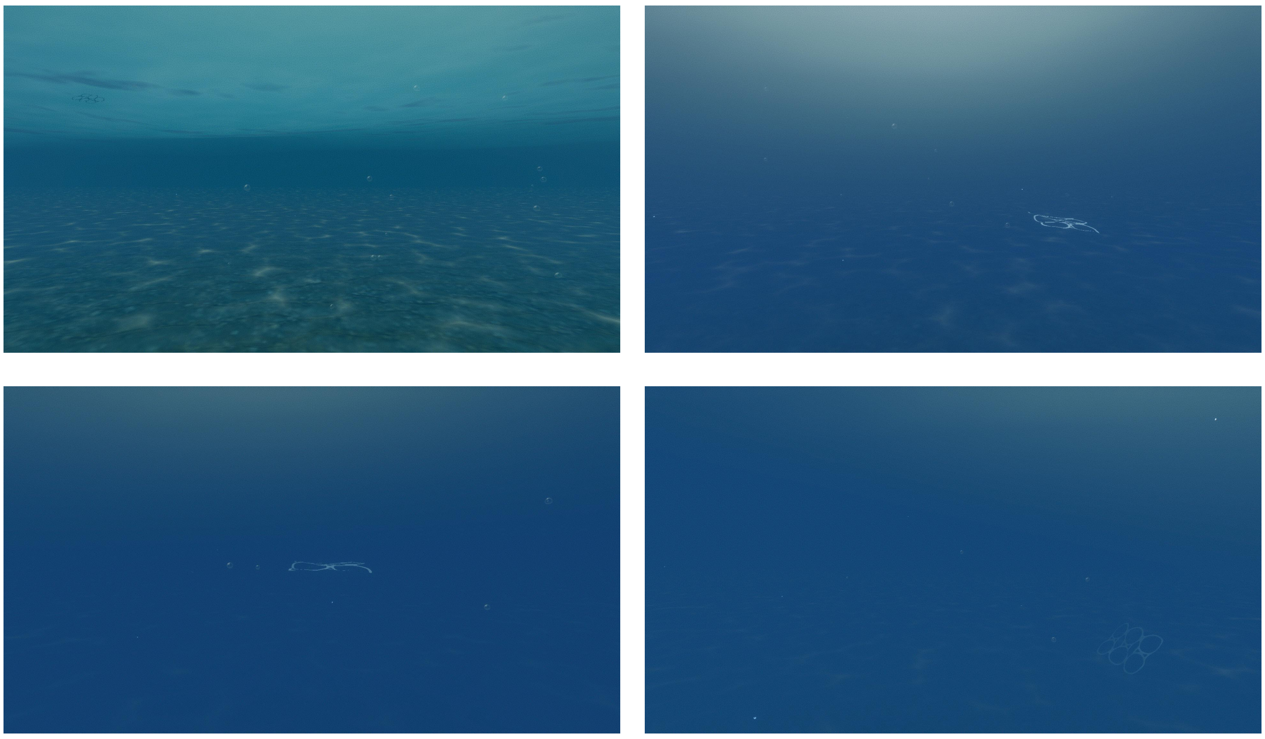


Fig. 17: Example images of plastic beer holder inside the simulation.

Fig. 18: Example images of the fish inside the simulation.

Appendix B: Motion extraction and clustering

This appendix discusses the motion extraction and clustering algorithm in more detail. The motion between two frames is extracted using optical flow. Many forms of optical flow exist, but the Farneback method is used for this research. The Farneback method is built on approximating each pixel's neighbourhood with a quadratic polynomial, found in (18). This results in a local signal model.

$$f(\mathbf{x}) \sim \mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c \tag{18}$$

$\mathbf{A}$ is a symmetric matrix, $\mathbf{b}$ is a vector and $\mathbf{c}$ is a scalar. The coefficients are estimated using a weighted least squares fit. The weighting has two components: certainty and applicability. The certainty reflects the quality or confidence of the pixel data. High contrast and sharpness typically have a high level of certainty. The applicability component refers to the relevance of the pixel's data. This value often decreases with distance from the centre of the neighbourhood. A new signal can be constructed by translating the polynomial over the distance d. The new signal can be found in (19).

$$f_2(\mathbf{x}) = f_1(\mathbf{x} - \mathbf{d}) = (\mathbf{x} - \mathbf{d})^T \mathbf{A}_1 (\mathbf{x} - \mathbf{d}) + \mathbf{b}_1^T (\mathbf{x} - \mathbf{d}) + c_1 \tag{19}$$

Subsequently, if A_1 is non-singular, an expression for d can be derived, shown in (20) with the parameters defined in (21).

$$\mathbf{d} = -\frac{1}{2}\mathbf{A}_1^{-1} (\mathbf{b}_2 - \mathbf{b}_1) \tag{20}$$

$$\begin{aligned} \mathbf{A}_2 &= \mathbf{A}_1, \\ \mathbf{b}_2 &= \mathbf{b}_1 - 2\mathbf{A}_1\mathbf{d}, \\ c_2 &= \mathbf{d}^T \mathbf{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1. \end{aligned} \tag{21}$$

This approach makes use of local polynomial approximations as global approximations are not realistic. The displacement is assumed to vary across the frame and a multiscale approach is taken to iteratively refine the displacements. The output of this approach is a mask with the magnitudes indicating the pixel velocity. The assumption is made that the object of interest has the highest velocity of the frame and thus the largest optical flow value. Finding the location of the object in the frame is not a trivial task. The particles, bubbles and waves also have an optical flow. These items add noise to the mask and assuming that the object's position coincides with the single highest value is untrue.

A more complex centroid extraction module is designed and an overview can be found in Fig. 19. The module applies the k-means clustering algorithm twice, once focused on the magnitudes and once focused on the positions. The k-means clustering algorithm is a form of unsupervised learning. It works by iteratively partitioning the dataset into k predefined distinct non-overlapping clusters. The algorithm minimizes the squared distance between each point and the centroid of the cluster. The objective function can be seen in (22).

$$J = \sum_{i=1}^{K} \sum_{j=1}^{n_i} \|x_j - \mu_i\|^2 \tag{22}$$

With K the number of clusters, n_i is the number of data points in cluster i, x_j is the j-th data point in cluster i and μ_i the centroid of the i-th cluster. The algorithm tries to find the centroid locations that yield the lowest value of J. K-means clustering is a very popular technique widely used in industry thanks to its speed and simplicity. Important to note is the sensitivity to outliers, dependence on the initialization and tendency to find local minima.

For the magnitude clustering, a value of 3 is used. The cluster covering the largest number of pixels is assumed to be the background. This can be observed in Fig. 20. Cluster zero indicates the background. Cluster 1 contains the object (plastic beer holder) and some noise. Cluster 2 contains only noise. Another example can

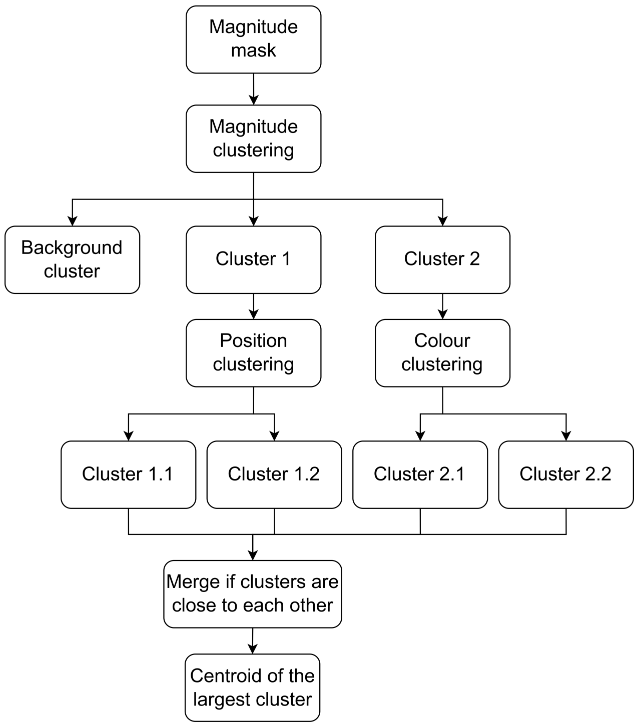


Fig. 19: Architecture motion extraction module.

be found in Fig. 21. Again, the background can be seen in cluster 0. In this case, the object (fish) is detected in both cluster 1 and cluster 2. The level of noise is more significant in cluster 2 than in cluster 1. This shows the benefit of using three magnitude clusters.

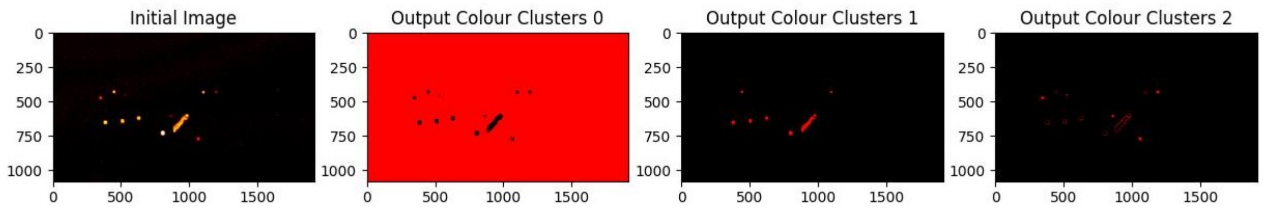


Fig. 20: Output magnitude clustering example 1.

The k-means clustering algorithm is applied again to the two extracted position clusters, but this time spatially focused. The spatial clustering algorithm is applied to magnitude cluster 1 and magnitude cluster 2 from Fig. 20 and the output can be found in Fig. 22 and Fig. 23 respectively. It can be seen that the object is present in Fig. 22 together with some noise. Spatial cluster 0 filters the majority of the noise out and spatial cluster 1 shows the outline of the object with limited noise. The output of applying the spatial clustering applied to magnitude cluster 2 is shown in Fig. 23. Although only noise is present, it verifies that the algorithm correctly spatially clusters the pixels.

The spatial clustering algorithm is also applied to magnitude cluster 1 and magnitude cluster 2 from Fig. 21. In this example, both magnitude clusters contain a part of the object. In Fig. 24, the object is split into two

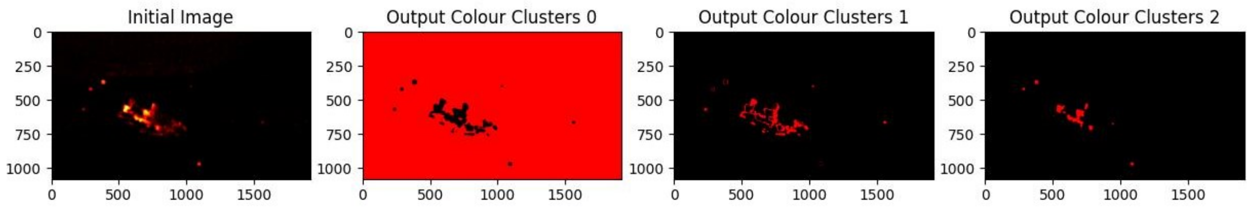


Fig. 21: Output magnitude clustering example 2.

parts. Similar behaviour can be observed in Fig. 25. To cover the cases where multiple clusters belong to the object, the centroids are compared and the clusters are merged if near each other. The final extracted clusters can be found in Fig. 26 and Fig. 27.

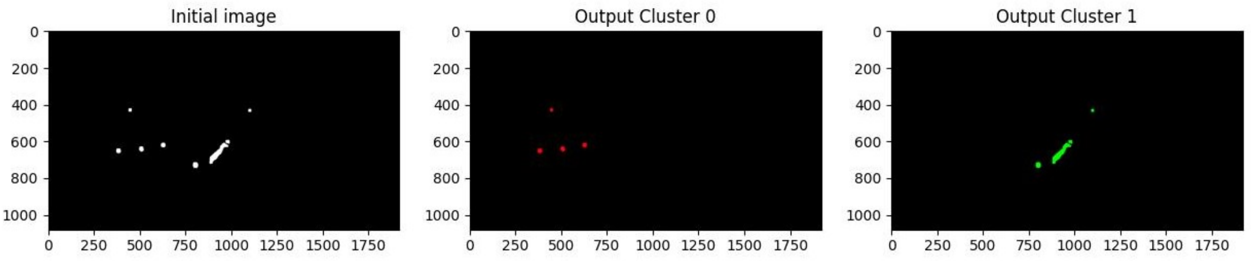


Fig. 22: Output spatial cluster 1 example 1.

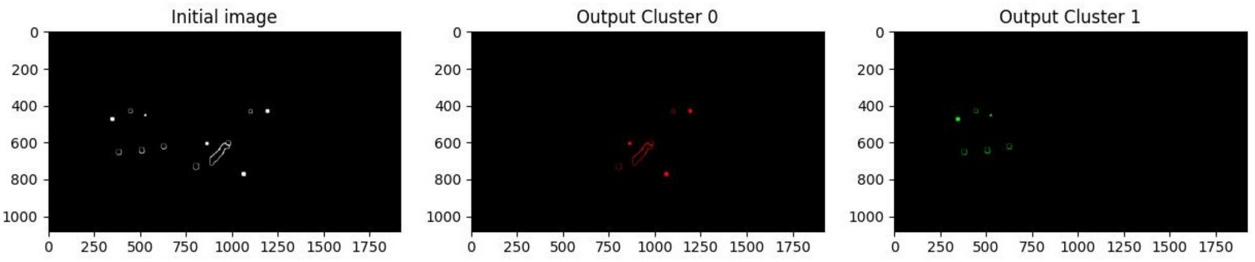


Fig. 23: Output spatial cluster 2 example 1.

Note that this method is built on the assumption that the object has a higher or similar velocity compared to the bubbles and marine snow. If this is not the case, the object is not seen on the mask such as the case in Fig. 28

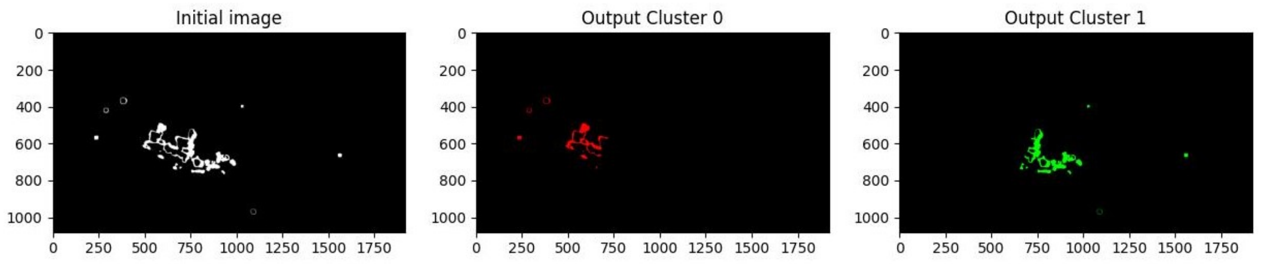


Fig. 24: Output spatial cluster 1 example 2.

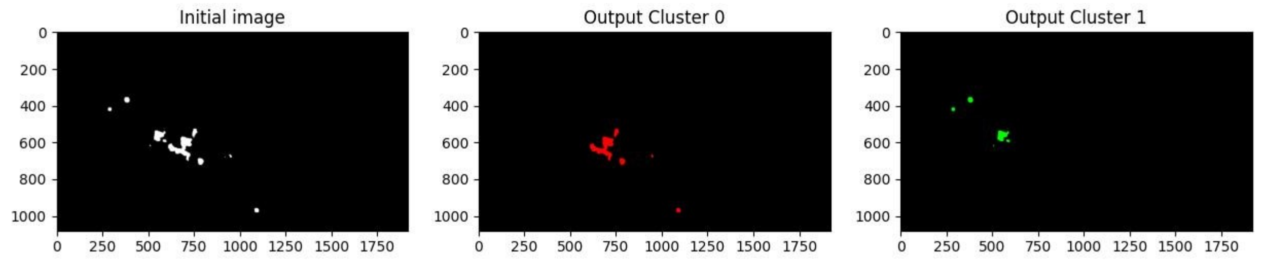
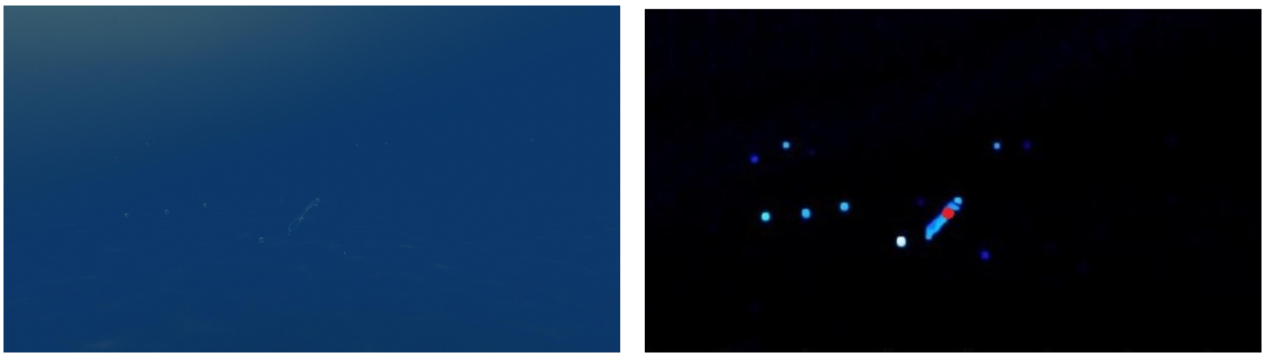
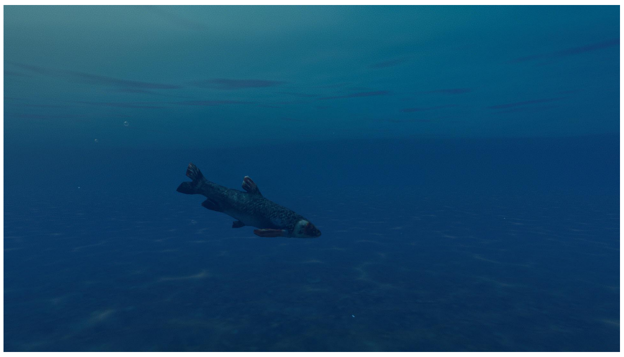


Fig. 25: Output spatial cluster 2 example 2.



(a) Original image example 1 containing plastic beer holder. (b) Final centroid (indicated in red) example 1.

Fig. 26: Original image and the optical flow output.


(a) Original image example 2 containing fish.

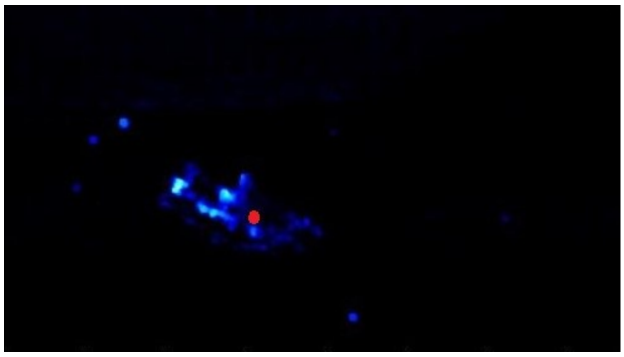
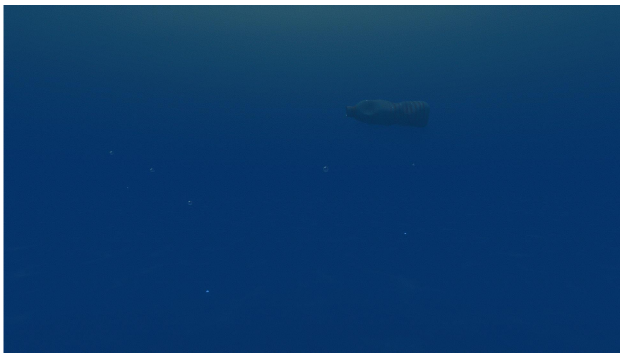

(b) Final centroid (indicated in red) example 2.

Fig. 27: Original image and the optical flow output.


(a) Original image containing plastic bottle.

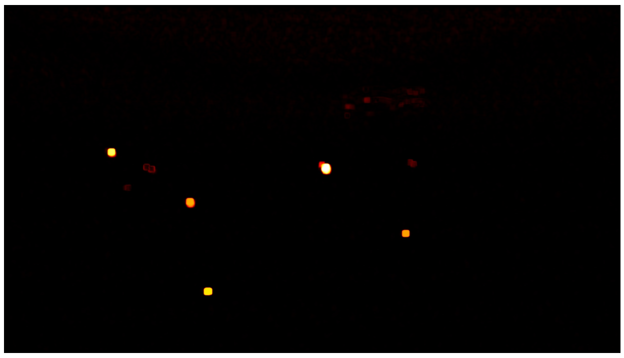

(b) Mask generated using optical flow.

Fig. 28: Example if the object's velocity is lower than the surroundings.

Appendix C: Mathematical model

The mathematical model module consists of two models: a passive plastic model and an active fish model. These two models are largely the same, with the fish force as the main difference. This accounts for the fish's tail movement generating thrust and is modelled as a sinusoidal motion with a tail frequency f^{tail}. Research indicates that frequencies ranging from 1 to 90 Hz are commonly observed to be used by fish in nature [82]. A value of 50 Hz is chosen.

This appendix discusses the verification performed to ensure the models are behaving as expected. The total generated dataset consists of 2500 trajectories. Each of these trajectories has three different sources: optical flow, Gaussian noise and ideal data. Each of these different types can be used with different parameter estimation settings: off, low, medium, high and full. Note that only the low setting is used in the main paper.

At the first level (parameter estimation off), all the parameters are extracted. The second level estimates the volume, which impacts the object's buoyancy, affecting its vertical position and velocity. The third level includes mass estimation, crucial for both buoyancy and drag equations, impacting the object's overall dynamics. The fourth level adds estimates for surface areas, influencing the drag forces in both directions. The final level includes the fish force which drastically impacts the fish mathematical model. This approach allows for tailored parameter estimation that can be adjusted based on the model's complexity and desired performance. An overview of the different levels can be found in Table 6. The mathematical model yields four outputs: y^o, z^o, $\dot{y}^o$, and $\dot{z}^o$ over the length of the input trajectory that can vary between 1 second and 20 seconds.

Table 6: Different levels of parameter estimation

Parameter estimation levels		
Level	**Param extracted**	**Param estimation**
Off	A_z, A_y, F_{f_z}, m, V	-
Low	A_z, A_y, F_{f_z}, m	V
Medium	A_z, A_y, F_{f_z}	m, V
High	F_{f_z}	A_z, A_y, m, V
Full	-	A_z, A_y, F_{f_z}, m, V

The mathematical model module receives three primary inputs: the input trajectory, the desired degree of parameter estimation, and the class. Its output comprises the calculated trajectories for either plastic or fish, utilising the initial conditions specified by the input trajectory.

To assess the accuracy of the mathematical model is the root mean squared error (RMSE) calculated for each of the states y^o, z^o, $\dot{y}^o$ and $\dot{z}^o$. The results can be found in Table 7.

In Table 7, a clear difference between the data sources can be observed. As expected, the mathematical model performs exceptionally well on the ideal data, significantly outperforming the other data sources. This is thanks to the high similarity between the differential equations present in the simulation and in both mathematical models.

The RMSE for the y^o optical flow is considerably lower than for the Gaussian noise, but this trend can not be observed in the other states. The disparities between y^o and z^o and $\dot{z}^o$ and $\dot{z}^o$ can be explained by the difference in methods to calculate the velocities between the two data sources. Optical flow uses the derivatives from the position while the Gaussian noise makes use of the ideal velocities and augments these with noise. This also explains why the errors in the Gaussian noise are constant across the states.

For each of the data sources, one might expect the error to be minimised when using the true parameters. However, an alternative behaviour can be observed; the error decreases with the increase of parameter estimation level up to a certain point beyond which the error begins to rise again. This tipping point varies with each data source and state. For optical flow, this tipping point is either high or non-existent. For the Gaussian noise scenario, it consistently occurs at a high level of parameter estimation while for the ideal data, this occurs at either medium or high. This pattern emerges because the parameter estimation allows the mathematical model

Table 7: Overview of average root mean squared error (RMSE) for 2500 trajectories for the mathematical model

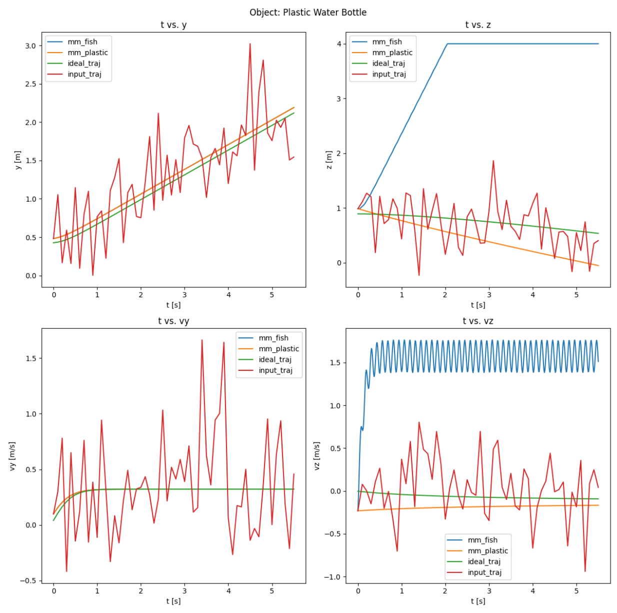
Data Source	Parameter Estimation	$RMSE_{y^o}$ [m]	$RMSE_{z^o}$ [m]	$RMSE_{\dot{y^o}}$ [m/s]	$RMSE_{\dot{z^o}}$ [m/s]
Optical flow	OFF	0.161	0.670	2.08	19.6
Optical flow	LOW	0.0472	0.669	1.98	19.6
Optical flow	MEDIUM	0.0320	0.594	1.98	19.5
Optical flow	HIGH	0.0386	0.543	1.97	19.5
Optical flow	FULL	0.0404	0.461	1.97	19.3
Gaussian noise	OFF	0.272	0.287	0.115	0.124
Gaussian noise	LOW	0.194	0.287	0.123	0.124
Gaussian noise	LOW	0.194	0.287	0.123	0.124
Gaussian noise	MEDIUM	0.174	0.265	0.122	0.119
Gaussian noise	HIGH	0.174	0.252	0.122	0.116
Gaussian noise	FULL	0.179	0.278	0.124	0.139
Ideal	OFF	0.00190	0.00130	0.00207	0.00203
Ideal	LOW	0.00370	0.00130	0.00320	0.00203
Ideal	MEDIUM	0.000315	0.00135	0.000924	0.00190
Ideal	HIGH	0.00168	0.00122	0.00105	0.00150
Ideal	FULL	0.00219	0.0281	0.00160	0.0177

to fine-tune itself on the input trajectory, resulting in a better performance compared to the scenario without fine-tuning.

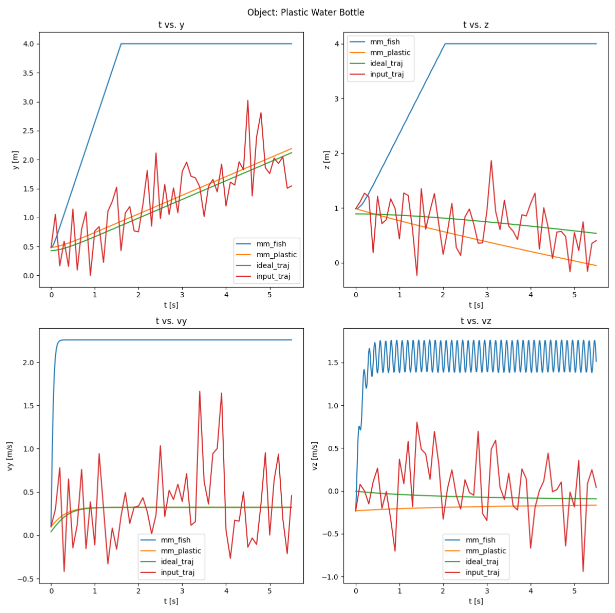
A similar trend can be observed when looking at the graphs. For each of the parameter estimation levels, the plots for y^o, z^o, $\dot{y^o}$ and $\dot{z^o}$ are generated. Both the plastic and fish models are generated using the levels of parameter estimation and plotted, together with the ideal trajectory. Important to note is the “absence” of the fish model in the case with parameter estimation off for y^o and $\dot{y^o}$. This happens because the plastic and fish models have the same differential equations (for y^o and $\dot{y^o}$) and thus produce identical trajectories when the same parameters are used. The graphs are produced for the case of a plastic bottle so the plastic model uses the parameters of the plastic bottle and so does the fish model.

The Gaussian noise graphs can be found in Fig. 29, Fig. 30 and Fig. 31. The mathematical model receives the Gaussian noise-added trajectory as input, performs parameter estimation and generates a trajectory aiming to mimic the input trajectory as closely as possible.

The optical flow graphs are shown in Fig. 32, Fig. 33 and Fig. 34. The difference between the optical flow and Gaussian noise trajectories becomes clear as well. The errors propagate to the velocities in the optical flow while this is not present in the Gaussian noise.

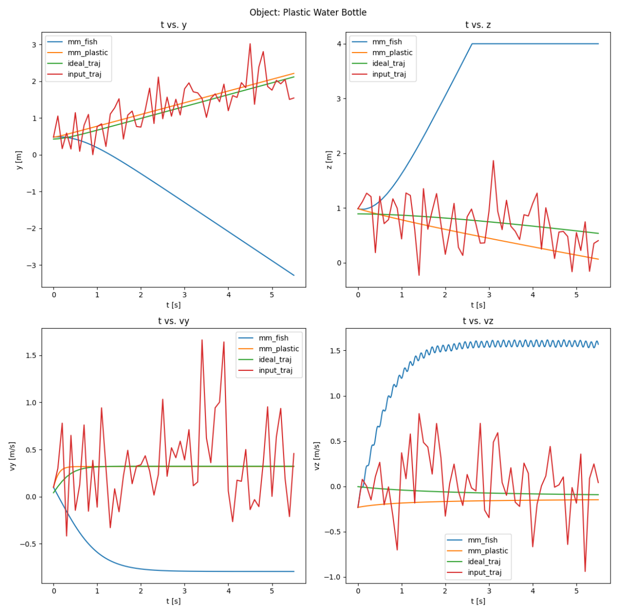


(a) Gaussian noise trajectory - parameter estimation off

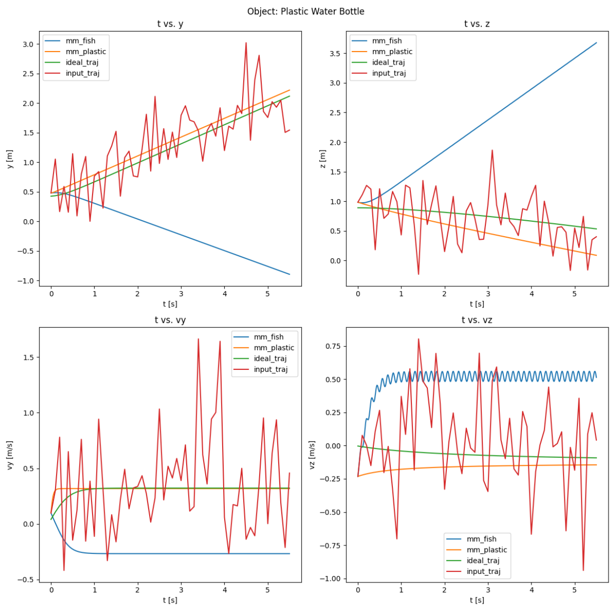


(b) Gaussian noise trajectory - parameter estimation low

Fig. 29: Comparison level parameter estimation off and low for the Gaussian noise case.



(a) Gaussian noise trajectory - parameter estimation medium



(b) Gaussian noise trajectory - parameter estimation high

Fig. 30: Comparison level parameter estimation medium and high for the Gaussian noise case.

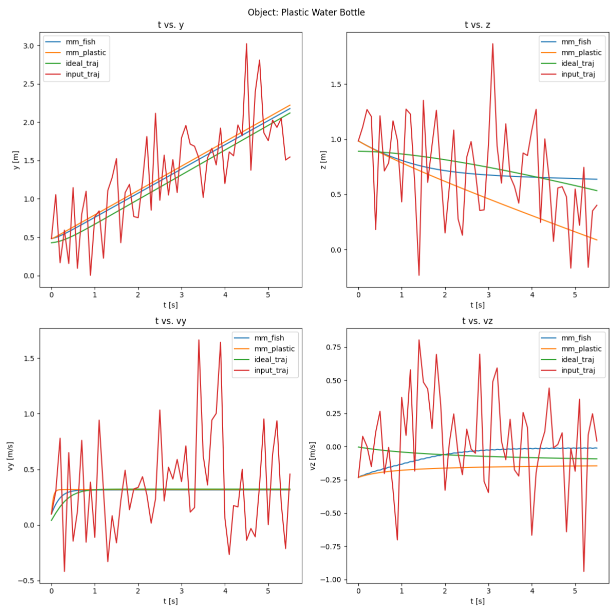
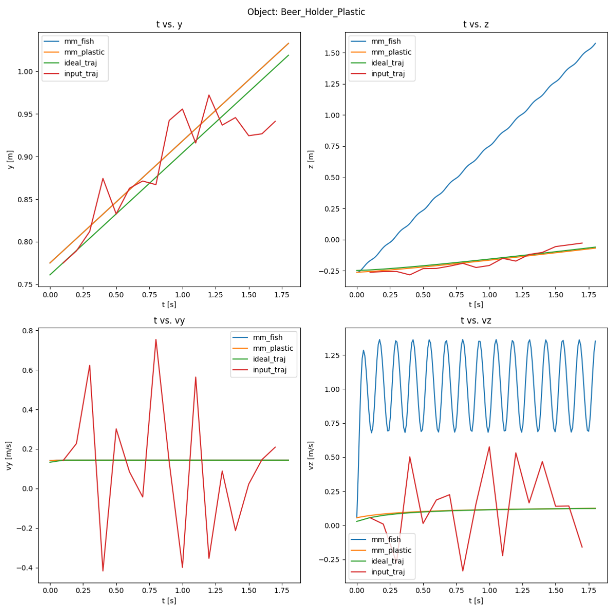
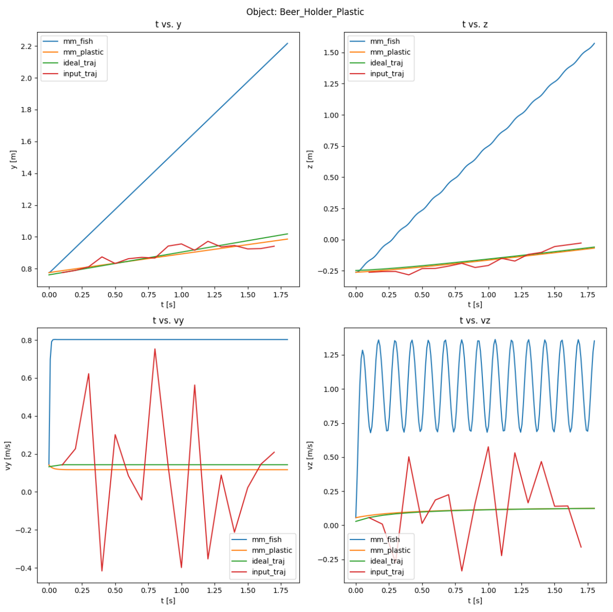


Fig. 31: Gaussian noise trajectory - parameter estimation full

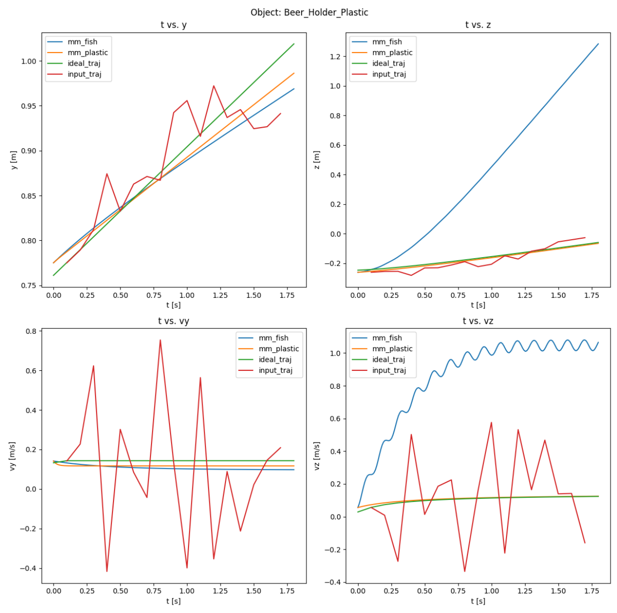


(a) Optical flow trajectory - parameter estimation off

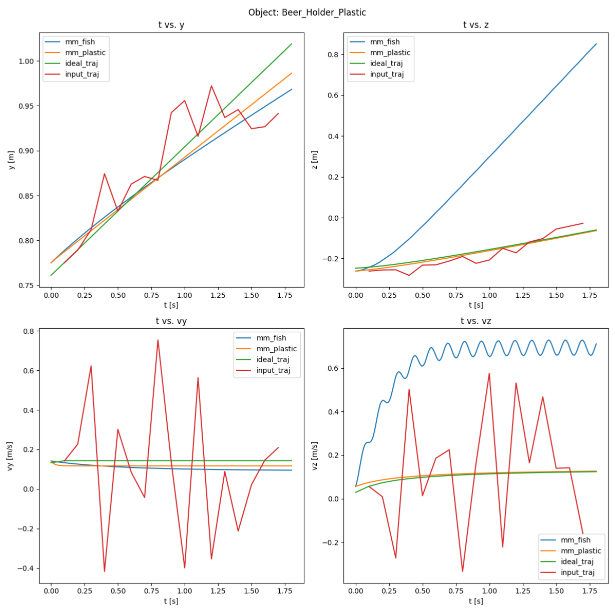


(b) Optical flow trajectory - parameter estimation low

Fig. 32: Comparison level parameter estimation off and low for the optical flow case.



(a) Optical flow trajectory - parameter estimation medium



(b) Optical flow trajectory - parameter estimation high

Fig. 33: Comparison level parameter estimation medium and high for the optical flow case.

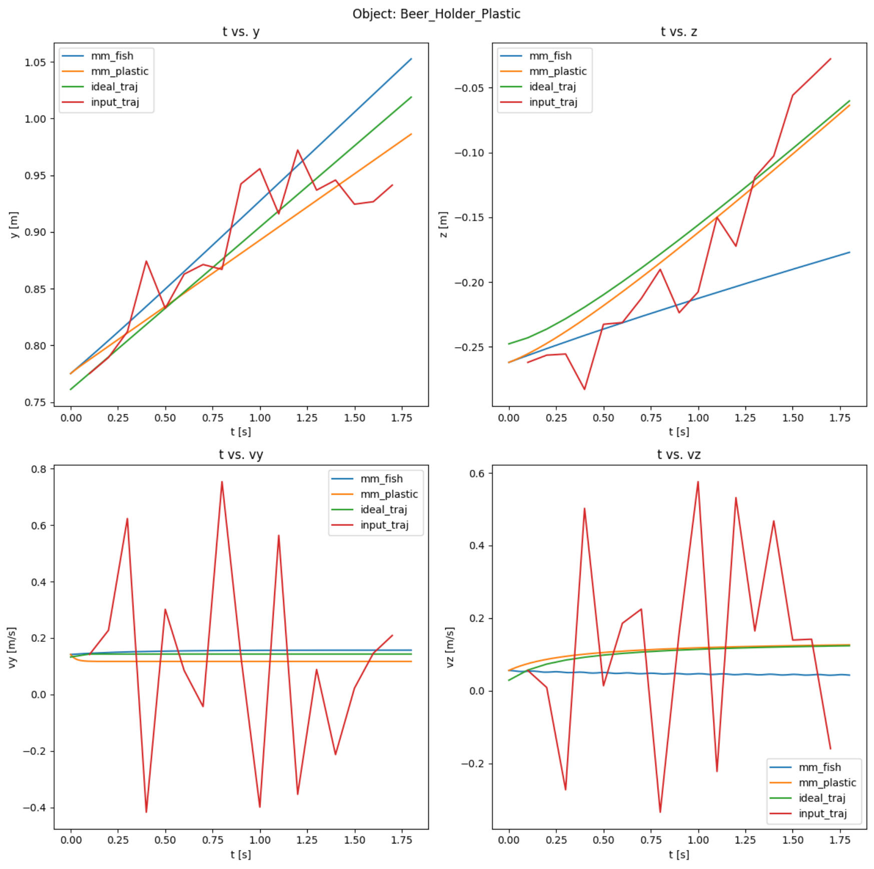


Fig. 34: Optical flow trajectory - parameter estimation full.

Appendix D: Domain knowledge module

The domain knowledge module is central to this method. This module primarily computes the similarity between the input trajectory and the mathematical models for plastic and fish. The class with the highest similarity is selected as the prediction for the mathematical model. It can be seen that the similarity metric is crucial to the entire setup; an inappropriate choice can lead to a suboptimal solution. To select the most optimal metric, two criteria are set: accuracy in predicting the correct class and the alignment between the expected and observed probability distribution. The accuracy is important such that the method does not hold back the training, but the probability distribution provides additional information beyond what is offered by the labels.

The accuracy is investigated by running each similarity metric on the entire dataset of 2500 samples. As mentioned before, the mathematical model has a large impact on the performance of the domain knowledge module. It is therefore important to investigate the behaviour of each metric on the different data types and various parameter estimations. The results can be found in Fig. 35, Fig. 36a, Fig. 36b, Fig. 37a, Fig. 37b for the parameter estimation levels off, low, medium, high and full respectively.

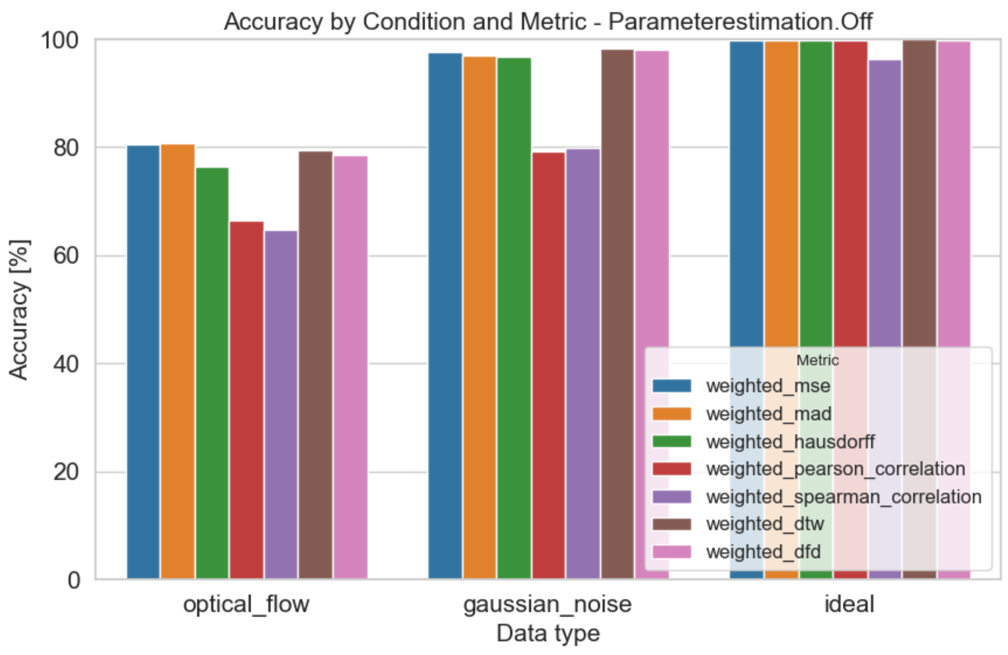
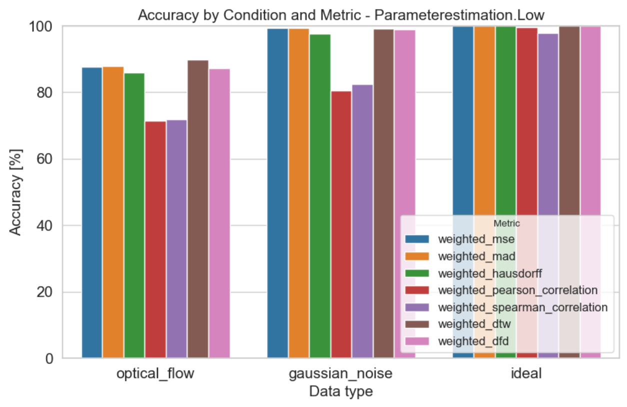


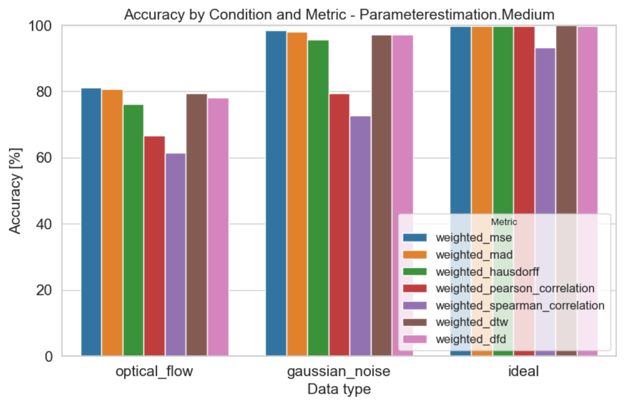
Fig. 35: Comparison accuracy of various similarity metrics for the parameter estimation level off.

The accuracy is lowest with the optical flow data, followed by the Gaussian noise and approaches 100% for the ideal data. This is expected as the mathematical models align most closely with the ideal data. It is observed that all metrics, except for the Pearson and Spearman correlations, maintain very similar levels of accuracy. Due to their significantly lower accuracy, these two metrics are excluded from the remainder of the analysis. In the higher parameter estimation levels, it can be observed that the accuracy of the MAD (Mean average distance) decreases for the ideal data.

The second criterion considered is the correspondence between the probability distributions. This ensures that additional data, not present in the annotations alone, is conveyed to the neural network. To assess the probability distribution, each of the metrics is applied to the entire dataset and the resulting probabilities are plotted. Note that each similarity metric is converted to probabilities using the softmax function. The graphs for the discrete Fréchet distance, dynamic time warping, Hausdorff coefficient, mean average distance and the mean squared error can be found in Fig. 38, Fig. 39, Fig. 40, Fig. 41, Fig. 42 respectively.

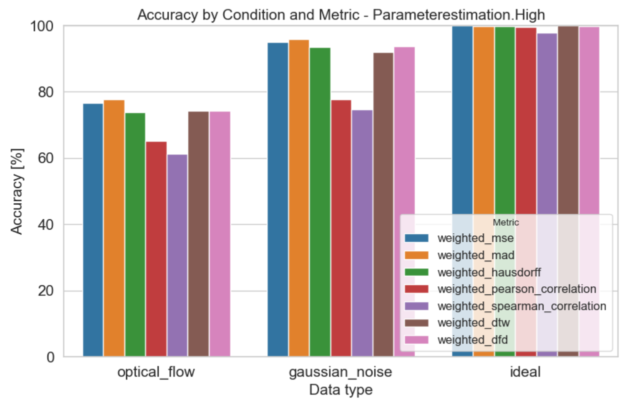


(a) Comparison accuracy of various similarity metrics for the parameter estimation level low.

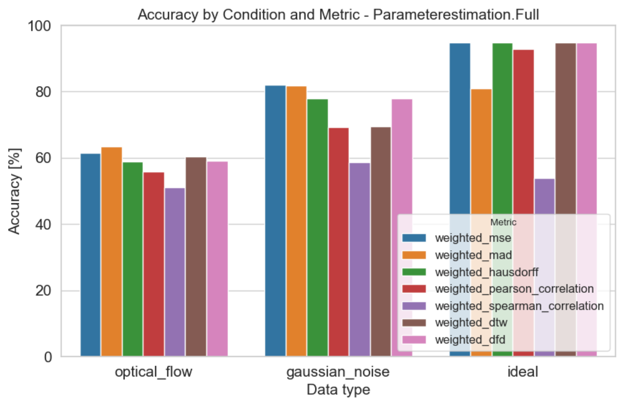


(b) Comparison accuracy of various similarity metrics for the parameter estimation level medium.

Fig. 36: Histograms of the various similarity metrics for the parameter estimation levels: low and medium.



(a) Comparison accuracy of various similarity metrics for the parameter estimation level high.



(b) Histograms of the various similarity metrics for the parameter estimation levels: high and full.

Fig. 37

The domain knowledge module must accurately reflect the correspondence between the input trajectories and the mathematical models. Since the mathematical models are made to match the ideal trajectories, the domain knowledge module must reflect that in the form of uncertainty about classifying the object to each target category. For instance, if the input object is a fish, in the ideal case, the mathematical model for fish should match perfectly, while the plastic model should not, resulting in a confidence of 0.95 and 0.05 for fish and plastic, respectively. However, for the optical flow case, some trajectories are not extracted properly and are wrong. In this case, the confidences should be low to indicate to the neural network that these trajectories are of low quality. Since Gaussian noise follows the same trend as the ideal trajectory, just augmented with noise, it is expected that a similar probability distribution as the ideal data is present.

The first similarity metric evaluated is the Discrete Fréchet distance, detailed in Fig. 38. For the optical flow case, the probabilities are centred around 0.5, as expected, although the presence of some values at the extremes would also be anticipated.

A significant difference can be observed between Gaussian noise and the ideal case. This indicates that the Discrete Fréchet distance is not the best metric. Both the dynamic time warping and the Hausdorff coefficient produce similar probability distributions. In contrast, the mean average distance presents a different distribution. Perfectly matching the expectation for the Gaussian noise and the ideal case, but missing the distribution around 0.5 for the optical flow. The final metric evaluated is the mean squared error, which adheres to the expected distributions for the optical flow, Gaussian noise, and ideal data. This metric is selected as the most optimal for embedding information into a neural network.

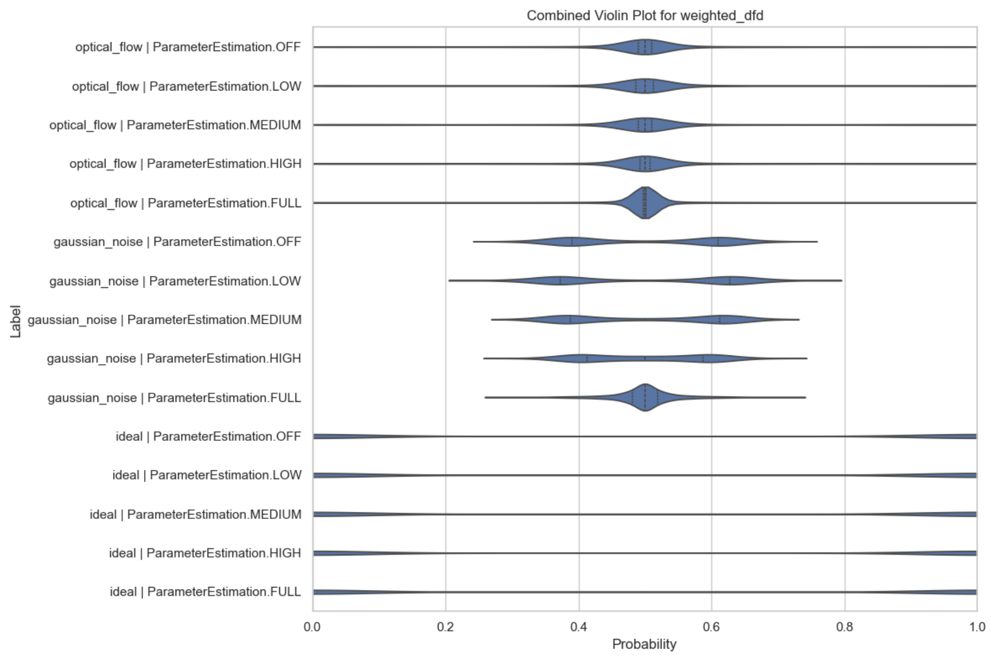


Fig. 38: Violinplot comparing the different parameter estimation levels per data type for the Discrete Fréchet distance.

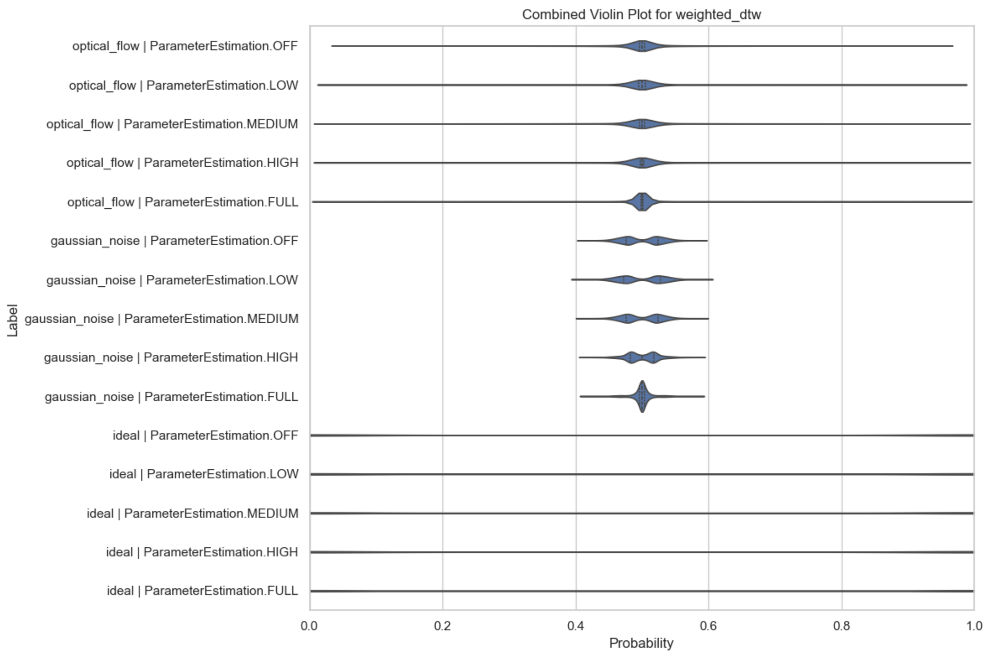


Fig. 39: Violinplot comparing the different parameter estimation levels per data type for the Dynamic time warping.

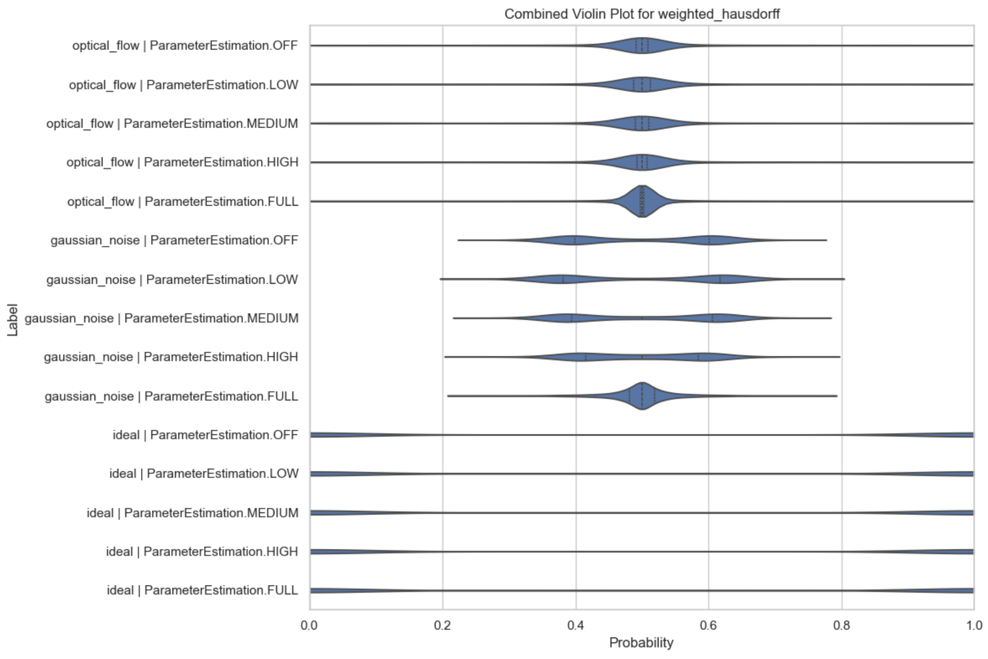


Fig. 40: Violinplot comparing the different parameter estimation levels per data type for the Hausdorff coefficient.

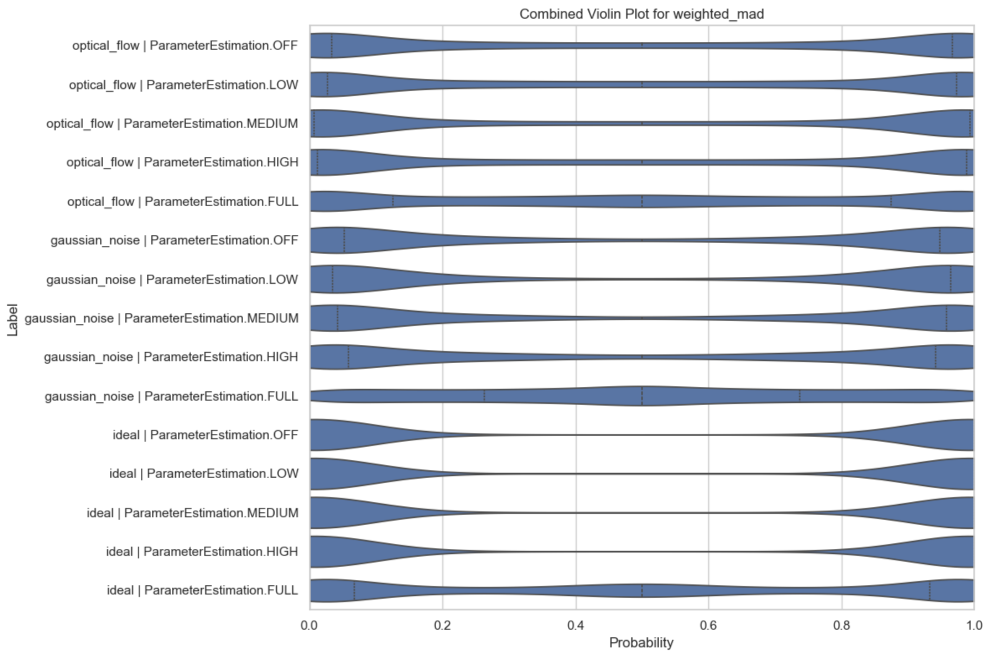


Fig. 41: Violinplot comparing the different parameter estimation levels per data type for the mean average distance.

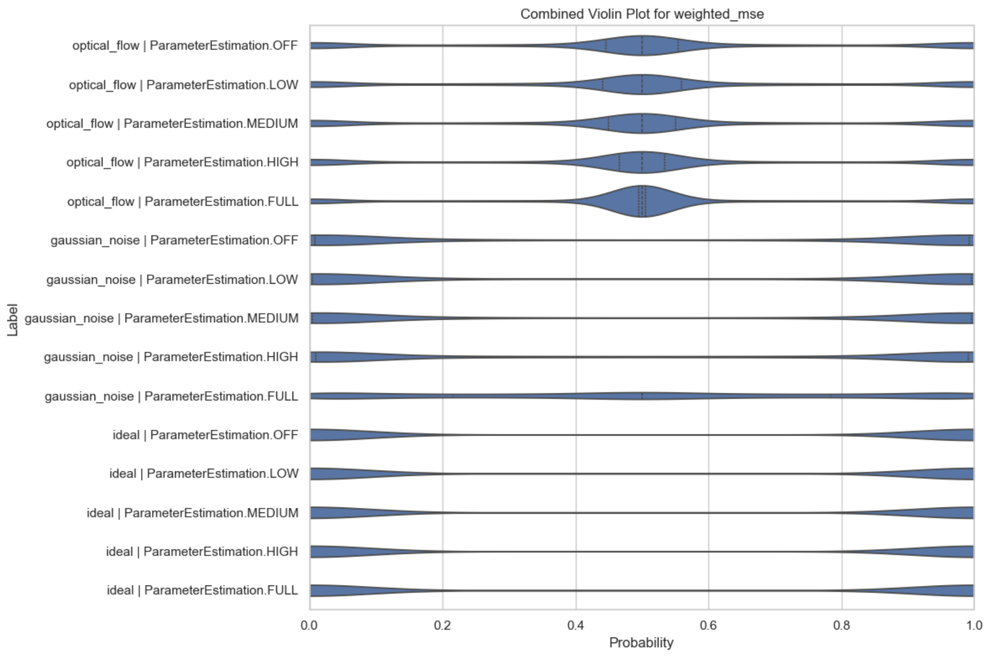


Fig. 42: Violinplot comparing the different parameter estimation levels per data type for the mean squared error.

Appendix E: Neural network training

The training of the neural network is where all the components come together and is controlled by the main script `train_trajectory.py`. This script begins by extracting all the values from the `parsed_data.json`, which are then used to determine the bounds for all of the parameters in the differential equations. Subsequently, this script initializes the training loop for the k-fold cross-validation. In each iteration of the k-fold, the training with domain knowledge and the one without domain knowledge are initiated. A single training run is managed by the script `neural_network/main.py` which starts by initializing the loss functions and parsing the input data. A custom dataset iterator is built to ensure that the different kinds of data are accessible during the training. The default training runs for 50 epochs but has an earlier stopping mechanism. If the validation loss increases for 6 consecutive epochs, the training process is halted. During the first epoch, the output from the domain knowledge module is saved to avoid redundant calculations in subsequent epochs. Both losses are calculated and combined using a weighting factor α. This value is kept constant at 0.5. For each epoch, the loss is calculated on both the training and validation sets, with the model tracking the lowest validation loss. At the end of the training process are the weights of the epoch corresponding to the lowest validation loss selected as optimal. These weights are then applied to the test set to evaluate the final performance.

During development, it is crucial to ensure that all data is logged accurately. In machine learning applications, this task is notoriously complex with the many different samples and loops present. A custom logging module is written for this specific purpose. Each experiment comprises 10 k-fold sets, each conducted with and without domain knowledge training, and each runs for 50 epochs unless terminated early due to the early stopping criterion. An example of one such training process is illustrated in Fig. 44. The solid lines indicate metrics on the train set, the dashed lines are from the validation set and the stars are from the test set. These stars correspond to a single value, as they are computed using the weights from the epoch with the lowest validation loss. Epoch 4 is the epoch with the lowest validation loss. The blue lines indicate the accuracy of the trained neural network, while the constant orange line is the accuracy of the domain knowledge module, which remains static over time. The red and green lines indicate the number of samples that the domain knowledge module and the neural network, respectively, have correctly predicted.

This metric is plotted to investigate if the neural network and mathematical model make different predictions. For instance, if both the domain knowledge and the neural network have an accuracy of 80%, the question arises: do they make the same errors? This metric demonstrates the degree of differences. If both make the same mistakes, little can be learned from each other; however, if they make different mistakes, there is a significant opportunity for mutual learning.

In Fig. 44 the loss over the training is depicted. Again, the solid line represents the training process and the dashed line corresponds to the validation. The blue line indicates the Cross-Entropy (CE) loss, and the red line represents the Kullback-Leibler (KL) divergence loss. The orange line is the combination of both losses. From these losses, it can be seen that overfitting begins to occur at epoch 4, indicated by the increasing validation loss. This behaviour is characteristic of semi-supervised learning which explains why the training loss for the KL divergence is so low. The KL divergence has already been optimized for this setting. The combined training loss is significantly more influenced by the CE loss than by the KL divergence. This supports the claim that using a dynamic alpha in semi-supervised learning has the potential to improve robustness against a poorly labelled dataset. In the supervised case, illustrated in Fig. 45, the opposite trend can be seen with the KL divergence higher than the CE loss. This training run also used a higher learning rate of 0.001 compared to the learning of 0.0001 used in Fig. 44. A lower learning rate shows a smoother loss function while a higher learning rate shows heavy oscillations.

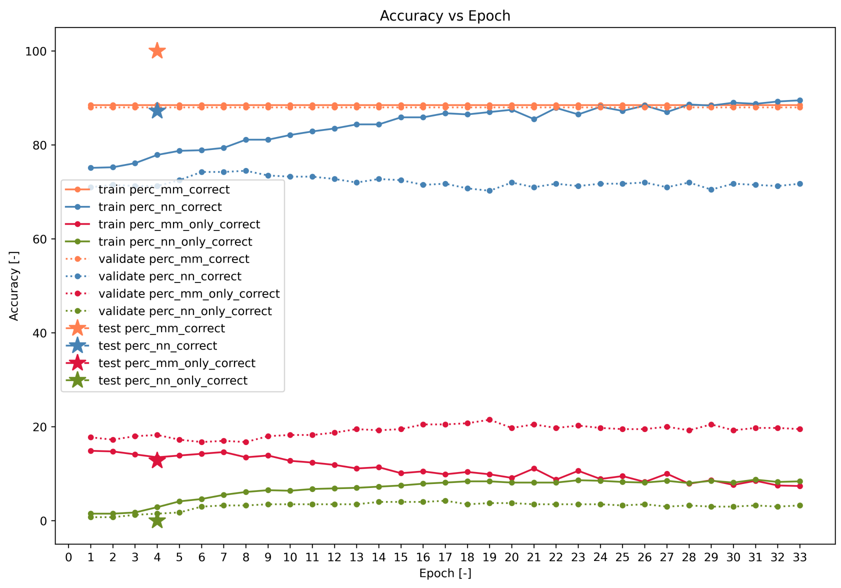


Fig. 43: Combined accuracy from a training run

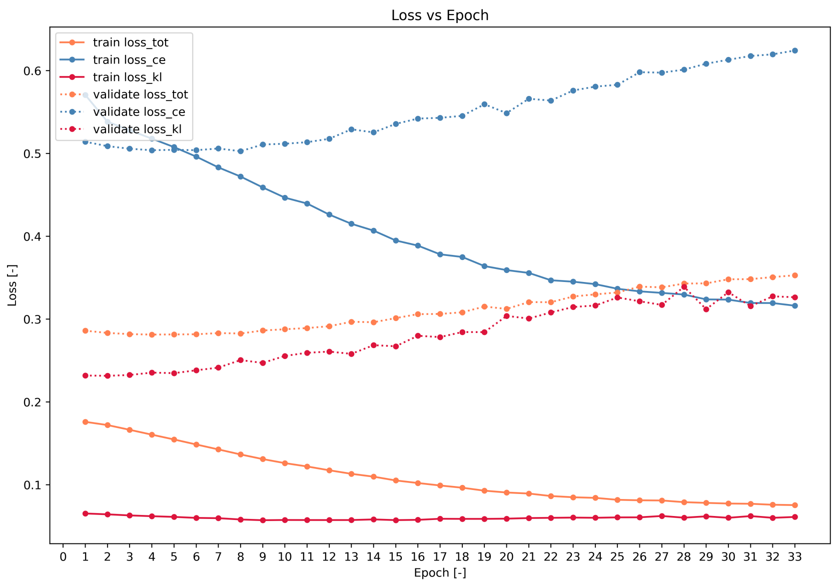


Fig. 44: Combined loss from a semi-supervised training run with a learning rate of 0.0001.

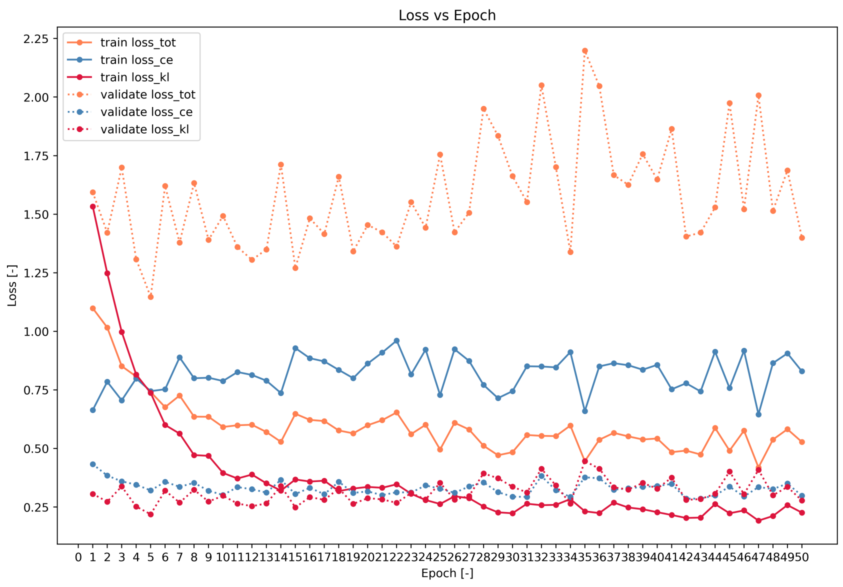


Fig. 45: Combined loss from a supervised training run with a learning rate of 0.001.

Part II

Literature study

Contents

Nomenclature

List of Abbreviations

AP	Average Precision
AUV	Autonomous Underwater Vehicle
CNN	Convolutional Neural Network
COCO	Common Objects in Context
EM	Expectation Maximization
EPE	Endpoint error
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
FSTA	Fish-school Tracking Algorithm
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
JAMSTEC	Japan Agency for Marine-Earth Science and Technology
LSTM	Long Short Term Memory
mAP	Mean Average Precision
MOT	Multiple Object Tracking
MOTA	Mutiple Object Tracking Accuracy
R-CNN	Region-based Convolutional Neural Network
RAFT	Recurrent All-Pairs Field Transforms
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
RoI	Region of Interest
ROV	Remotely Operated Vehicle
RPN	Region Proposal Network
SORT	Simple Realtime Tracking
SOT	Single Object Tracking
SOTA	State-of-the-art
SPP	Spatial Pyramid Pooling
TN	True Negative
TP	True Positive
VGG	Visual Geometry Group
YOLO	You Only Look Once

List of Symbols

α	Learning Rate
$\boldsymbol{F_m}$	State Transition matrix
$\boldsymbol{P}_{n+1,n}$	Estimation Uncertainty matrix at time n+1 after measurement n
δ	TD-error
ϵ	Exploration coefficient
η	Entropy Temperature
$\hat{\boldsymbol{x}}_{n+1,n}$	Estimation state vector at time n+1 after measurement n
$\hat{y}_i$	Predicted Value
κ	Frame number
κ	Huber-loss parameter
Ψ	Distortion risk-measure
σ_v	Total error
θ	Weights Vector
$\vec{a}_n$	Flow field parameter
$\vec{c}_k$	Motion constraint vector
$\vec{p}_n$	Component Probability Distribution
$\vec{v}$	Actual velocity vector
$\vec{x}_k$	Image location vector
ξ	Risk-distortion parameter
ξ	Total error

ξ_b	Quantification and noise error
A	Partial derivative to y of the Pixel Intensity
B	Amount of bounding boxes per grid cell
D	Depth of the image
F	Spatial size of the kernel
G	Control Matrix
H	Image height in pixels
H_m	Observation matrix
I_i	Identity matrix
I_x	Partial derivative to x of the Pixel Intensity
I_y	Partial derivative to y of the Pixel Intensity
K_n	Kalman Gain
L	Loss function
L_{gm}	Likelihood
m_n	Mixture probabilities
P	Amount of padding
Q	Process Noise Uncertainty matrix
R	Measurement Uncertainty matrix
S	Stride length
S_{gs}	Grid size
t	Time-step
u	Apparent pixel velocity in the x-axis
u_n	Input variable
v	Apparent pixel velocity in the y-axis
v_n	Measurement Noise Error
W	Image width in pixels
w_n	Process Noise Vector
y_i	Ground Truth

List of Figures

1 Introduction

The world is currently grappling with numerous environmental issues, among which marine pollution is one of the most significant [1]. Currently, 50-75 trillion pieces of plastic and microplastics are present in the oceans [2] and are estimated to increase by approximately a million tons every year [3]. Organisations such as The Ocean Cleanup are actively tackling this problem, primarily focusing on floating pollution and beach cleanups. While the issue of non-floating pollution is addressed by very few, it is estimated that over 14 million tons of such pollutants are residing on the ocean bed [4]

Plastic constitutes the majority of all marine litter, and it does not decompose but rather degrades into smaller pieces known as microplastics. It is proven that plastics in the oceans are not only harmful to humans and marine life, but also have a significant financial impact. The problem of plastic pollution is estimated to cost the global economy over 13 billion euros annually [5]. If current trends persist, it is projected that there will be more plastic in the ocean than fish by mid-century [6].

Pollution in the water column and on the seabed is a pressing issue that demands immediate attention. The longer this problem persists, the more challenging it becomes to address. Removing objects larger than 10 cm is significantly easier compared to cleaning up plastics that are only a few millimetres in size. The resources required and the complexity of the problem increase exponentially when dealing with microplastics.

This project is part of the SEACLEAR initiative [1], a collaboration between universities to enable autonomous trash cleanup.

1.1. Research Objective

Given the scale of plastic pollution and the urgency of addressing it, the most efficient solution lies in the use of autonomous technologies. Significant advancements have been made in the field of autonomy such as self-driving cars, autopilots, etc. However, research on underwater autonomy has been limited due to the harsh underwater environment which presents a unique set of challenges. SEACLEAR is a pioneering project that aims to use autonomous robots for underwater litter collection. The initial step in this process is the autonomous detection and tracking of litter. The field of computer vision has evolved rapidly in recent years, with substantial advancements enabling complex tasks such as autonomous driving, surveillance, wildlife monitoring, and facial recognition [7]–[9].

Computer vision encompasses various fields such as object detection, classification, and segmentation. For litter collection, identifying the location of objects is essential. This work focuses on the task of underwater object detection. Due to the nature of the underwater domain, object detection is a difficult task. Object shapes and light propagation get distorted in the water medium [10]. The quality of the images is highly dependent on the water quality and clarity. Most objects are situated on the ocean bed, and capturing images or videos of these objects often disturbs sediment, which then floats and significantly degrades image quality. The background is also a blurry body which distorts perspectives and decreases the colours and shapes [11]. Additionally, the presence of marine snow increases the difficulty of underwater object detection as it clutters the image and leads to many false positives (i.e. wrongfully classifying

[1] https://seaclear-project.eu/

marine life as litter). Underwater currents introduce turbulence, causing ROV shaking and resulting in unstable footage. Altogether, these factors create a highly challenging environment for computer vision.

Detecting trash presents a complex problem due to the high variability in the objects. Trash can consist of various materials, shapes, colours and sizes. For instance, a fish and a piece of trash might be similar in size and shape, although their colours may differ. Conversely, another object might share the same colour as the fish but differ in size and shape. This minimal distinction between classes (plastic and fish) significantly complicates the detection process. Furthermore, objects found in the underwater environment may exhibit varying degrees of degradation. The more degraded an object is, the more it blends into the underwater ecosystem, making detection even more challenging. Examples of problems associated with underwater vision can be observed in Figure 1.1.

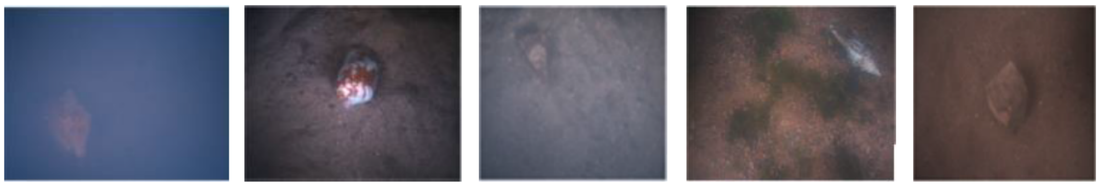


Figure 1.1: Some examples of elements present in the marine environment that affect the use of vision underwater. (a) Water with high turbidity, (b) Uneven illumination, (c) Low contrast, (d) Complicated underwater background, and (e) Monotonous colour [12]

Objection detection has come a long way and the performance of current State-of-the-Art models is truly remarkable [13] [14] [15]. Many models have proven to accurately detect objects in an underwater environment and these will be discussed in the following sections.

1.1.1. Image Object Detection

Image object detection can be categorised into two main types: single-stage neural networks and multi-stage neural networks. The current state-of-the-art (SOTA) models of these classes are the YOLO family (the latest model is YOLOV8) and Faster R-CNN. Both the YOLO and R-CNN families have demonstrated strong performance in object detection. The following paragraphs review the literature on underwater image object detection. As datasets are the most important part of computer vision, the paragraphs are organised according to the datasets used.

One of the most extensively used and largest underwater trash datasets available is the JAMSTEC (Japan Agency for Marine-Earth Science and Technology) Deep-sea Debris Dataset [16]. A portion of this dataset is annotated by the authors in [17] and named the part that is annotated ”Trashcan”. The Trashcan dataset comprises 7,212 images across 22 classes, including crabs, fish, wood, and cups. The dataset is compiled from nearly 1,000 videos. The authors of the Trashcan dataset employed a Faster R-CNN model on the dataset to achieve benchmark results of 55.4% Mean average precision with at least 50% overlap between the annotation and prediction (mAP_{50}) using all the available classes.

Another study, using all 22 classes of the Trashcan dataset, reported an mAP_{50} of 65.0% with a modified version of Mask R-CNN [18]. A separate research effort attained an mAP of 81% using the Faster R-CNN model on an earlier version of the Trashcan dataset, known as Trash-ICRA19, which includes only 5,700 images. This result, however, was based on just three classes: plastic, bio, and ROV [19]. The Trash-ICRA19 is also utilized by the authors of [20]. Their study optimized a YOLOV5 architecture for run time and achieved an mAP of over 98.4%. The dataset consists of sequential images, but a random split was adopted. The issue with applying a random split to sequential images is the small variation between frames. If frame 1 is included in the training set, frame 2 in the validation set, and frame 3 in the test set, the minimal differences between these frames mean that the performance on the test set may not be representative when different footage is used.

Another model by [21] trained on the latest Trashcan dataset achieved a mAP of 90.6% using an improved YOLOV5 architecture, also, only 3 classes were used. The paper [22] uses YOLOV3 with a ResNet50 backbone and achieves 83.4% mAP_{50} across seven classes: cloth, fishing net and rope, glass, metal, natural debris, plastic, and rubber.

In another study, a new classification model architecture was developed to detect seven classes of trash, achieving an F1 score (shown in Equation 1.1) of 0.946 [23]. However, this model did not include object localisation, reducing its applicability for ROVs. The authors of [24] used a binary classification approach and reported an mAP of 98.15% on the previous version of Trashcan with YOLOv3. The newer YOLO models, YOLOV6 and YOLOV7, have been applied to the Trashcan dataset in [14], where an mAP of 77.6% is reached across 4 classes. Comparisons among YOLOv5, YOLOv6, and YOLOv7 indicated that YOLOv6 performed best with default settings [14].

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1.1}$$

Besides the widely used Trashcan dataset, also custom datasets are used to train underwater detectors. The study described in [25] trained a YOLOV4 network on a custom dataset of underwater trash with three classes and tested it on a small ROV in a tank where an mAP of 82.7% is achieved. This result demonstrates the significant potential of using ROVs to detect underwater trash. Another study utilised a modified U-Net architecture for image segmentation of underwater trash and reached a mAP of over 94% [26]. Although this is a very impressive result, it should be noted that the custom dataset consists of images taken in a pool with clear water and a clean bottom, conditions that are unrepresentative of a realistic natural aquatic environment. Therefore these results cannot be generalised to a real-world application. In addition, only 3 classes were used making this not directly comparable to other studies.

The authors of [27] developed a YOLOv3 model to detect underwater marine life and achieved an mAP of 69.6%. Another study annotated a new dataset based on the large JAMSTEC database and claimed to have higher results than benchmark results provided by the authors of the original Trahscan dataset [28]. A Mask R-CNN is applied to the new dataset named CleanSea corpus and has achieved an mAP of 59.7% and 67.1% on all classes and material-grouped classes respectively. Another study utilised a custom low-quality underwater trash dataset with three classes to train a custom classification algorithm [29]. The authors have reached an accuracy of 86% using transfer learning. This shows that it is possible to reach good results on very low-quality images, but no localization is performed thus limiting the application for autonomous cleanup.

The model described in [30] was trained using a portion of the JAMSTEC dataset [16], achieving an accuracy of 82% with three classes. Other underwater detectors, such as the one in [31], have achieved an mAP of over 91% on a six-class fish dataset comprising approximately 6,000 to 7,000 images.

It can be concluded that the achieved mAP has a direct relation with the number of classes used. The highest performance is observed when the number of classes is minimal. Detection models with a small number of classes have demonstrated high accuracies, approaching 100%, leaving little room for improvement. However, the high accuracies are all achieved on either the Trashcan dataset, its variations or in a controlled environment with a limited number of classes.

1.1.2. Video Object Detection

Video object detection is less prevalent than image object detection. The Trashcan dataset [17] is annotated and split in video frames, making it a viable option for video object detection. However, the objects are not assigned IDs and are thus not tracked across frames. To the best of the author's knowledge, no research has focused on video recognition using the Trashcan dataset or underwater videos.

A more extensively researched area in the same field is underwater fish recognition. In [32], a combination of Gaussian mixture models (GMM), optical flow, and a YOLO model achieved good results, with accuracy exceeding 90%, resulting in a robust detector. This approach leverages different techniques with different strengths and weaknesses. Optical flow performs well in detecting every small difference between frames but suffers a high number of false positives. GMM has fewer false positives but often misses the small objects and YOLO performs very well in a static environment. This combination of techniques has great potential for the application of litter and marine life detection as it adds temporal information to the image information.

The authors of [33] employed a technique to detect periodic movement using a colour threshold tracker to identify regions coupled with a Fourier tracker. This study mentions the important note that image detection, especially in the underwater environment, is not able to detect everything in each scenario. By adding a temporal approach to image detection, it complements the pitfalls of object detection. Similarly, [34] notes that relying solely on image detection is challenging and that incorporating temporal data significantly simplifies the task.

1.1.3. Conclusion

It has been demonstrated using the Trashcan dataset that high results can be achieved in underwater trash detection when using a small number of classes and controlled scenarios. If more training data were available, these state-of-the-art networks could potentially be improved to detect a larger number of classes. However, the question remains how well these models generalize if they are applied to different underwater environments with various types of objects.

Similar underwater objects can have very different shapes and outlooks due to the nature of the environment. The movement of litter and marine life in water leads to blurry footage. For example, a rigid piece of plastic might look a certain way at a certain timestamp, but after a while, this piece will degrade and overgrow with algae causing it to look very different than before. This results in a need to have a dataset containing objects at various stages of their lifetime.

The limits are being reached in how much information can be extracted from a single image. If current state-of-the-art underwater trash detection models are applied in real-world environments, they often fail to make accurate detections when objects blend with the environment.

In such cases, even humans struggle to make accurate predictions. However, the advantage a human brain has over an image detection model is the presence of domain knowledge which aids in understanding and interpreting complex scenes. To enable high-accuracy underwater trash detection, domain knowledge must be used to fill in the gaps. Domain knowledge in underwater trash detection has never been utilized to the knowledge of the author.

As demonstrated in [32], the temporal domain unlocks additional information beyond the pixel values extracted from a single image. Authors in [35] have shown that adding the temporal domain greatly improves the accuracy. The writers of [33] have used a temporal tracking model looking at the periodic movement of flippers to follow divers underwater. This is combined with colour cues to make it more robust. These colour cues could be replaced by convolutional neural network (CNN) modules to improve detections. The periodic tracker exemplifies the value and potential of incorporating the temporal domain, further underscoring its importance in improving underwater trash detection models.

1.2. Domain knowledge

As discussed in the previous sections, integrating domain knowledge into detection systems has significant potential to improve accuracy. This section provides a high-level overview of the domain knowledge relevant to underwater environments, focusing on its application to trash and marine life detection.

1.2.1. Underwater Environment Knowledge

Lighting

Water is a different medium than air which results in a different behaviour of light underwater. Light is more refracted and absorbed, causing objects to appear significantly different over time. Understanding the principles of light absorption and scattering in water can be leveraged to greatly improve image quality.

Setting

An understanding of the underwater environment, such as the spatial distribution of objects, can be highly beneficial. For instance, if it is known that a certain location contains abundant sea grass beneath which trash frequently accumulates, this information could be used to specifically calibrate the Convolutional Neural Network (CNN) for these conditions. The image quality can vary a lot underwater and many errors can occur such as bubbles/dirt being stuck on the camera. Bubbles can stay in front of the camera causing false detections as their appearance underwater is very similar to that of plastic pieces. For a human without temporal information, it is impossible to determine if it is a piece of plastic or not. An example of this can be observed in Figure 1.2.

Figure 1.2: Dubious piece of litter

Figure 1.3: Difficult to capture marine life

Hydrodynamics
Understanding hydrodynamics is crucial for improving underwater object detection. Underwater objects often move in complex ways due to currents and other hydrodynamic factors. By incorporating hydrodynamic models to predict the movement of objects over time, the accuracy of object tracking could be significantly enhanced.

1.2.2. Object Knowledge

Features
Each object has a distinct combination of features and textures which makes them recognizable. Unfortunately, this is less the case for underwater litter as this can contain every shape, form and colour. There are some differences between man-made objects and marine life. Marine life often has colours which blend into the environment while plastics have very bright colours.

Dynamics
Knowing the motion of the objects adds valuable extra information. For example, as shown in Figure 1.3, a fish is swimming in front of the camera which can clearly be observed in the video footage but almost not in the frames.

Fish typically exhibit startled high-frequency reactions when an ROV approaches them. Embedding this behavioural knowledge into detection models can enhance their accuracy. Various studies have shown the instinctive behaviour of fish to swim against the current [36] [37]. Embedding this information into models can improve the performance. Additionally, crabs are known to move only within a 2D plane, and this dynamic can also be integrated into detection models to increase accuracy.

1.3. Research Outline

It is shown that research made significant advancements in the field of underwater image object detection. However, limited progress has been made in integrating domain knowledge to enhance detection performance. As discussed in Section 1.2, temporal domain knowledge, particularly the trajectory of objects, holds the greatest potential for improving accuracy.

Extracting the trajectory of an object from a video stream requires tracking this object across multiple sequential frames. This means that initial object detection has to be performed and then a tracking algorithm should be applied to confirm the detection. This approach allows the object detection model and the tracking model to complement each other. If the object detection model includes uncertainties about the object's position and/or class, extracting the trajectory could provide vital information for the object detector. Conversely, the object detector can enhance the tracking algorithm by providing an initial guess of the object's location and class, allowing the tracker to adapt its algorithm for that specific object.

A similar approach is described in [32] which serves as a baseline for this thesis. Both the image level and temporal information are used to detect the objects. The thesis aims to use a similar architecture with several modifications to further improve performance.

- An analysis of the state-of-the-art object detection models will be performed. This can be found in Chapter 2.
- A review of different motion extraction techniques is carried out in Chapter 3.
- A survey about the SOTA tracking algorithms is conducted in Chapter 4

For every machine learning problem, data is incredibly important. An overview of the various available datasets and techniques to create more data can be found in Chapter 5. The research question is discussed in Chapter 6, and the report concludes with Chapter 7, which contains the conclusions.

2 Object Detection Networks

With the rapid development of computer vision and deep learning techniques, neural networks have shown remarkable performance in image and video detection tasks. The ability of deep neural networks to automatically learn relevant features and patterns from images and videos has enabled significant progress in various fields, such as autonomous driving, surveillance systems, and medical imaging. This chapter focuses on different neural networks that exist for image and video detection, which include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and their variants. CNNs are widely used for image classification, object detection, and segmentation, while RNNs are applied to video classification and video object detection. Due to the novelty and data hungriness of video Transformers, these are not investigated further.

To select the best network for the application of underwater detection, it is important to understand how it is built up and how it works. Consequently, fundamental elements and layers of CNNs and RNNs are explained in Section 2.1. More specific neural network parts are described in Section 2.2. This is followed by an overview of the SOTA architectures in Section 2.3. The chapter concludes with a description of the training process and evaluation metrics commonly used in Section 2.4 and Section 2.5 respectively.

2.1. Fundamental Neural Network Layers

Neural networks consist of different layers which make up the building blocks of the network. Understanding each of these building blocks is vital in order to select the optimal architecture.

2.1.1. Convolutional layer

The convolutional layer is used to extract features from an image. Features can be viewed as the visual building blocks of an object. These can include edges, corners, colour gradients, textures, shapes, and other visual cues characteristic of the object. The convolutional layer typically contains two types of matrices. The first type is the input matrix. In the application of object detection, this input matrix or 2D array is a matrix of the pixels which is named the input feature map. The second type of matrix is the kernel (or filter). This is a matrix which consists of learnable parameters and is spatially smaller than the input feature map. Although the kernel is spatially smaller than the receptive field (or input feature map), it is often deeper.

The two types of matrices interact with each other by using the convolution operation and a sliding window approach. The kernel slides over the input feature map. The length the kernel slides every step is called the stride length. At every step, the kernel is convoluted with a part of the input feature map by using the convolution operation. This produces an activation map. Note that the number of channels of the input feature map must be the same as the number of kernel channels. If the input image is of the RBG format, then the input size is W (width of the image in pixels) x H (height of the image in pixels) x D (depth, in the case of RBG = 3). In the RGB example, the kernel must have 3 channels as well. The different activation maps are stacked (summed) such that the output has the same depth as the amount of filters. A visualization can be found in Figure 2.1 [38] [39].

$$W_{\text{out}} = \frac{W - F + 2P}{S} + 1 \tag{2.1}$$

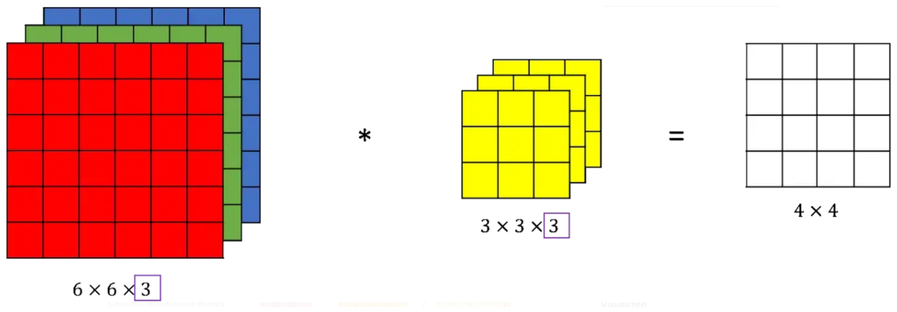


Figure 2.1: Convolution of a 3 channel image [38]

The output dimension can be calculated using Equation 2.1 where W is the width, F is the spatial size of the kernel, P is the amount of padding (the amount of 0 units added on the edge) and S is the stride length. Often more than one feature needs to be extracted from the input image and multiple kernels are used. Note that all the kernels must have the same depth as the original image. The output of the convolution operation is the feature output map with a depth equal to the number of kernels used. The last step in the convolutional layer is applying the activation functions. These functions are required to introduce non-linearity into the network. Without these functions, the entire network would be linear which does not represent the physical world. The activation functions transfer the output of one neuron to another, similar to what happens in a human brain [40]. These functions do not alter the shape of the feature maps and more information can be found in Section 2.1.7. A bias value is added to the feature maps before the activation functions. The output is then referred to as activation maps with a depth equal to the number of kernels. A schematic overview can be found in Figure 2.2.

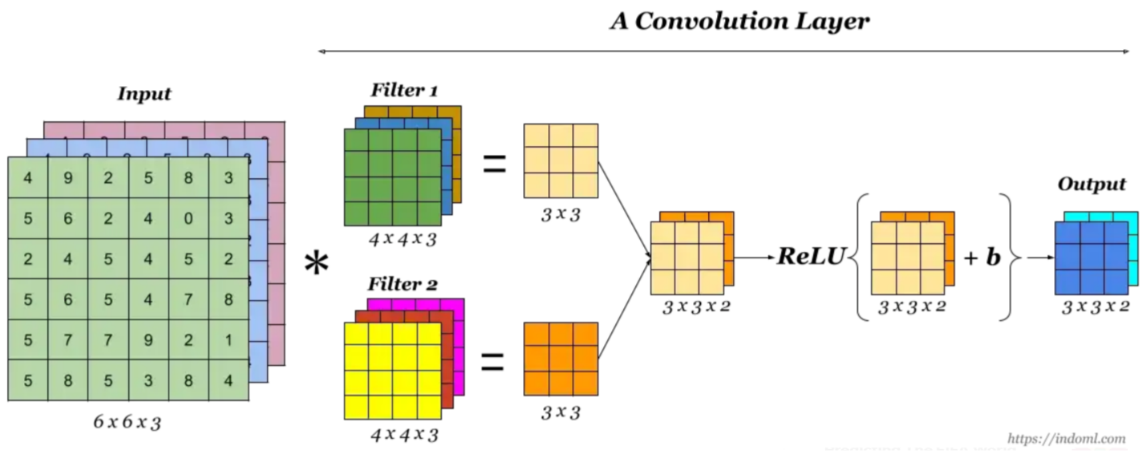


Figure 2.2: Overview of all the steps in a convolutional layer [39]. ReLu equal Rectified Linear Unit

2.1.2. Pooling layer

The pooling layer is almost always implemented after the convolutional layer with the main purpose of increasing robustness and reducing the size of the network. This layer works similarly to the convolutional layer with sliding windows and kernels. The kernel slides over the input array and extracts the maximum value or takes the average of the values, which is called max-pooling and average-pooling respectively. The outputs of this layer are feature maps, similar to the input but with reduced size and only the most prominent features are captured. The output of this layer is called the pooling map where the spatial dimension is reduced but the depth is preserved [41].

2.1.3. Fully connected layer

This layer often comes after a convolutional neural network. It takes the flattened feature maps as input. The output is a 1D array of features where all the input neurons are connected to all the output neurons.

2.1.4. Spatial Pyramid Pooling (SPP) Layer

In deep neural networks, a fixed input size is only required for the classifier or fully connected layer. Convolutional layers, on the other hand, are flexible and can accommodate any input image size. The Spatial Pyramid Pooling (SPP) layer is typically used after the last convolutional layer. It extracts spatial bins proportional to the input size, producing a total of $k \times M$ vectors, where k is the number of kernels and M is the number of bins. The SPP divides the feature map into multiple non-overlapping bins of different sizes, which are then max-pooled into a fixed-length feature vector.

2.1.5. Batch Normalization Layer

The batch normalization layer is applied to normalize the activations in a layer. As the output of one layer is the input of another, often the values can spread a wide range. Batch normalization normalizes all the values to make sure the distribution of values across all the layers stays within bounds. This ensures that the training process is stabilized and makes the training less dependent on the initial values.

2.1.6. Residual block

A residual block consists of a stack of layers. Typically these layers are a combination of convolutional layers, batch normalization layers and pooling layers. The input x of a residual block is the feature maps produced by the previous layer. The information flows in two ways: one way is through the residual block of stacked layers. At the output of this flow, a non-linear activation function is added to introduce non-linearity. The output of this layer is called $H(x)$ with $H(x) = F(x) + x$ and $F(x)$ the activation function. The second flow is a shortcut from the input directly to the output. The shortcut has two different options: identity mapping shortcuts or projection mapping shortcuts. The identity mapping shortcuts introduce a shortcut where the input x is directly connected to the output. The projection mapping shortcuts are introduced for the cases where $H(x)$ has a different shape than x. The disadvantage of the projection mapping is that it introduces extra parameters, while the identity mapping adds no extra parameters. The two flows are added at the end of the block, making the output equal to a sum of $H(x)$ and x. A graph overview is shown in Figure 2.3.

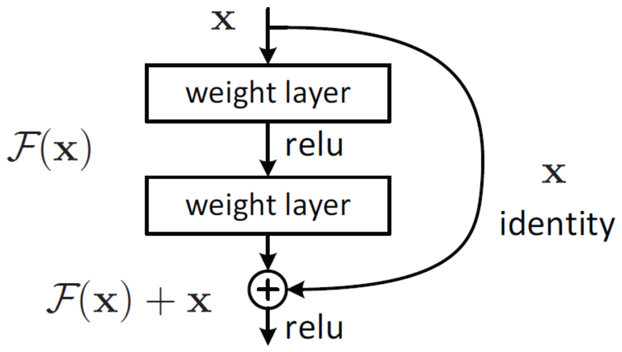


Figure 2.3: Overview of the residual block [42]

2.1.7. Activation functions

Activation functions are functions that determine what neurons will be activated. They are important to introduce non-linearity into the network. The activation functions have an effect on the vanishing and exploding gradients, computational resources required and training time. The most commonly used activation function is ReLU (Rectified Linear Unit), a piecewise linear function that outputs the input directly if it is positive; otherwise, it outputs zero. The activation function used in the last layer of a multiclass model is typically the softmax function. This function computes the probability distribution from the output vector of real numbers of the previous layer. Activation functions also have the capability to bind the output within a specific range. Various activation functions are applied to the output feature maps to achieve different effects [40].

2.2. Specialized Components of Neural Network Architectures

The previous section described the most fundamental layers of neural networks. This section will discuss different concepts which are used in various SOTA models.

2.2.1. Region Proposal Extraction

Region proposal extraction is a technique applied to the input image to identify regions of interest (RoI). These regions are then fit into the object detector. Using region proposals significantly reduces the search space, improving the efficiency and accuracy of object detection. Various methods exist for extracting region proposals, with the selective search algorithm and the family of region proposal networks (RPNs) being the most common.

Selective Search

Selective Search aims to generate all possible object locations in an image by using exhaustive search and segmentation. It is designed with the following three principles: capture all scales, diversification, and fast to compute. A bottom-up approach is used to capture all the different scales. It starts at a small level and the grouping method keeps grouping parts together until the entire image is one group. By applying various colour spaces and similarity measures, bounding boxes are constructed. Although this method is highly accurate, it is also time-consuming and lacks customizability [43].

Regional Proposal Networks (RPN)

Region Proposal Networks (RPNs) are designed to address the high cost of generating region proposals. This approach makes use of neural networks to generate the region proposals. It uses a CNN to generate a feature map from the input image, then a sliding window is applied to this feature map to generate proposals. By default, this sliding window considers a maximum of three scales and three aspect ratios for the anchors, resulting in $k = 9$ anchors for each sliding window position. These anchors are translation-invariant.

The output generates two layers: the box classification layer (cls) and the box regression layer (reg). The box classification contains a binary variable p which is the predicted probability of an anchor being an object. $p = 1$ indicates a high probability and $p = 0$ indicates a low probability. The overlapping of boxes is addressed using non-maximum suppression (NMS) on the proposed regions based on their cls scores. This results in few but accurate proposals. The box regression is only applied to objects with $p = 1$. The regression layers output the bounding box coordinates.

The cls and reg layers are normalized and weighted in the loss function. Cls is normalized by the min-batch and reg by the number of anchor locations. This approach generates high accuracies on datasets such as the Common Object Context (COCO) dataset [44] and the Pascal Visual Object Classes (VOC) dataset [45].

COCO is a large-scale object detection, segmentation, and captioning dataset. It contains over 330,000 images, each labelled with object bounding boxes and object segmentation masks captured in various environments. The VOC dataset is a widely used benchmark dataset for object detection and segmentation challenges in computer vision. The dataset consists of images containing various objects such as people, animals, vehicles, and common household objects, labelled with bounding boxes and masks.

The strength of RPNs lies in the increased speed. This approach enables the initial feature map to be shared between the region proposal network and the detection network, greatly reducing the number of parameters involved and increasing the training speed. The training involves several steps:

- RPN training: The first step focuses on training the RPN to generate region proposals.
- Training the Detection Network: In the second step, the detection network is trained separately from the RPN.
- Layer Sharing: In the third step, the layers between the RPN and the detection network are shared. During this stage, only the layers unique to the RPN are fine-tuned.
- Fine-Tuning the Detection Network: In the final step, the layers unique to the detection network are fine-tuned, resulting in a unified network.

The staged training process ensures that both networks are optimally tuned for their respective tasks while maintaining a high level of integration and efficiency.

2.2.2. Backbones

A backbone network is a pre-trained network used to extract features from the input image. Multiple backbones are available, with the VGG [46] and Resnet backbones [42] the most popular.

VGG

The VGG model, which stands for ”Visual Geometry Group” is a neural network used as a backbone in various models. It employs very small receptive field sizes (3x3) throughout the network. The network makes use of the rectification function to introduce non-linearity. Multiple different versions of the network exist with the main difference being the different number of layers. It always has three fully connected layers and a variable number of convolutional layers [46].

Resnet

Resnet is a very deep neural network making use of residual blocks. These are detailed Section 2.1.6. Typically, very deep neural networks suffer from the issue of vanishing gradients. Thanks to the use of residual blocks, this problem is mitigated. ResNet also incorporates batch normalization layers, which further stabilize and accelerate the training process. During training, the residual function is defined as $F(x) = H(x) - x$, where $H(x)$ is the output and x is the input. This residual mapping, which represents the difference between the output and the input, enhances the network’s robustness to changes in the input during training. ResNet has demonstrated high performance, even with depths of up to 100 layers [42].

2.2.3. Bounding Box Regressor

A bounding box regressor is used to refine and improve the localisation of bounding boxes. Several different regressors are available, with one of the best-performing methods being bounding box regression with uncertainty, which employs KL Loss [47]. The first step is bounding box parameterization where the bounding box coordinates of the diagonal vertices are expressed as (x_1, x_2) and (y_1, y_2). A probability distribution is calculated instead of only bounding box locations. For simplicity, a simple single-variate Gaussian function is used. Note that it is assumed that the bounding box locations are independent. The ground truth is also represented with the Gaussian function, but here the variance goes to 0, effectively becoming a Dirac pulse. To find the difference, the KL divergence is used. This is a method to calculate the difference between the probability function of the prediction and the ground truth. This method works by measuring how much information is lost between the ideal distribution (ground truth) and the model prediction (predictions).

2.2.4. Non-maximum suppression (NMS) algorithm

The non-maximum suppression step is essential to remove multiple detections of the same object. Originally the GreedyNMS is used as a separate step to solve this problem. It is a simple algorithm utilizing the confidence and proximity of the boxes. Although GreedyNMS performs well in many cases, it tends to decrease recall and precision when objects are very close together. An alternative non-maximum suppression technique often used is learning non-maximum suppression named GossipNet [48]. This method communicates with the neighbouring detections and iteratively reduces the number of bounding boxes until one remains per object, providing a more robust solution in scenarios with closely packed objects.

2.3. Neural Network Architectures

In the current computer vision landscape, a variety of different architectures can be found. These can be grouped into different families: multiple-stage convolutional neural networks, single-stage convolutional neural networks and recurrent neural networks.

2.3.1. Multiple Stage Convolutional Neural Networks

The most common architectures are:

- Fast R-CNN
- Faster R-CNN

Fast R-CNN
This architecture is quite straightforward. The first step involves extracting the region proposals using selective search (Section 2.2.1). Consequently, the CNN is run on all the proposed regions after resizing and warping the regions to match the desired input size. In the final step, a greedy algorithm is implemented that checks detections for a high Intersection over Union (IoU) to prevent duplicate detections [49] [50].

Faster R-CNN
The Faster R-CNN is an upgraded network that addresses the limitations of Fast R-CNN. While Fast R-CNN makes use of the slow selective search as a region proposal extraction method, Faster R-CNN utilizes a region proposal network. This is a much faster method and it is proven to also have a slight increase in accuracy compared to Fast R-CNN. The architecture can be observed in Figure 2.4. Instead of running the CNN on each region proposal, it runs the CNN once on the entire image, extracting region proposals from the resulting feature map. These region proposals are then used by the Fast R-CNN detector module. The final steps involve applying bounding box regression and non-maximum suppression. The training process for Faster R-CNN consists of four steps, starting with training the RPN separately, followed by training the unique CNN layers. Thanks to its high accuracy and fast inference time, Faster R-CNN has become one of the most widely used architectures in object detection [15].

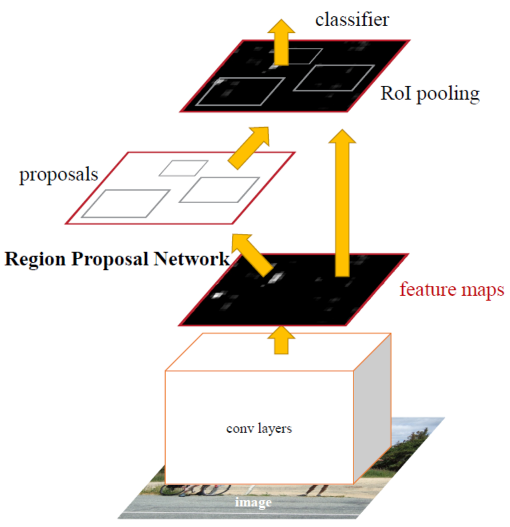


Figure 2.4: Architecture of Faster R-CNN [15]

2.3.2. Single Stage Neural Networks

Single-stage neural networks work, in contrast to multiple-stage neural networks, by only passing once over the image without needing a separate region proposal step. The most well-known group of single-stage neural networks is the YOLO family.

YOLO - Original
The original YOLO (You Only Look Once) architecture is a simple yet fast model that revolutionized the landscape of object detection. It splits the image up into grid cells and each grid cell is responsible for detecting the objects which fall in it. Each cell predicts a fixed number of bounding boxes. In the original YOLO, this number is set to 2. The total output vector size is shown in Equation 2.2, where S_{gs} is the grid size, B the amount of bounding boxes per grid cell, 5 is the number of parameters per bounding box (centre x-coordinate, centre-y coordinate, height, width and confidence (value between 0 and 1 which indicates how confident the model is in the prediction) and C is the conditional class probability vector and its size equals the number of classes.

$$\text{size} = S_{\text{gs}} \times S_{\text{gs}} \times (B \cdot 5 + C) \tag{2.2}$$

Often many grid cells do not contain any objects and they are filtered out based on confidence levels. NMS is also applied to eliminate double predictions. The YOLO architecture has proven to be much faster than all the other SOTA models at that time while also being more accurate. Due to the fixed anchor boxes approach, the model performs less on small and occluded objects [51].

YOLOV7

The YOLOV7 model slightly increases accuracy over the current state-of-the-art models and greatly increases the inference speed. The existing YOLO network is improved using different methods such as reparameterization and model scaling. It also allows to use of different backbones and has a novel loss function. All these improvements lead to a model which has a slightly higher accuracy than SOTA models but a much higher inference speed [52]. An overview of the AP vs inference speed for the SOTA models can be found in Figure 2.5.

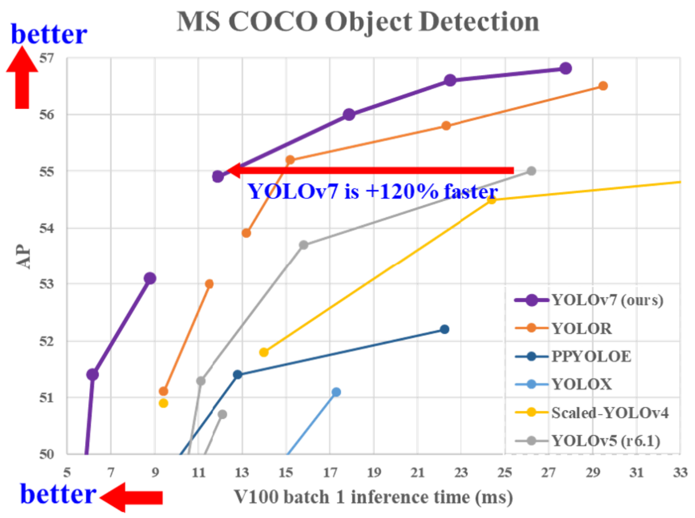


Figure 2.5: Peformance of SOTA models [52]

2.3.3. Recurrent Neural Networks

Recurrent neural networks (RNN) are designed to process sequential data where the temporal aspect is of importance. Unlike traditional neural networks, where data flows in only one direction, RNNs allow data to flow back into the neurons, enabling the network to maintain a memory of previous inputs. A common problem with recurrent neural networks is the exploding or vanishing gradient. The exploding or vanishing gradient is a phenomenon where the gradient which is used during training to update the weights either becomes very large or decreases to zero respectively. To address these problems, two popular solutions are Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks. These architectures are designed to mitigate the exploding and vanishing gradient problems, thereby improving the performance and stability of RNNs.

Long Short Term Memory Networks

Long Short-Term Memory (LSTM) networks aim to solve not only the problem of the vanishing and exploding gradients but also of the short-term memory. The LSTM has three different ”states”: the input state (x(t)), the hidden state (h(t)) which is also known as the short-term memory and the cell state (c(t)) or long-term memory. In every iteration, these three values pass through several gates to set new cell states and hidden states. These gates decide if the new inputs will be saved in the short or long-term memory and how the memories are affected. A visual representation of the states and the different gates can be found in Figure 2.6.

The first gate is the Forget Gate. This gate is a multiplication between the cell state and a combination of the hidden state and input inside a sigmoid activation function. The output of this activation function is a value between 0 and 1; the closer the value is to zero, the less important that part of the cell state is, and it can be forgotten. Conversely, the closer the value is to 1, the more it should be remembered. The second

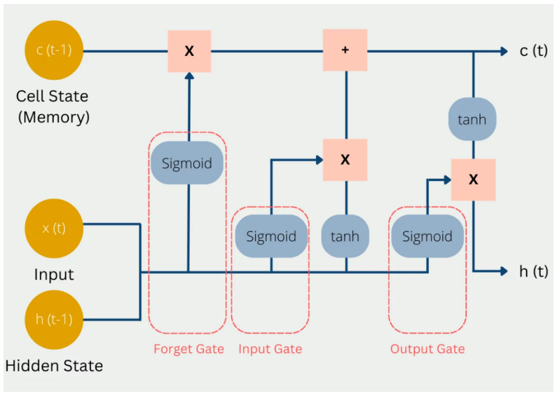


Figure 2.6: LSTM Architecture overview [53]

gate is the Input Gate which determines how important the input value is to the cell state. This time, both sigmoid and tanh activation functions are used and the output determines the significance. If it is deemed significant, it will change the cell state. The third and last gate is the Output gate. This gate multiplies the cell memory in a tanh activation function with the input in a sigmoid function to determine the new hidden state. The LSTM network is widely used in various applications with sequential data. Important to note is that these networks typically require large amounts of data to be trained [54].

Gated Recurrent Unit Networks

Gated Recurrent Unit (GRU) networks are very similar to LSTM networks with the difference that GRU networks have a simpler architecture making them faster to train. Instead of three gates, the GRU only has two gates: the Reset Gate (short term) and the Update Gate (long term). In contrast to the two states present in the LSTM networks, GRU networks only have one state: the hidden State. To learn more about this architecture, please refer to [55]. This architecture is simpler than LSTMs and it is preferred for smaller datasets.

2.4. Training of Neural Networks

An essential part of the object detection conversation is model training. As discussed before, a neural network consists of different layers and each of these layers is built up of neurons. These neurons are connected to previous and future layers with weight parameters assigned to them. These parameters are the values which decide how the network will behave and are modified for each application. The training of a neural network is thus iteratively changing these parameters to get the most optimal combination for the specific application.

A crucial part of the training process is the loss function. This is a function which compares the target output with the network output and indicates how well the network models the dataset. One of the most famous loss functions for multi-class classification is shown in Equation 2.3 where y_i is the ground truth and $\hat{y}_i$ is the predicted value.

$$L(y, \hat{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log\ (\hat{y}_i) \tag{2.3}$$

The goal of the entire training process is to minimize the loss function, making it an optimisation problem. The most commonly used optimisers are the Stochastic Gradient Descent (SGD) and the Adam optimiser.

2.4.1. Stochastic Gradient Descent (SGD)

As the name indicates, this optimisation algorithm uses the gradient descent of the loss function. The gradient descent is an iterative optimization algorithm which is used to minimize the loss function. The algorithm starts by calculating the gradient of the loss function for each parameter. The gradient can be seen as the direction to maximize the loss function. To decrease the loss, a small step is taken in the opposite direction. This is also shown in Equation 2.4, where θ is the vector of weights to be updated and α is the learning rate. The learning rate is a hyperparameter which influences the training process.

$$\theta = \theta - \alpha \cdot \frac{\partial L}{\partial \theta} \tag{2.4}$$

To find the gradient in a network with millions of parameters is not a straightforward task. The gradient is calculated using the concept of backpropagation. In simple terms, this means that instead of starting at the input, it starts at the output of the network. Fully explaining the algorithm is beyond the scope of the literature review.

2.4.2. Adam Optimiser

The Adam optimiser is widely used in literature. It is based on the SGD and incorporates two techniques to improve the results. Instead of having a fixed learning rate, an adaptive learning rate is used. The optimiser works by using biased first and second moments of the gradients. These moments are computed as the exponential moving averages of the gradients and their squared values respectively. This allows the optimiser to incorporate past gradients to adapt the learning rates for different parameters. Momentum is also included which provides inertia in the search direction to overcome local minima and noisy gradients oscillation [56].

The Adam optimizer has proven to converge faster and with fewer oscillations than SGD. It should be kept in mind that in some cases the results of SGD are superior [57].

2.5. Evaluation metrics

Evaluation metrics are essential for assessing the performance of neural networks on datasets. They quantify how well a network performs and identify its limitations. Commonly used metrics include accuracy, precision, recall, mean average precision (mAP), and Intersection over Union (IoU). The results of a model can be categorized into four groups, illustrated here using the example of trash classification:

- TP - True Positive: The model correctly predicted that an image contains trash.
- TN - True Negative: The model correctly predicted that no trash is found on the image.
- FP - False Positive: The model falsely classifies an image containing trash while it does not.
- FN - False Negative: The model classified an image with trash as an image without trash.

2.5.1. Accuracy

Accuracy is the simplest of all metrics. It is calculated by dividing the correct predictions by the total predictions. Or as in the terms defined above, the true positive and true negative are combined and divided by the total.

$$\text{acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.5}$$

2.5.2. Precision and Recall

Precision and recall are two very important metrics to evaluate the performance of a model. The precision is defined as how many of the positively identified items are actually positive and the formula can be found in Equation 2.6. The recall shows how many of the predicted results are actually correct and can be observed in Equation 2.7.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.6}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

2.5.3. Intersection over Union (IoU)

The IoU is an important metric for object detection as it shows how accurate the localization and size of the bounding box are. The metric is calculated by dividing the area of overlap by the area of union. This is displayed in Figure 2.7. The higher the value, the more overlapping is present and the better the prediction. A value of 0 indicates no overlapping and a value of 1 signifies a perfect match.

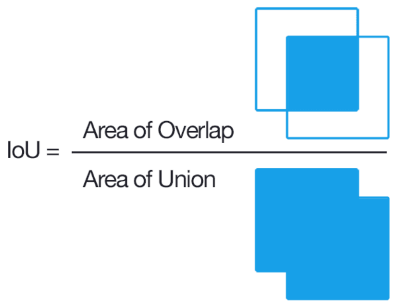


Figure 2.7: Intersection over Union [58]

2.5.4. Mean Average Precision (mAP)

The mAP is the most widely used metric to quantify the performance of an object detection network. Often the mAP is noted as mAP@0.50 or mAP@0.95 which defines to which IoU the mAP corresponds. By default, an IoU of 0.5 is used.

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N}\text{AP}_i \tag{2.8}$$

The mAP is calculated in Equation 2.8. AP_i refers to the Average Precision for every class. This value is found by calculating the area under the precision-recall curve.

2.6. Conclusion

This chapter outlined various key components of the current state-of-the-art (SOTA) neural network architectures employed in computer vision tasks. It not only presented the architectures but also explained the necessary components to fully understand neural networks. This chapter provided essential building blocks and introduced several concepts that are referenced throughout the document.

3 Motion Extraction Techniques

This chapter discusses various techniques for extracting motion from a sequence of images, often seen as optical flow. Optical flow describes the motion of objects over consecutive frames, caused by the relative movement between the object and the observer (camera). It is important to note that optical flow is not the same as the motion field. An example of this is a rotating sphere with a fixed light source. The optical flow field will be zero, but the motion field will be non-zero. Conversely, if the sphere has no movement but the light source moves, the optical flow will be non-zero and the flow field will be zero.

An image provides a 2D representation of a 3D reality, meaning that movement captured by optical flow attempts to embed 3D movement in a 2D representation [59]. This should be kept in mind while analyzing optical flow results. Three categories are discussed in this chapter: classical optical flow techniques, Gaussian Mixture Models and machine learning-based optical flow methods.

3.1. Classical Optical Flow Techniques

In optical flow, the timestamp of a specific frame is referred to as t, and the location of the pixel in the frame at time t is defined as (x, y). The goal is to compute the apparent velocity or the optical flow vector of the pixel which is expressed in the x-axis (u) and y-axis (v). The formulas are described in Equation 3.1 and Equation 3.2 respectively and are used to make up the optical flow vector **U** $= [u, v]^T$.

$$u(x, y, t) = \frac{\triangle x}{\triangle t} \tag{3.1}$$

$$v(x, y, t) = \frac{\triangle y}{\triangle t} \tag{3.2}$$

In optical flow, the brightness consistency assumption is used. This assumption states that the apparent intensity of the same object does not change across different frames. The formula which represents this can be observed in Equation 3.3 where the intensity of a pixel in a frame at time t at location (x, y) is the same as the intensity of a pixel at the next frame at a slightly different location.

$$I(x, y, t) = I(x + \triangle x, y + \triangle y, t + \triangle t) \tag{3.3}$$

By using the small motion assumption, the motion from frame to frame is deemed small. This results in the possibility to linearize I with a first-order Taylor series expansion. The truncated Taylor expansion can be found in Equation 3.4

$$I(x + \triangle x, y + \triangle y, t + \triangle t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\triangle x + \frac{\partial I}{\partial y}\triangle y + \frac{\partial I}{\partial t}\triangle t \tag{3.4}$$

If Equation 3.3 and Equation 3.4 are combined, Equation 3.5 is created and known as the Optical Flow Constraint.

$$\begin{aligned}\frac{\partial I}{\partial x}\triangle x+\frac{\partial I}{\partial y}\triangle y+\frac{\partial I}{\partial t}\triangle t=0\\ \frac{\partial I}{\partial x}\frac{\triangle x}{\triangle t}+\frac{\partial I}{\partial y}\frac{\triangle y}{\triangle t}+\frac{\partial I}{\partial t}=0\\ I_x u+I_y v+I_t=0\end{aligned} \tag{3.5}$$

The Optical Flow Constraint equation is an underdetermined system with two unknowns (u and v) and only one equation. Different methods are available to determine a second constraint that allows for solving the system. The two most popular algorithms are the Lucas–Kanade method [59] and the Horn–Schunck method [60].

The Lucas-Kanade method is the most straightforward and fastest method of the two. The degree of freedom of Equation 3.5 is also known as the aperture problem. This means that the direction of movement is not known without another equation. In the Lucas-Kanade method, this problem is solved by the spatial smoothness assumption: the neighbouring points all belong to the same optical flow **u**. An $N \times N$ area is defined around the current pixel where it is assumed that the optical flow **u** is the same. This is captured in Equation 3.6 with p_i referring to the pixel i. This formula makes the system overdetermined and the least-squares solution is found by minimizing the error. This fully constrains the problem and gives a unique solution [59]. This method works well in practice but may be subjective to noise.

$$\begin{gathered}\mathbf{A}\mathbf{u}=\mathbf{b}\\ \begin{pmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1)\\ \vdots & \vdots \\ I_x(\mathbf{p}_{N^2}) & I_y(\mathbf{p}_{N^2})\end{pmatrix}\begin{pmatrix}u\\ v\end{pmatrix}=-\begin{pmatrix} I_t(\mathbf{p}_1)\\ \vdots \\ I_t(\mathbf{p}_{N^2})\end{pmatrix}\end{gathered} \tag{3.6}$$

The second approach to constrain the system is the Horn-Schunck method which utilizes the smoothness constraint. This constraint limits the difference between the flow velocity at a point and the average velocity of a neighborhood which constraints the points. The sum of the squares of the Laplacians of the x and y components is minimized. The Laplacians can be found in Equation 3.7.

$$\nabla^2 u=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2} \quad \text{and} \quad \nabla^2 v=\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2} \tag{3.7}$$

The image is discretized along the x-axis, y-axis and frames κ. The derivatives of the brightness from the discrete set are calculated by averaging the adjacent measurements. The Laplacians are estimated using Equation 3.8 and the local average can be found in Equation 3.9.

$$\nabla^2 u \approx \kappa\left(\bar{u}_{i,j,k}-u_{i,j,k}\right) \quad \text{and} \quad \nabla^2 v \approx \kappa\left(\bar{v}_{i,j,k}-v_{i,j,k}\right) \tag{3.8}$$

$$\begin{aligned}\bar{u}_{i,j,k}&=\frac{1}{6}\left\{u_{i-1,j,k}+u_{i,j+1,k}+u_{i+1,j,k}+u_{i,j-1,k}\right\}\\ &+\frac{1}{12}\left\{u_{i-1,j-1,k}+u_{i-1,j+1,k}+u_{i+1,j+1,k}+u_{i+1,j-1,k}\right\}\\ \bar{v}_{i,j,k}&=\frac{1}{6}\left\{v_{i-1,j,k}+v_{i,j+1,k}+v_{i+1,j,k}+v_{i,j-1,k}\right\}\\ &+\frac{1}{12}\left\{v_{i-1,j-1,k}+v_{i-1,j+1,k}+v_{i+1,j+1,k}+v_{i+1,j-1,k}\right\}\end{aligned} \tag{3.9}$$

The above formulas are used to construct the error ξ_c in smoothness in the velocity flow which is described in Equation 3.10. Due to quantification errors and noise, Equation 3.5 is never equal to zero. This is set equal to ξ_b. If ξ_c and ξ_b are combined, Equation 3.11 is formed which represents the total error. The aim is to minimise the total error by finding values for the optical flow velocity (u, v). Equation 3.11 is differentiated to both u and v and set to zero. By using constrained minimization this is solved to find u and v [60].

$$\xi_c^2 = (\bar{u} - u)^2 + (\bar{v} - v)^2 \tag{3.10}$$

$$\xi^2 = \alpha^2 \xi_c^2 + \xi_b^2 \tag{3.11}$$

3.2. Gaussian Mixture Models (GMM) for Optical Flow

Gaussian Mixture Models (GMMs) are clustering algorithms that offer a probabilistic approach to clustering. Instead of K-means [61] which is a hard clustering method where each point is associated with only one cluster, GMMs are soft clustering algorithms which associate a probability to each point how close to a cluster it belongs. Due to the nature of these algorithms, they are often used to determine the optical flow by clustering pixels which have the same apparent velocity.

GMMs start from the data conservation constraint [62]. This constraint is similar to the brightness consistency assumption mentioned in Section 3.1, but more general. This data conservation constraint states that instead of intensity, the information in the image sequence is conserved locally in space and time in the direction of the motion. Equation 3.12 shows this with S_{is} being the image sequence. Limitations of this constraint are shading variation, transparency and occlusion boundaries. To cope with these limitations, outlier robustness is required.

$$S_{is}(x + dx, t + dt) = S_{is}(x, t) \tag{3.12}$$

If Equation 3.12 is differentiated, the motion constraint equation is obtained which expresses v_1 and v_2 which are the velocities in the x_1 and x_2 directions respectively. Currently, only one constraint is present with two unknowns which is insufficient to obtain a unique solution. The aperture problem arises when attempting to determine the motion direction of a contour or edge from a partial view or limited information. The motion direction of a contour or edge can vary depending on the orientation of the contour or edge relative to the observer. Without additional information, such as knowledge of the edges or other constraints, it is impossible to determine the true motion direction of the contour or edge.

Multiple motions are often present in an image sequence and similar motions can be found in different regions of the image. The concept of layers is introduced to keep the discussion streamlined. Each layer corresponds to a single consistent motion such as for example the movement of a car, the movement of a pedestrian and the movement of a bicycle which is extracted at the image level but does not say where exactly in the image each of them are present.

To fit a layered flow model to the set of motion constraint vectors which are measured within an image match, the parameter values of $\vec{a}_n$ with $n = 1, ..., N$ where N is the total number of distinct flow fields or layers (N=3 in the previously mentioned example) are searched. The motion constraint vectors refer to the measurements of motion within a region of the image or image patch. The goal is to match the motion constraint vectors $\vec{c}_k$ at each location in the image $\vec{x}_k$ with the different flow fields with parameters $\vec{a}_n$. This matching is performed using the component probability distribution $p_n(\vec{c_k}|\vec{x_k}, \vec{a_n})$. In words, this means: what is the probability that a motion constraint vector $\vec{c_k}$ at location $\vec{x_k}$ matches the flow field described by the parameter $\vec{a_k}$. The probability for outliers is $p_0(\vec{c_k})$. To get the final probability of selecting a layer, m_n is introduced which represents the mixture probabilities. Note that the sum of all m_n is equal to 1. The total equation can be observed in Equation 3.13.

$$p\left(\vec{c}_k \mid \vec{x}_k, \vec{m}, \vec{a}_1, \ldots, \vec{a}_N\right) = \sum_{n=0}^{N} m_n p_n\left(\vec{c}_k \mid \vec{x}_k, \vec{a}_n\right) \tag{3.13}$$

To solve the optical flow, the parameter values $\{\vec{a}_n\}_{n=1}^{N}$ and mixture probabilities $\{m_n\}_{n=0}^{N}$ are required. These provide a maximum likelihood to fit the data set. The log-likelihood of generating these observations from a specific model is shown in Equation 3.14.

$$\log L_{gm}\left(\vec{m},\vec{a}_1,\ldots,\vec{a}_N\right)=\sum_{k=1}^{K}\log p\left(\vec{c}_k \mid \vec{x}_k,\vec{m},\vec{a}_1,\ldots,\vec{a}_N\right) \tag{3.14}$$

For obtaining a maximum likelihood fit, the EM algorithm is used. The EM-algorithm [63] stands for Expectation Maximization which is a well-known algorithm in statistics to find maximum likelihood using an iterative approach. For Gaussian distributions (Equation 3.15), the maximization step can be solved by using the iterative EM algorithm. More details about the derivation can be found in [62]. This method is an effective and computationally efficient way of extracting motion from a sequence of images.

$$p_n(\vec{c} \mid \vec{v})=\frac{1}{\sqrt{2\pi}\sigma_v}\exp\left(-\frac{d^2(\vec{c},\vec{v})}{2\sigma_v^2}\right) \tag{3.15}$$

In Equation 3.15, the Gaussian Distribution is shown where σ_v^2 is equal to the estimate for the combined variance between the component velocity measurement and the modelling error. The parameter d is the probability that $\vec{c}$ is observed knowing that the actual velocity is equal to $\vec{v}$.

3.3. Machine Learning-based Optical Flow Techniques

The trend of the last years has been to apply Deep Neural Networks to different problems in computer vision including optical flow. This section will discuss the four most popular deep-learning models for optical flow.

The first model to use CNNs to predict optical flow was FlowNet [64]. The authors of this model compared two different methods: a default architecture where two images were stacked and fed into the model and a modified architecture. The modified architecture consisted of two identical models processing each image separately and combining them at a deeper stage of the network (Figure 3.1). To assist the model in the matching process an extra module is added known as the ”correlation layer”. This layer performs a comparison between two feature maps. It works by convolving data with each other rather than using a kernel as in the other layers. To decrease the amount of parameters, the maximum displacement d is limited. The correlations $c(x_1, x_2)$ are only computed in a neighbourhood of $D = 2d + 1$. Another feature of the custom architecture is the upconvolutional layers which aim to improve the resolution of the prediction. The endpoint error (EPE) is used as training loss. This is a standard error measure for estimating optical flow. It is calculated by averaging the Euclidean distance between the ground truth and the predicted flow vector for all the pixels. FlowNet reported competitive results on various datasets, despite being trained only on synthetic data, indicating good generalization. However, it did not achieve the accuracy of traditional methods on real-world data, particularly for small displacements.

An improvement of FlowNet is proposed in [65] in the form of FlowNet 2.0. This model is designed to improve FlowNet. The first improvement made is the addition of dataset schedules, where datasets are trained in a specific sequence. It is found that starting with a simple dataset to learn the general features and then fine-tuning on a more realistic dataset yields 25% higher results than the original FlowNet. The next addition is stacking networks. A stacking architecture is employed with additional warping based on the iterative methods the other state-of-the-art models use. The improved architecture can be observed in Figure 3.2. The final addition is solving the issue of low performance on small motion. This is tackled by adding a custom dataset with small motions in a different network and adding a fusion step to extract the final flow. This updated version of FlowNet achieves state-of-the-art performance with higher inference speeds [65].

The third model is PWC-Net designed by NVIDIA [66]. This model is developed to improve the accuracy of optical flow and have real-time capabilities. The key components of this model are the use of feature pyramid extractors, a warping layer, a cost volume layer, an optical flow estimator and a context network. From the two input images, L-Level pyramids of features are generated with the input image at the bottom. The pyramid is built up using convolutional filters for downsampling. The warping layer uses bilinear interpolation to compute the gradients to the input CNN features and flow for backpropagation. Warping helps with recognizing large motions. This layer is followed by the cost volume layer which stores the matching costs for correlating pixels to the corresponding pixels in the next frame. The output of the warping and cost volume are fed into a multi-layer CNN which calculates the optical flow. The last

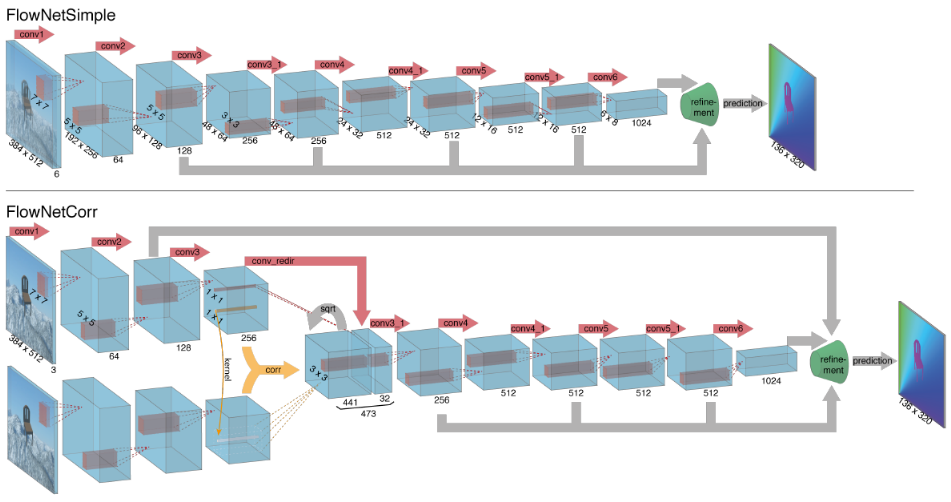


Figure 3.1: Two architectures discussed in [64], the top is the default architecture and the bottom is the custom-designed one.

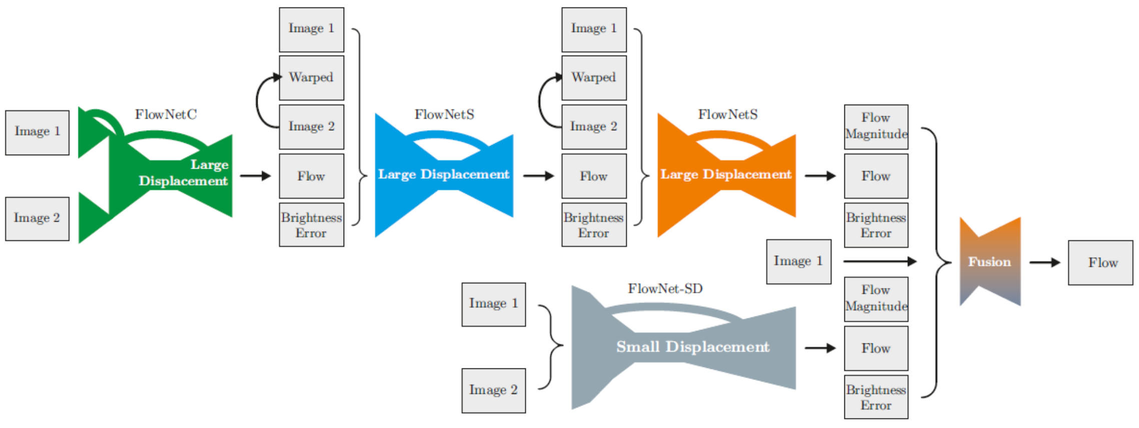


Figure 3.2: The FlowNet 2.0 architecture [65]

step is a context network which is a feed-forward CNN that enlarges the receptive field size. PWC-Net uses the MPI Sintel [67] and KITTI [68] datasets for benchmarking. It reaches high results but often fails to recover sharp motion boundaries and rapidly moving objects [66].

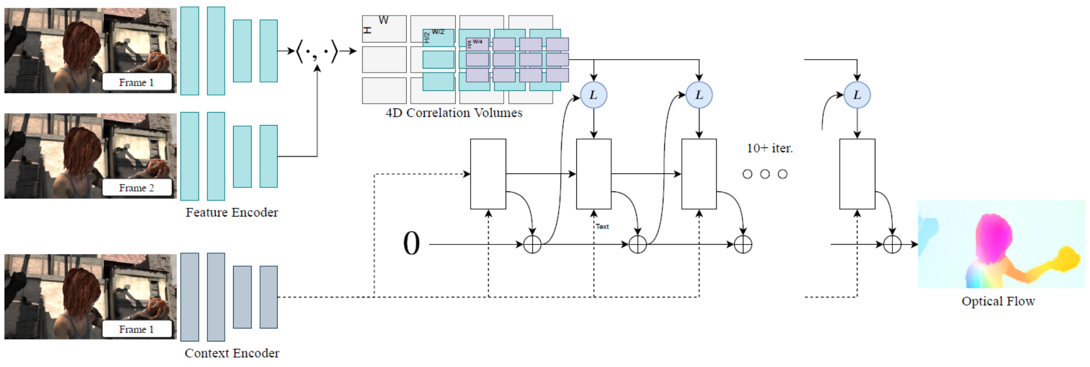


Figure 3.3: The RAFT architecture [69]

The last model discussed is Recurrent All-Pairs Field Transforms (RAFT) [69] for optical flow. The architecture can be seen as learning to optimize where many blocks are used to emulate an optimization algorithm. It consists of three main elements:

- Feature extraction: The images are applied to a feature encoder network with dense feature maps as output. Also, a context network is added to extract features from the first image.
- Computing visual similarity: A correlation volume is formed by taking the dot product between all the pairs of feature vectors. A 4-layer pyramid is constructed by pooling the last two dimensions of the correlation volume.
- Iteratively updating a flow field initialized at zero, using a recurrent GRU-based update operator. This operator retrieves values from the correlation volumes to update the flow field.

It is interesting to note that the model, which architecture can be found in Figure 3.3, operates on a single high-resolution flow. This prevents the errors of missing small fast-moving objects and many training iterations because of downsampling and upsampling of the images which is done in the previous models. Another advantage of this method is the high number of iterations possible. Experiments on Sintel and KITTI show that this model is superior compared to all the previous models in terms of accuracy on both these sets. [69]

This chapter discussed three of the most widely used motion extraction techniques used in literature. Firstly, the classical optical flow techniques such as Lucas-Kanade and Horn-Schunck were discussed. These methods have been used for many years and have been optimized for computational efficiency. Gaussian mixture models use a clustering method to extract motion and have good accuracy. Lastly were the machine learning-based techniques covered. These have proven to outperform all the other techniques but require quality data for training.

4
Object Tracking Methods

For the application of autonomous underwater cleanup, it is paramount that objects can be tracked across frames. The object-tracking landscape can be split up into two main parts: Single Object Tracking (SOT) and Multiple Object Tracking (MOT). It is chosen to focus on MOT as underwater environments are often cluttered and complex, with numerous objects moving around in the water column. In these situations, SOT can be limited in its ability to accurately detect and track trash. In contrast, MOT algorithms are better equipped to deal with multiple objects moving simultaneously and can track each object individually. Also, MOT algorithms can help to improve the accuracy and reliability of trash detection in underwater environments. By tracking multiple objects and comparing their movement patterns over time, MOT algorithms can identify unnatural or irregular object movement, which may indicate the presence of trash. In contrast, SOT algorithms are limited in their ability to detect such patterns and may miss important information about the movement of objects in the water.

Tracking algorithms can be online and offline. Online algorithms are defined as only using frames from the past. Offline tracking algorithms also use future frames to improve performance. These trackers are used for post-processing data but can not be applied in a real-time situation. To summarize, this chapter will focus on Multiple Object Tracking algorithms and it will start by explaining common algorithms used in tracking algorithms (Section 4.1), continued by an overview of the SOTA models from the MOT challenge in Section 4.2 and concluded by a summary of the tracking algorithms used in the underwater environment in Section 4.3.

4.1. Support algorithms

This section discusses the Kalman Filter and the Hungarian Algorithm. These algorithms are often used in object tracking.

4.1.1. Kalman Filter

The Kalman Filter is in its essence a mathematical algorithm which combines the past system states with the measurements to predict the current and future states [70]. It is widely used in various applications such as guidance, navigation and control of spacecraft, airplanes etc. The first step of the Kalman Filter is the initialization. After the initialization, the filter runs in a loop where it continuously predicts the future state and updates the measurement. During the initialization, the variables $\widehat{\boldsymbol{x}}_{0,0}, \boldsymbol{P}_{0,0}$ are set which are equal to the estimate of the state vector at time zero after measurement zero and the estimate uncertainty matrix at time zero after measurement zero respectively. The next step is the prediction which uses Equation 4.1 and Equation 4.2.

$$\hat{\boldsymbol{x}}_{n+1,n} = \boldsymbol{F_m}\hat{\boldsymbol{x}}_{n,n} + \boldsymbol{G}\boldsymbol{u}_n \tag{4.1}$$

$$\boldsymbol{P}_{n+1,n} = \boldsymbol{F_m}\boldsymbol{P}_{n,n}\boldsymbol{F_m}^T + \boldsymbol{Q} \tag{4.2}$$

The variable $\hat{\boldsymbol{x}}_{n+1,n}$ represents the estimated state vector in the future, $\boldsymbol{F}_{\mathrm{m}}$ is the state transition matrix, $\boldsymbol{G}$ is the control matrix, $\boldsymbol{u}_n$ is the input variable and $\boldsymbol{Q}$ is the process noise uncertainty which is equal to

$E\left(\boldsymbol{w}_n\boldsymbol{w}_n^T\right)$ where w_n is the process noise vector. This completes the prediction step where in the case of object tracking, the future position of the object is estimated. The third step is the measurement update where the Kalman Filter is used to improve the accuracy. This is required as no sensor is perfect and is always subjected to noise. Also in the measurement update, both the state vector and the uncertainty are calculated. The equations can be found in Equation 4.3 and Equation 4.4 respectively.

$$\hat{\boldsymbol{x}}_{n,n} = \hat{\boldsymbol{x}}_{n,n-1} + \boldsymbol{K}_n\left(\boldsymbol{z}_n - \boldsymbol{H_m}\hat{\boldsymbol{x}}_{n,n-1}\right) \tag{4.3}$$

$$\boldsymbol{P}_{n,n} = \left(\boldsymbol{I_i} - \boldsymbol{K}_n\boldsymbol{H_m}\right)\boldsymbol{P}_{n,n-1}\left(\boldsymbol{I_i} - \boldsymbol{K}_n\boldsymbol{H_m}\right)^T + \boldsymbol{K}_n\boldsymbol{R}_n\boldsymbol{K}_n^T \tag{4.4}$$

$$\boldsymbol{K}_n = \boldsymbol{P}_{n,n-1}\boldsymbol{H}_{\boldsymbol{m}}^T\left(\boldsymbol{H_m}\boldsymbol{P}_{n,n-1}\boldsymbol{H}_{\boldsymbol{m}}^T + \boldsymbol{R}_n\right)^{-1} \tag{4.5}$$

In the above equations is $\boldsymbol{z}_n$ equal to $\boldsymbol{H_m}\boldsymbol{x}_n$. $\boldsymbol{H_m}$ the observation matrix, $\boldsymbol{R}$ is the measurement uncertainty and equal to $E\left(\boldsymbol{v}_n\boldsymbol{v}_n^T\right)$ with v_n equal to the measurement noise error. $\boldsymbol{K}_n$ is a special variable and equal to Kalman Gain. This gain is calculated using Equation 4.5 and is a number between 0 and 1. As it can be seen in Equation 4.3, this gain determines how much weight is given to the measurements and the predictions. The higher the gain, the more weight is given to the measurements and the filter will respond more quickly to changes. However, a lower gain puts more weight on the predictions and the filter will be more robust to noise. The gain is calculated using the covariance matrices which quantifies the uncertainty of each component (e.g. an inaccurate sensor should have a lower gain). The full derivation of the Kalman Filter can be found in [71] [72].

4.1.2. Hungarian Algorithm

The Hungarian Algorithm is a matching algorithm using the Bipartite Graph [73]. The Bipartite Graph can be seen as a matrix with workers as rows and jobs as columns. Each worker can be matched with each job and this combination has a cost assigned. The goal of the Hungarian Algorithm is to match the workers with jobs in the most cost-effective way possible. The Bipartite Graph is a $N \times N$ matrix with N workers and N jobs. The Hungarian Algorithm uses 5-steps to find the most optimal solution:

1. In every row the minimum value is identified and subtracted from the rest of the row.
2. In every column the minimum value identified is subtracted from the rest of the column.
3. The previous steps introduced zero values in the matrix. This step crosses all these zeros while using a minimal amount of lines. E.g. the row: $[5, 0, 0, 0, 6, 2]$ contains three zeros next to each other, these can be crossed using 1 line. However the row $[5, 0, 0, 8, 0, 2]$ needs a minimum of 2 lines to cross all the zeros. If the amount of lines is equal to N, continue with step 5. If the amount of lines is smaller than N, continue with step 4.
4. The smallest element in the matrix which has not been crossed is identified. Subtract this element from the other non-crossed elements, but add it to the elements where two lines cross. Continue by repeating step 3.
5. Start with the column which has only 1 zero. This one is matched with a worker and the column and row can be crossed out. Continue with the next row which will have 1 zero. Continue until all the workers are assigned to a job.

By iteratively applying the above steps, the most cost-efficient solution is found. In the case of matching objects over frames, there are multiple different costs which can be used such as the Euclidean distance, Intersection over Union (IoU), Convolutional Cost, etc. Also, a combination of different parameters can be used to create a custom cost function. At first sight, this algorithm seems limited to cases where an $N \times N$ matrix can be formed and the problem is a minimization. However, these problems can be solved by adding dummy rows, and columns and switching the maximum and minimum values around. It is shown that the Hungarian Algorithm is a very powerful tool and is widely used in various applications. The complexity scales $O(n^3)$ which means the computational time drastically increases with the size of the matrix. Fortunately, in object tracking, the size of the matrix stays reasonably small which makes the computational complexity less of a concern [74] [75].

4.2. MOT Challenge Tracking Models

The MOTChallenge is the most important challenge related to object tracking and provides a good benchmark to compare the different SOTA models currently available. Unfortunately, many different datasets and versions are available making a fair comparison arduous. In total 22 different datasets were published from 2015 to 2023. Out of these 22 different challenges only the top-performing models on relevant datasets are included in this discussion. The MOT17 [76], CVPR 2020 [77], TAO Challenge [78], 3D-ZeF [79] and the STEP-ICCV21 [80] are deemed to be the most relevant to the problem of underwater object tracking as they contain a relatively low number of objects in a dynamic environment. None of the aforementioned datasets contains underwater objects besides the 3D-ZeF which contains Zebrafish in a laboratory environment. Caution should be kept when comparing the results of SOTA trackers on open-air datasets and assuming similar results will be achieved in the underwater domain. In [81], the authors proposed a new labelled underwater dataset to facilitate further research into underwater tracking. The paper also sets a benchmark using the current SOTA models on the underwater dataset. Interesting to note is that the performance these trackers have on open-air datasets does not come close to the performance on the underwater dataset. These findings reaffirm the notion that the underwater domain poses significant hurdles for the field of computer vision. It is still essential to consider and understand these SOTA models.

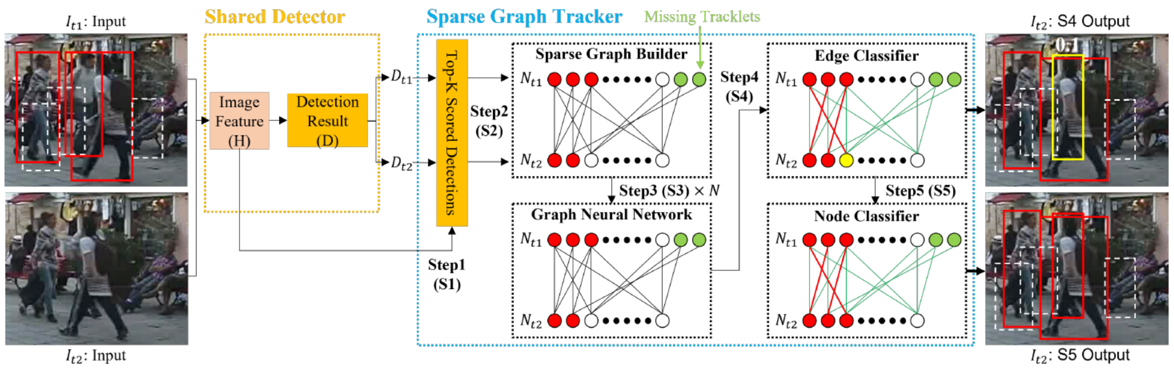


Figure 4.1: The Sparse Graph Tracker architecture [82]

In a tracking framework, a detected object is often assigned a state. This state usually has the options: active, tracked, lost or inactive. This format is followed in [83] where the tracker is formulated as the Markov Decision Process and uses SOT for each object. Every single tracker consists of four modules: tracking module, detector module, integrator module and learning module. Optical flow is used as a tracking module. The detector consists of a sliding window to detect the objects in the frame and the detection is combined with the tracking output using the integrator. The learning module uses the P-N learning principle [84] where it updates the positive and negative samples. This tracker reached a MOTA (Multiple Object Tracking Accuracy) of 44.4 on the MOT 2017 dataset and a MOTA of 58 on the KITTI dataset. These are good results but currently, better results are available. A different approach is taken in [85] where the regression step in object detection is exploited. The tracker named Tracktor is built on the assumption that an object only moves a few pixels over frames. Tracktor uses Faster-RCNN with a ResNet-101 backbone and FPNs (Feature Pyramid Networks). The bounding boxes detected in frame $t-1$ are regressed to frame t. The Region of Interest (RoI) pooling is then performed in the new ”estimated” area in frame t to find the tracked object. This is a very simple yet effective approach. A MOTA score of 53.5 is achieved on the MOT17 dataset which is considerably higher than the tracker in [83]. This score is increased to 76.4 by [82]. The tracker uses a combination of a detector and a sparse graph tracker. The sparse graph tracker has four different steps: Spare Graph Builder, Graph Neural Network, Edge Classifier and Node Classifier and reached high accuracies on many of the MOT datasets. An overview of the architecture can be found in Figure 4.1.

In [86], detection and tracking are performed jointly. One branch comprises a ConvNet network based on the R-FCN network with a ResNet-101 backbone. Candidate regions of interest (RoIs) are proposed by the region proposal network (RPN), the RoI pooling layer is applied, and the softmax function generates the output. The second branch applies the same process to the subsequent frame. The last layers of both branches are merged into a RoI tracking layer, where inter-frame bounding box regression is performed. This layer is included in the loss function. The authors note the limitation of processing high-quality frames, which reduces the framerate. This approach is also end-to-end trainable, achieving state-of-the-art results on the ImageNet VID dataset [87].

The most relevant dataset for this project is the 3D-ZeF dataset [88] which can be found in Figure 4.2. In [88], a method is proposed for 3D tracking Zebrafish. Unfortunately, this method does not apply to the topic of this literature study as fixed anchor boxes are used and the model is focused on 3D tracking and not 2D tracking.

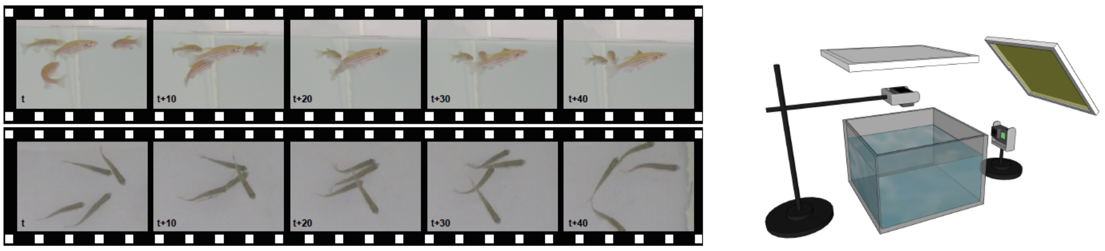


Figure 4.2: The Sparse Graph Tracker architecture [88]

4.3. Underwater Tracking Models

Numerous research articles have been identified relating to the application of tracking algorithms in the underwater domain, but the majority of these studies centre on the tracking of fish. The most used tracking technique is tracking-by-detection. In this technique, object detection is performed at every frame and tracking is conducted subsequently over the frames. The final tracking result is heavily influenced by the performance of the object detection algorithm. A famous online tracking algorithm is the Simple Online Realtime Tracking (SORT) algorithm [89] that uses the Faster Region CNN [15] as an object detector. Once the objects are detected, a linear constant velocity model is used to predict future displacements. A Kalman filter is used to refine the predictions and a Hungarian algorithm is responsible for matching the predictions with the detections. The Intersection over Union (IoU) metric is used to construct the assignment cost matrix. The algorithm assigns a large uncertainty value to newly detected objects to prevent tracking false positives. When objects are lost after 1 frame, they are deleted. This is one of the main limitations of the SORT algorithm. Exactly this problem is solved by the authors of the Deep SORT algorithm which is built on the original architecture [90]. The main difference is the change in the cost matrix. Instead of the IoU, the Mahalanobis distance is used and accompanied by a cosine distance appearance descriptor. This appearance descriptor is in practice a pre-trained CNN. The appearance of 100 objects is kept and a cost function based on the cosine distance (which indicates how similar feature vectors are to each other) is used. The Mahalonobis distance is powerful in the short term where the object's location can be estimated and the appearance descriptor has the advantage of recognizing objects in the case of rapid camera movement and occlusion. The Deep SORT algorithm is applied to underwater object tracking in [91]. This paper combines the YOLOv3 model for object detection with the Deep Sort algorithm and an LSTM network. Higher results were achieved on the Fish4Knwoledge [92] and NOAA dataset compared to the standalone Deep Sort algorithm. It shows that the combination of a better object detector and LSTM yields better tracking results.

Siamese networks are another popular technique used in tracking. In [94] a Siamese network is used as a component of the tracking algorithm. Initially, a GMM is used for background modelling followed by morphological operations to conclude the step of filtering the background out. Subsequently, the Siamese network is applied to the processed images to extract a feature vector as output. A miniature neural network is used to quantify the difference between the two input images. A number between 0 and 1 is constructed with 0 indicating dissimilarity and 1 full similarity. The output of the Siamese network and the Miniature Network are combined in the cost matrix. A Kalman filter is used to increase the object location and a Hungarian Algorithm to match the detections across frames. A similar approach using siamese networks is described in [95], where it is used for feature extraction. These features are then fed into an RPN to calculate the similarity score. This approach was also implemented on a small AUV and good

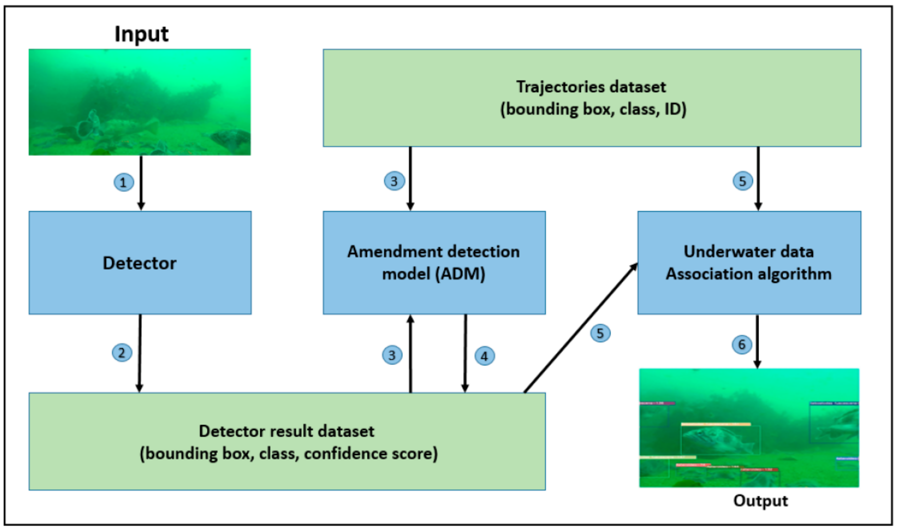


Figure 4.3: The FSTA architecture [93]

results were achieved.

Occlusion is always a big problem in tracking especially in the underwater domain. The authors of [93] present their Fish-school tracking algorithm (FSTA) which uses prior knowledge to reidentify objects. The architecture can be found in Figure 4.3. It is built up of 4 main components: the detector, the amendment detection model and the underwater data association algorithm. The detector is a simple object detection algorithm such as YoloX. The amendment model is responsible for making the detection results more reliable. Detections over time can drastically decrease in quality due to the nature of the environment. Also, objects can occlude each other, resulting in a temporary loss of specific objects. The amendment model uses tracklets as prior knowledge, feeding this into a Kalman Filter to predict future locations. The confidence scores are then modified with the IoU distance and the detection set of the current frame is outputted.

The association algorithm consists of the ByteTrack [96] algorithm combined with a re-ID feature extractor network. ByteTrack is a tracking algorithm built on the principle that every bounding box contains valuable information not only the ones with a high confidence score. This is a SOTA tracking which uses YoloX as a detector. This method has a MOTA of 79.1 which is comparable to SOTA MOTA discussed in the previous section.

All the previously discussed methods require a large amount of data to train. In [97] unsupervised learning is used to identify fish. Optical flow (RAFT) is used for background subtraction to generate pseudo-labels. In the second stage, the generated pseudo-labels are refined by a self-supervising model. The last stage consists of a segmentation model to predict the final label. By using pseudo-labels, the last step comes very close to a normal supervised model. The tracking is performed by the SORT [89] algorithm. This method shows good results and proves the feasibility of unsupervised learning.

4.4. Conclusion

This chapter described various tracking techniques with a focus on SOTA models and methods applied to the underwater domain. Unfortunately, none of the SOTA trackers were applied to the underwater so a fair comparison is not possible. The FSTA tracking algorithm has shown high results and potential. It has been applied to the underwater domain and uses a modular approach. This allows for easy reuse of specific blocks in the pipeline.

5 Training Data Sources and Tools

Data is the most important part of machine learning [98]. Without a high-quality dataset, it is impossible to get high-quality results. The underwater setting makes generating data a very expensive and complex job. Even if data is generated, annotating all this data is a very demanding job. Fortunately, some datasets are made available online for free usage. These datasets are described in Section 5.1. Besides these open source datasets, SEACLEAR itself also has a dataset which is described in Section 5.2. An alternative approach involves using a simulation environment to generate artificial data, as discussed in Section 5.3.

5.1. Open source real-world datasets

Since the focus is on sequential data, all non-sequential datasets such as TACO [99], TrashNet [100] or other open source datasets [101] can not be used.

Fortunately, there are also sequential datasets available. A significant database of underwater footage captured by remotely operated vehicles (ROVs) is the Deep-sea Debris Database [16]. This database contains only snippets of footage without labels. Multiple efforts have been made to annotate parts of this dataset. The most widely known is the Trashcan dataset [17], which includes 7,212 images and is an updated version of the Trash-ICRA19 dataset [19], with 22 different classes labelled with high accuracy. Another effort is described in [28], where more than 1,200 images were labelled across 19 different categories. Although many sequential images and labels are available online, they can not be used for this project as the goal is to extract an object trajectory. The trajectories get compromised by the movement of the ROV. Without knowing the exact position and orientation of the ROV, it is impossible to determine the object's trajectory. Unfortunately, none of the existing datasets provide the precise ROV orientation and position. Therefore, no suitable datasets are available for this application.

5.2. SEACLEAR dataset

The SEACLEAR team has performed various tests with their ROV named Tortuga in four different locations: Marseille in France, Dubrovnik Bay of Mali Ston, and Dubrovnik Lokrum which are both situated in Croatia and in the harbour in Hamburg Germany. In total 3441 images were captured in Marseille and can be observed in Figure 5.1. In Dubrovnik were 5169 images captured which can be seen in Figure 5.1.

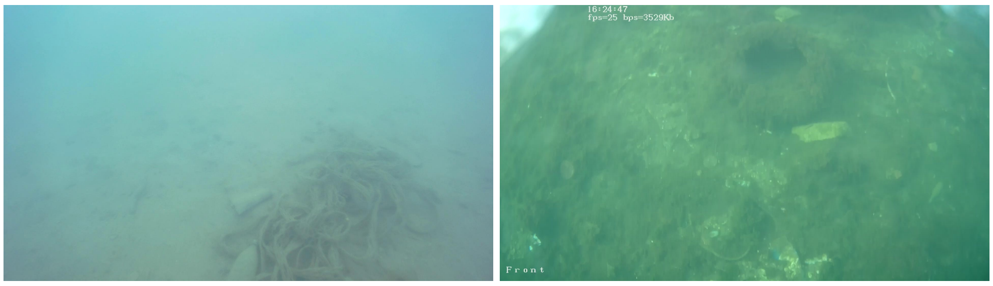


Figure 5.1: Examples of SEACLEAR data, left is an example from Croatia and right is an example from France.

Unfortunately, the quality of images from the tests in Hamburg was very low and cannot be used. The visibility was lower than 15cm which makes detecting plastic impossible. The visibility has a large effect on the feasibility of underwater trash cleanup. If the visibility is only 50cm, it is impossible to locate the trash. The images from both France and Croatia are of sufficient level for further use and the position and orientation of the ROV are known. Unfortunately, the camera also moves independently of the ROV and no data is available on this movement which renders this dataset also unusable for this study.

5.3. Simulators

As mentioned in the previous sections, no data is available to use for this project. Gathering real-world data is a very expensive and time-intensive operation which requires much manual data cleaning and labelling. This is beyond the scope of this research.

Another option is to use simulated data. Simulated data can be generated in controlled conditions with automatic labelling and with the exact position and orientation of the ROV. Different vision-based underwater simulators are available and to compare them in a structured manner are the following points considered:

1. Documentation: For efficient development, a simulator with extensive documentation is required. The scope of this thesis is not designing a simulator but using it and no additional time should be lost setting up the environment.
2. Flexibility: the aim is to add different objects to the scene so the simulator should allow for customization. Also, automatic data gathering is required, so the simulator should have an API to interact with.
3. Visual realism: this project involves computer vision and it is therefore of high importance that the simulator includes all the visual aspects which make underwater vision challenging.
4. Dynamics: as the thesis aims to use the trajectory of an object to improve the detection, the trajectory should be as realistic as possible. The real world is a turbulent environment and this should be reflected in the simulator. Important dynamics that should be present are the buoyancy, drag force, gravity, hydrostatic pressure and flow conditions.

The available visual simulators can be split up into three different families: ROS-based simulators, Gazebo-based simulators and Unity-based simulators. Each of these will be discussed more in detail in the remainder of this section.

5.3.1. ROS based simulator

UWSim [102] is an underwater simulator which uses ROS for the interface with osgOcean for rendering the scene. osgOcean shows realistic renders with an example detailed in Figure 5.2).

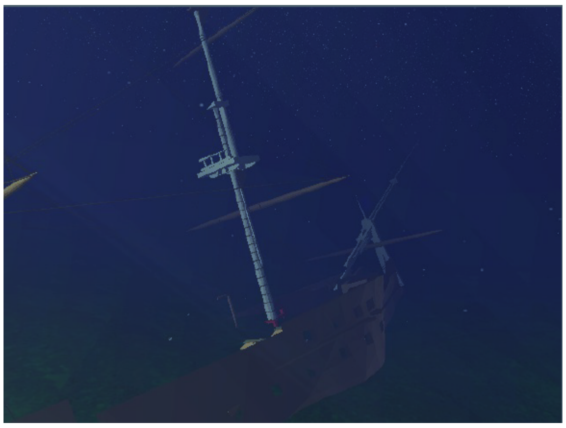


Figure 5.2: Example of visual in UWSim simulator with an object modelled in Blender [102]

An overview of all the components and how they are linked can be found in Figure 5.3. A modular and dynamic structure is employed with an XML file to set the scene. Dynamic simulation of rigid body motion is present and a state-space model is utilized based on two relationships:

- Non-linear 6 DOF rigid body motion equations
- Jacobian matrix which maps body velocities in an inertial frame.

Additional dynamics can also be incorporated using a Matlab model developed by researchers at the IRSLab (Jaume-I University, Castellón). However, the associated wiki was last updated in 2017, raising concerns about the maintenance of the simulation. Despite this, the simulator includes many key dynamics, offering significant flexibility and a high level of realism. The user community for this simulator is limited.

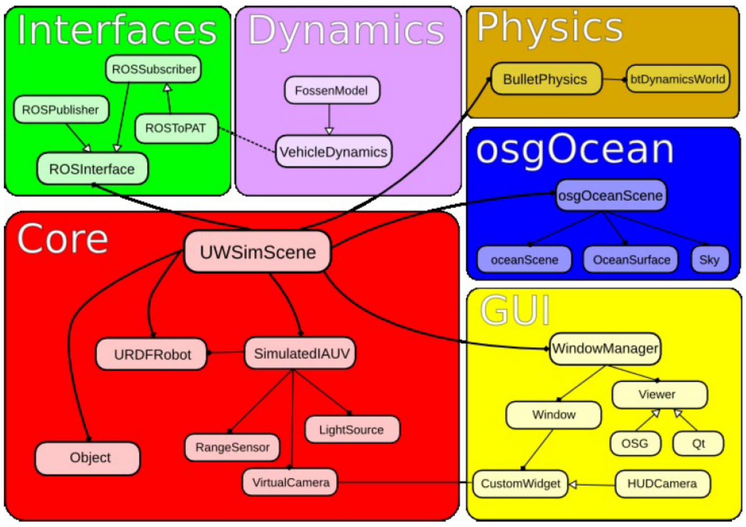


Figure 5.3: Architecture UWSim [102]

5.3.2. Gazebo based simulator

There are three Gazebo based simulators available: freefloating-gazebo [103], Rock-gazebo [104] and UUV simulator [105]. Besides these three, some studies [106] have created extra plugins for the underwater environment in Gazebo. The advantage of using Gazebo is the presence of a large community, extensive documentation and active development. The dynamics simulation in Gazebo operates by accepting inputs of forces and torques acting on a body, allowing for the addition of multiple forces if needed.

Freefloating-gazebo is a combination of UWSim and Gazebo and mainly consists of two plugins:

- World plugin for hydrodynamics: Simulating overall buoyancy, water surface and water current.
- Model plugin for thruster control: Model for low-level control through thruster effort.

Besides the plugins, a PID controller is also developed. As it is integrated with UWSim, the visual integration is of a high level and an example can be observed in Figure 5.4.

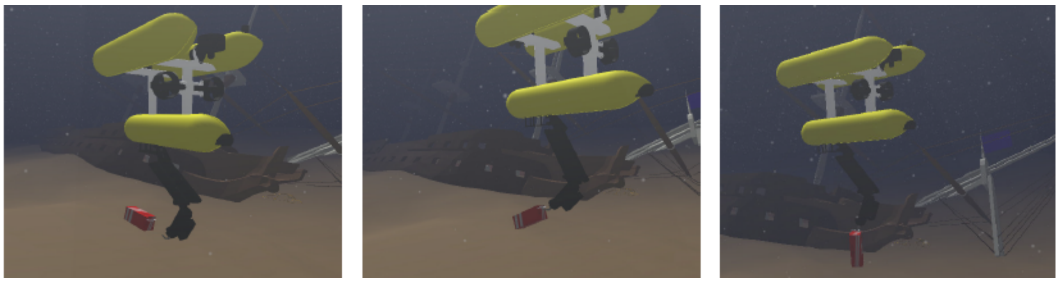


Figure 5.4: Gazebo simulator visual example [103]

The Rock-gazebo plug-in is similar to the free-floating-gazebo in the sense that it also uses XML files to develop a scene and makes use of osgOcean classes for the rendering. The main purpose of these plugins is to work in the Rock framework in Gazebo. The final plugin is the UUV simulator. This simulator has to goal of meeting the requirements set by the authors in [105]. Neither one of the previously mentioned simulators meets all their requirements such as multiple robot simulation and low complexity to set up real-world scenarios. Therefore UUV developed new packages for Gazebo with the main contributions of:

- Actuator models
- Underwater sensors
- Hydrodynamic and hydrostatic forces and moments
- Underwater worlds and environmental loads.

Unfortunately, the Github page of this simulator has been archived indicating that no ongoing developments..

5.3.3. Unity based simulator

Unity is one of the most famous cross-platform gaming engines. It comes with a physics engine and extensive real-time rendering. The authors of [107] show how Unity3D was used to create an underwater environment and automatically capture images to build a dataset which is used for computer vision. A Yolov5 model was trained on these images to create an underwater gate detection model. The graphics of Unity3D are shown in Figure 5.5. The combination of a large community, the realistic renders and proof that an object detector can be trained on this data makes it a very attractive choice.

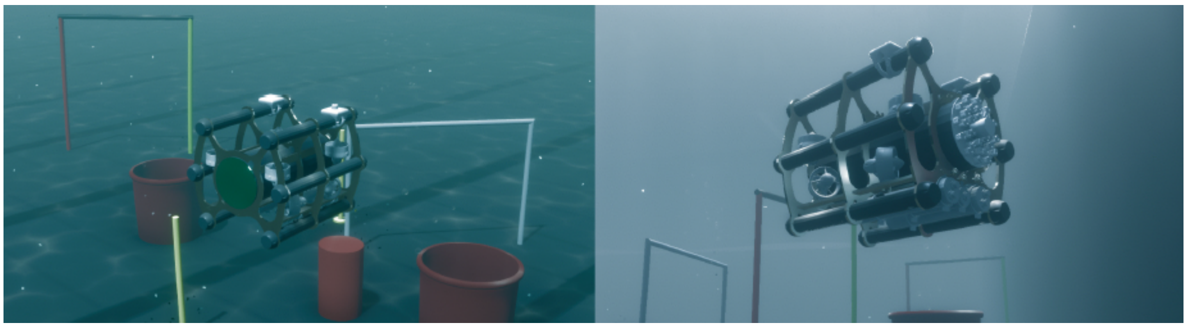


Figure 5.5: Unity3D simulator visual example [107]

5.4. Conclusion

A comprehensive overview of various open-source datasets is provided. Although underwater datasets are available, none of them meet the requirements of being sufficiently large, containing various objects, being fully annotated, and having the exact orientation and position of the camera for each frame. Generating a custom dataset using real-world cameras is deemed infeasible due to time and financial constraints. Therefore, the only viable option is to use simulated data. To model the environment as accurately as possible, a simulator with high visual realism is preferred. Among the available options, the Unity-based simulator is favoured due to its extensive documentation, high-fidelity visuals, and previous work demonstrating its feasibility.

6 Research Proposal

This chapter will discuss the research proposal for the thesis based on the outcome of the previous chapters.

6.1. Introduction

The introduction has shown that image object detection has reached good performances on the Trash-can dataset. Instead of improving the processing of the current information available (i.e., pixel values from the input images) by designing a novel model architecture, it is chosen to increase the amount of information available and use existing models to reach better results. Increasing the available information is accomplished by embedding temporal domain knowledge to enhance detections. It is shown that limited domain knowledge is embedded in the single image frames; however, videos provide a much richer source of information by incorporating the temporal domain. Information about trajectories and dynamics can be used to assist the model in making better detections. As concluded in Chapter 5 a simulation will be used to generate a dataset for model training. Instead of images, this dataset will consist of videos filmed by a simulated ROV. Consequently, the movement of marine life and vegetation will need to be accurately modelled.

6.2. Research Questions

The previous section has formulated the identified gap in research which will be formalized into research questions in this section. The main research question is the following:

”Does including specific temporal domain knowledge into deep learning techniques improve object detection accuracy applied to underwater applications?”

As Chapter 5 has shown, simulated data will have to be used. This translates into a theoretically unlimited dataset. These results will be compared to SOTA off-the-shelf trackers. This leads to the subquestions of the research question:

- What is the feasibility of setting up an underwater simulator to capture temporal data of litter, marine life and vegetation?
- What is the optimal way to couple a deep-learning-based architecture with temporal domain knowledge?
- What are the limitations of the study, such as the generalisability of the model on real-world data?

The next steps will consist of generating the required data using a simulator, developing an architecture which embeds temporal domain knowledge about the underwater environment and evaluating the performance. This study aims to identify the effect of domain knowledge on object performance and has the potential to impact environmental conservation efforts and the broader field of underwater imaging and analysis.

7 Conclusion

This literature review covered a variety of topics related to embedding domain knowledge in an underwater object detection and tracking algorithm. This work is part of the SEACLEAR project which aims to clean up the water column and ocean floor from litter. Oceanic litter has been a problem for decades but in the last years, it has become one of the largest environmental problems humanity has to face. It has a devastating effect on the marine environment and impacts all life on Earth. As litter degrades, it is consumed by the lowest organisms in the food chain, resulting in many animals, including humans, having microplastics in their bodies. Addressing this pressing problem is crucial.

Due to the immense scale of the marine pollution, autonomous cleanup is essential. To enable autonomous cleanup, autonomous tracking and detection are required. A comprehensive study is performed on current SOTA detection methods. It is concluded that computer vision research has made incredible advancements in the last decade and great things can be achieved nowadays. Current SOTA models are highly effective at extracting nearly all information encapsulated in an image. Therefore, the focus is on embedding domain knowledge in deep learning SOTA architectures to enhance their performance and compensate for limitations due to a lack of annotated data.

An in-depth analysis was performed on the availability of data. It was concluded that the required data was not available, leading to the pursuit of creating data artificially.

The goal of this work is to study the existing deep-learning architectures related to object detection in underwater applications. A comprehensive and in-depth overview of these architectures is provided. It was noted that none of them considers temporal domain knowledge, and it is proposed that integrating deep learning methods with temporal domain knowledge should be further explored [108].

References

[1] D. Robinson. “14 biggest environmental problems of 2023.” (Jan. 2023), [Online]. Available: https://earth.org/the-biggest-environmental-problems-of-our-lifetime/.

[2] M. Fava. “Ocean plastic pollution an overview: Data and statistics.” (May 2022), [Online]. Available: https://oceanliteracy.unesco.org/plastic-pollution-ocean/#:~:text=Currently%5C%2C%5C%20there%5C%20are%5C%20about%5C%2050,ends%5C%20u%5C%20forming%5C%20garbage%5C%20patches..

[3] L. J. J. Meijer, T. van Emmerik, R. van der Ent, C. Schmidt, and L. Lebreton, “More than 1000 rivers account for 80% of global riverine plastic emissions into the ocean,” *Science Advances*, vol. 7, no. 18, eaaz5803, 2021. DOI: 10.1126/sciadv.aaz5803. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.aaz5803. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.aaz5803.

[4] J. Barrett, Z. Chase, J. Zhang, *et al.*, “Microplastic pollution in deep-sea sediments from the great australian bight,” *Frontiers*, Oct. 2020.

[5] K. Amadeo. “How air, water, and plastic pollution affect the economy.” (Jun. 2022), [Online]. Available: https://www.thebalancemoney.com/pollution-facts-economic-effect-4161042#:~:text=Plastic%5C%20pollution%5C%20costs%5C%20%5C%2413%5C%20billion,flexible%5C%2C%5C%20lightweight%5C%2C%5C%20and%5C%20sustainable..

[6] A. Morlet, D. Waughray, and M. R. Stuchtey, “The new plastics economy rethinking the future of plastics,” Jan. 2016.

[7] W. Fang, L. Ding, P. E. Love, *et al.*, “Computer vision applications in construction safety assurance,” *Automation in Construction*, vol. 110, p. 103 013, 2020. DOI: https://doi.org/10.1016/j.autcon.2019.103013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580519301487.

[8] B. G. Weinstein, “A computer vision for animal ecology,” *The Journal of animal ecology*, vol. 87, no. 3, pp. 533–545, May 2018. DOI: 10.1111/1365-2656.12780. [Online]. Available: https://doi.org/10.1111/1365-2656.12780.

[9] T. Liu, A. W. Burner, T. W. Jones, and D. A. Barrows, “Photogrammetric techniques for aerospace applications,” *Progress in Aerospace Sciences*, vol. 54, pp. 1–58, 2012. DOI: https://doi.org/10.1016/j.paerosci.2012.03.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376042112000267.

[10] D. L. Rizzini, F. Kallasi, F. Oleari, and S. Caselli, “Investigation of vision-based underwater object detection with multiple datasets,” *International Journal of Advanced Robotic Systems*, Mar. 2015.

[11] A. Jesus, C. Zito, C. Tortorici, E. Roura, and G. D. Masi, “Underwater object classification and detection: First results and open challenges,” in *OCEANS 2022 - Chennai*, IEEE, Feb. 2022. DOI: 10.1109/oceanschennai45887.2022.9775417. [Online]. Available: https://doi.org/10.1109%5C%2Foceanschennai45887.2022.9775417.

[12] R. A. Dakhil and A. R. H. Khayeat, “Review on deep learning techniques for underwater object detection,” in *Data Science and Machine Learning*, Academy and Industry Research Collaboration Center (AIRCC), Sep. 2022. DOI: 10.5121/csit.2022.121505. [Online]. Available: https://doi.org/10.5121%5C%2Fcsit.2022.121505.

[13] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “Yolox: Exceeding yolo series in 2021,” 2021. arXiv: 2107.08430 [cs.CV].

[14] S. Rath. “Yolov6 custom dataset training – underwater trash detection.” (Nov. 2022), [Online]. Available: https://learnopencv.com/yolov6-custom-dataset-training/.

[15] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016. arXiv: 1506.01497 [cs.CV].

[16] J. A. for Marine-Earth Science and T. (JAMSTEC), *Deep-sea debris database*, (Accessed on 02/03/2023). [Online]. Available: https://www.godac.jamstec.go.jp/dsdebris/e/index.html.

[17] J. Hong, M. Fulton, and J. Sattar, “Trashcan: A semantically-segmented dataset towards visual detection of marine debris,” *CoRR*, vol. abs/2007.08097, 2020. arXiv: 2007.08097. [Online]. Available: https://arxiv.org/abs/2007.08097.

[18] H. Deng, D. Ergu, F. Liu, B. Ma, and Y. Cai, “An embeddable algorithm for automatic garbage detection based on complex marine environment,” *Sensors*, vol. 21, no. 19, 2021. DOI: 10.3390/s21196391. [Online]. Available: https://www.mdpi.com/1424-8220/21/19/6391.

[19] M. Fulton, J. Hong, M. J. Islam, and J. Sattar, “Robotic detection of marine litter using deep visual detection models,” *CoRR*, vol. abs/1804.01079, 2018. arXiv: 1804.01079. [Online]. Available: http://arxiv.org/abs/1804.01079.

[20] C. Wu, Y. Sun, T. Wang, and Y. Liu, “Underwater trash detection algorithm based on improved yolov5s,” *Journal of Real-Time Image Processing*, vol. 19, Jul. 2022. DOI: 10.1007/s11554-022-01232-0.

[21] X. Teng, Y. Fei, K. He, and L. Lu, “The object detection of underwater garbage with an improved yolov5 algorithm,” in *Proceedings of the 2022 International Conference on Pattern Recognition and Intelligent Systems*, ser. PRIS ’22, Wuhan, China: Association for Computing Machinery, 2022, pp. 55–60. DOI: 10.1145/3549179.3549189. [Online]. Available: https://doi.org/10.1145/3549179.3549189.

[22] B. Xue, B. Huang, W. Wei, *et al.*, “An efficient deep-sea debris detection method using deep neural networks,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 12 348–12 360, 2021. DOI: 10.1109/JSTARS.2021.3130238.

[23] B. Xue, B. Huang, G. Chen, H. Li, and W. Wei, “Deep-sea debris identification using deep convolutional neural networks,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 8909–8921, 2021. DOI: 10.1109/JSTARS.2021.3107853.

[24] J. C. Hipolito, A. Sarraga Alon, R. V. Amorado, M. G. Z. Fernando, and P. I. C. De Chavez, “Detection of underwater marine plastic debris using an augmented low sample size dataset for machine vision system: A deep transfer learning approach,” in *2021 IEEE 19th Student Conference on Research and Development (SCOReD)*, 2021, pp. 82–86. DOI: 10.1109/SCOReD53546.2021.9652703.

[25] Y.-C. Wu, P.-Y. Shih, L.-P. Chen, C.-C. Wang, and H. Samani, “Towards underwater sustainability using rov equipped with deep learning system,” in *2020 International Automatic Control Conference (CACS)*, 2020, pp. 1–5. DOI: 10.1109/CACS50047.2020.9289788.

[26] L. Wei, S. Kong, Y. Wu, and J. Yu, “Image semantic segmentation of underwater garbage with modified u-net architecture model,” *Sensors*, vol. 22, no. 17, 2022. DOI: 10.3390/s22176546. [Online]. Available: https://www.mdpi.com/1424-8220/22/17/6546.

[27] J.-I. Watanabe, Y. Shao, and N. Miura, “Underwater and airborne monitoring of marine ecosystems and debris,” *Journal of Applied Remote Sensing*, vol. 13, p. 1, Oct. 2019. DOI: 10.1117/1.JRS.13.044509.

[28] A. Sánchez-Ferrer, A. J. Gallego, J. J. Valero-Mas, and J. Calvo-Zaragoza, “The cleansea set: A benchmark corpus for underwater debris detection and recognition,” in *Pattern Recognition and Image Analysis*, A. J. Pinho, P. Georgieva, L. F. Teixeira, and J. A. Sánchez, Eds., Cham: Springer International Publishing, 2022, pp. 616–628.

[29] A. Balakrishnan, B. S, and S. M H, “Classification of low quality underwater objects using convolutional neural networks and transfer learning,” Feb. 2022, pp. 1–4. DOI: 10.1109/OCEANSChennai45887.2022.9775387.

[30] R. Bajaj, S. Garg, N. Kulkarni, and R. Raut, “Sea debris detection using deep learning : Diving deep into the sea,” in *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, 2021, pp. 1–6. DOI: 10.1109/GUCON50781.2021.9573722.

[31] J. Jia, M. Fu, X. Liu, and B. Zheng, “Underwater object detection based on improved efficientdet,” *Remote Sensing*, vol. 14, no. 18, 2022. DOI: 10.3390/rs14184487. [Online]. Available: https://www.mdpi.com/2072-4292/14/18/4487.

[32] A. Jalal, A. Salman, A. Mian, M. Shortis, and F. Shafait, “Fish detection and species classification in underwater environments using deep learning with temporal information,” *Ecological Informatics*, vol. 57, p. 101 088, 2020. DOI: https://doi.org/10.1016/j.ecoinf.2020.101088. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574954120300388.

[33] J. Sattar and G. Dudek, “Where is your dive buddy: Tracking humans underwater using spatio-temporal features,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 3654–3659. DOI: 10.1109/IROS.2007.4399527.

[34] D. McIntosh, T. P. Marques, A. B. Albu, R. Rountree, and F. De Leo, “Tempnet: Temporal attention towards the detection of animal behaviour in videos,” 2022. DOI: 10.48550/ARXIV.2211.09950. [Online]. Available: https://arxiv.org/abs/2211.09950.

[35] H. Måløy, A. Aamodt, and E. Misimi, “A spatio-temporal recurrent network for salmon feeding action recognition from underwater videos in aquaculture,” *Computers and Electronics in Agriculture*, vol. 167, p. 105 087, 2019. DOI: https://doi.org/10.1016/j.compag.2019.105087. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919313262.

[36] P. Oteiza, I. Odstrcil, G. Lauder, R. Portugues, and F. Engert, “A novel mechanism for mechanosensory-based rheotaxis in larval zebrafish,” *Nature*, vol. 547, Jul. 2017. DOI: 10.1038/nature23014.

[37] J. C. Liao, “A review of fish swimming mechanics and behaviour in altered flows,” *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 362, pp. 1973–1993, 2007.

[38] S. Pokhrel, “Beginners guide to convolutional neural networks,” *Towards Data Science*, Sep. 2019, (Accessed on 03/29/2023).

[39] M. Mishra, “Convolutional neural networks, explained,” *Towards Data Science*, Aug. 2020, (Accessed on 03/29/2023).

[40] V. Jain, “Everything you need to know about “activation functions” in deep learning models,” *Towards Data Science*, Dec. 2019, (Accessed on 03/29/2023).

[41] R. G. Fei-Fei Li Jiajun Wu, *Cs231n convolutional neural networks for visual recognition*, https://cs231n.github.io/convolutional-networks/, 2022.

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015. arXiv: 1512.03385 [cs.CV].

[43] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, “Selective search for object recognition,” *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013. [Online]. Available: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013.

[44] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, “Microsoft coco: Common objects in context,” 2015. arXiv: 1405.0312 [cs.CV].

[45] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303–308, Sep. 2009, Printed version publication date: June 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-pascal-visual-object-classes-voc-challenge/.

[46] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[47] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, “Bounding box regression with uncertainty for accurate object detection,” 2019. arXiv: 1809.08545 [cs.CV].

[48] J. Hosang, R. Benenson, and B. Schiele, *Learning non-maximum suppression*, 2017. arXiv: 1705.02950 [cs.CV].

[49] Y. Ç. Aktaş, “Object detection with convolutional neural networks,” *Towards Data Science*, Jan. 2022, (Accessed on 03/29/2023).

[50] R. Girshick, “Fast r-cnn,” 2015. arXiv: 1504.08083 [cs.CV].

[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016. arXiv: 1506.02640 [cs.CV].

[52] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. arXiv: 2207.02696 [cs.CV].

[53] D. B. Camp, *Long short-term memory networks (lstm)- simply explained!* https://databasecamp.de/en/ml/lstms#:~:text=LSTM%20models%20are%20a%20subtype,term%20memory%20or%20discard%20it., May 2022.

[54] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[55] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014. arXiv: 1406.1078 [cs.CL].

[56] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. arXiv: 1412.6980 [cs.LG].

[57] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and W. E, “Towards theoretically understanding why sgd generalizes better than adam in deep learning,” 2021. arXiv: 2010.05627 [cs.LG].

[58] A. Rosebrock, *Intersection over union (iou) for object detection*, https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2022.

[59] B. Chiang and J. Bohg, *Lecture notes cs231a: Computer vision, from 3d reconstruction to recognition: Optical and scene flow*, Feb. 2022.

[60] B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[61] S. Na, L. Xumin, and G. Yong, “Research on k-means clustering algorithm: An improved k-means clustering algorithm,” pp. 63–67, 2010. DOI: 10.1109/IITSI.2010.74.

[62] A. Jepson and M. Black, “Mixture models for optical flow computation,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1993, pp. 760–761. DOI: 10.1109/CVPR.1993.341161.

[63] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://www.jstor.org/stable/2984875 (visited on 04/12/2023).

[64] A. Dosovitskiy, P. Fischer, E. Ilg, *et al.*, “Flownet: Learning optical flow with convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2758–2766. DOI: 10.1109/ICCV.2015.316.

[65] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” 2016. DOI: 10.48550/ARXIV.1612.01925. [Online]. Available: https://arxiv.org/abs/1612.01925.

[66] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, “Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume,” Jun. 2018, pp. 8934–8943. DOI: 10.1109/CVPR.2018.00931.

[67] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, “A naturalistic open source movie for optical flow evaluation,” Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed., pp. 611–625, Oct. 2012.

[68] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. DOI: 10.1177/0278364913491297. eprint: https://doi.org/10.1177/0278364913491297. [Online]. Available: https://doi.org/10.1177/0278364913491297.

[69] Z. Teed and J. Deng, “Raft: Recurrent all-pairs field transforms for optical flow,” 2020. arXiv: 2003.12039 [cs.CV].

[70] Q. Li, R. Li, K. Ji, and W. Dai, “Kalman filter and its application,” in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015, pp. 74–77. DOI: 10.1109/ICINIS.2015.35.

[71] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[72] A. Becker, *Kalman filter*, https://www.kalmanfilter.net/default.aspx, 2022.

[73] A. Asratian, T. Denley, and R. Häggkvist, *Bipartite Graphs and Their Applications* (Cambridge Tracts in Mathematics). Cambridge University Press, 1998. [Online]. Available: https://books.google.nl/books?id=cImr4BGQ85kC.

[74] J. Cohen, “Exactly how the hungarian algorithm works,” Feb. 2023.

[75] H. W. Kuhn, “The Hungarian Method for the Assignment Problem,” vol. 2, no. 1–2, pp. 83–97, Mar. 1955. DOI: 10.1002/nav.3800020109.

[76] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, “MOT16: A benchmark for multi-object tracking,” *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831. [Online]. Available: http://arxiv.org/abs/1603.00831.

[77] P. Voigtlaender, M. Krause, A. Osep, *et al.*, “Mots: Multi-object tracking and segmentation,” Jun. 2019.

[78] A. Dave, T. Khurana, P. Tokmakov, C. Schmid, and D. Ramanan, “Tao: A large-scale benchmark for tracking any object,” in *European Conference on Computer Vision*, 2020. [Online]. Available: https://arxiv.org/abs/2005.10356.

[79] M. Pedersen, J. B. Haurum, S. H. Bengtson, and T. B. Moeslund, “3d-zef: A 3d zebrafish tracking benchmark dataset,” *arXiv:2006.08466[cs]*, 2020, arXiv: 2006.08466. [Online]. Available: https://arxiv.org/abs/2006.08466.

[80] M. Weber, J. Xie, M. Collins, *et al.*, “Step: Segmenting and tracking every pixel,” 2021. arXiv: 2102.11859 [cs.CV].

[81] L. Kezebou, V. Oludare, K. Panetta, and S. S. Agaian, “Underwater object tracking benchmark and dataset,” pp. 1–6, 2019. DOI: 10.1109/HST47167.2019.9032954.

[82] J. Hyun, M. Kang, D. Wee, and D.-Y. Yeung, “Detection recovery in online multi-object tracking with sparse graph tracker,” 2022. arXiv: 2205.00968 [cs.CV].

[83] T. Yang, C. Cappelle, Y. Ruichek, and M. El Bagdouri, “Online multi-object tracking combining optical flow and compressive tracking in markov decision process,” *Journal of Visual Communication and Image Representation*, vol. 58, pp. 178–186, 2019. DOI: https://doi.org/10.1016/j.jvcir.2018.11.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S104732031830316X.

[84] Z. Kalal, J. Matas, and K. Mikolajczyk, “P-n learning: Bootstrapping binary classifiers by structural constraints,” pp. 49–56, 2010. DOI: 10.1109/CVPR.2010.5540231.

[85] P. Bergmann, T. Meinhardt, and L. Leal-Taixé, “Tracking without bells and whistles,” *CoRR*, vol. abs/1903.05625, 2019. arXiv: 1903.05625. [Online]. Available: http://arxiv.org/abs/1903.05625.

[86] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Detect to track and track to detect,” 2018. arXiv: 1710.03958 [cs.CV].

[87] O. Russakovsky, J. Deng, H. Su, *et al.*, “Imagenet large scale visual recognition challenge,” 2015. arXiv: 1409.0575 [cs.CV].

[88] M. Pedersen, J. B. Haurum, S. H. Bengtson, and T. B. Moeslund, “3d-zef: A 3d zebrafish tracking benchmark dataset,” 2020. arXiv: 2006.08466 [cs.CV].

[89] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, Sep. 2016. DOI: 10.1109/icip.2016.7533003. [Online]. Available: https://doi.org/10.1109%5C%2Ficip.2016.7533003.

[90] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” 2017. arXiv: 1703.07402 [cs.CV].

[91] A. Mathias, S. Dhanalakshmi, and R. Kumar, “Occlusion aware underwater object tracking using hybrid adaptive deep sort-yolov3 approach,” *Multimedia Tools and Applications*, pp. 1–13, 2022.

[92] R. Fisher, K.-T. Shao, and J. Chen-Burger, “Overview of the fish4knowledge project,” in Mar. 2016, pp. 1–17. DOI: 10.1007/978-3-319-30208-9_1.

[93] T. Liu, S. He, H. Liu, Y. Gu, and P. Li, “A robust underwater multiclass fish-school tracking algorithm,” *Remote Sensing*, vol. 14, no. 16, 2022. DOI: 10.3390/rs14164106. [Online]. Available: https://www.mdpi.com/2072-4292/14/16/4106.

[94] M. V. Rahul, R. Ambareesh, and G. Shobha, “Siamese network for underwater multiple object tracking,” ICMLC 2017, pp. 511–516, 2017. DOI: 10.1145/3055635.3056579. [Online]. Available: https://doi.org/10.1145/3055635.3056579.

[95] M.-F. R. Lee and Y.-C. Chen, “Artificial intelligence based object detection and tracking for a small underwater robot,” *Processes*, vol. 11, no. 2, 2023. DOI: 10.3390/pr11020312. [Online]. Available: https://www.mdpi.com/2227-9717/11/2/312.

[96] Y. Zhang, P. Sun, Y. Jiang, *et al.*, “Bytetrack: Multi-object tracking by associating every detection box,” 2022. arXiv: 2110.06864 [cs.CV].

[97] A. Saleh, M. Sheaves, D. Jerry, and M. R. Azghadi, “Unsupervised fish trajectory tracking and segmentation,” 2022. arXiv: 2208.10662 [cs.CV].

[98] Y. Tkachova, “How important data quality for machine learning,” *mastheadata*, May 2022, (Accessed on 28/02/2023).

[99] P. F. Proença and P. Simões, “TACO: trash annotations in context for litter detection,” *CoRR*, vol. abs/2003.06975, 2020. arXiv: 2003.06975. [Online]. Available: https://arxiv.org/abs/2003.06975.

[100] G. Thung and M. Yang, “Classification of trash for recyclability status,” 2016.

[101] P. Machado, *Underwater plastic dataset*, version 1.0, Jul. 2022. DOI: 10.5281/zenodo.6907230. [Online]. Available: https://doi.org/10.5281/zenodo.6907230.

[102] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, “An open source tool for simulation and supervision of underwater intervention missions,” in *International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, Oct. 2012.

[103] O. Kermorgant, “A dynamic simulator for underwater vehicle-manipulators,” vol. 8810, Oct. 2014. DOI: 10.1007/978-3-319-11900-7_3.

[104] T. Watanabe, G. Neves, R. Cerqueira, *et al.*, “The rock-gazebo integration and a real-time auv simulation,” in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, 2015, pp. 132–138. DOI: 10.1109/LARS-SBR.2015.15.

[105] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, “Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation,” in *OCEANS 2016 MTS/IEEE Monterey*, 2016, pp. 1–8. DOI: 10.1109/OCEANS.2016.7761080.

[106] J. Britto, A. Conceição, S. Joyeux, and J. Albiez, “Improvements in dynamics simulation for underwater vehicles deployed in gazebo,” in *OCEANS 2017 - Anchorage*, 2017, pp. 1–6.

[107] P. Szlęg, P. Barczyk, B. Maruszczak, S. Zieliñski, and E. Szymañska, “Simulation environment for underwater vehicles testing and training in unity3d,” in *Intelligent Autonomous Systems 17: Proceedings of the 17th International Conference IAS-17*, Springer, 2023, pp. 844–853.

[108] A. Daw, A. Karpatne, W. Watkins, J. Read, and V. Kumar, “Physics-guided neural networks (pgnn): An application in lake temperature modeling,” 2021. arXiv: `1710.11431 [cs.LG]`.

Part III

Conclusion

Conclusion

This chapter serves as the conclusion to the thesis and revisits the research questions posed in the literature study.

Revisiting Research Objectives

The research objective is structured as one main research question and three sub-questions. These are repeated here for clarity and are all discussed individually.

”Does including specific temporal domain knowledge into deep learning techniques improve object detection accuracy applied to underwater applications?”

This research objective led to the development of an underwater simulator to ensure the availability of high-quality data for testing the hypotheses. Subsequently, two mathematical models were developed as part of the domain knowledge module. It is chosen to decouple objection detection and object classification to reduce complexity. The impact of including temporal domain knowledge on the accuracy is assessed by training the trajectory classifier using a default training process and the knowledge-infused process. In the basic case of training, validating and testing on high-quality ideal data the domain knowledge has no benefit. However, a significant improvement was observed when noise was introduced into the training and validation data.

The first sub-question, which pertains to the setup of the simulator, is formulated as follows:

What is the feasibility of setting up an underwater simulator to capture temporal data of litter, marine life and vegetation?

Due to the scarcity of data meeting the requirements, a custom underwater simulator was developed to generate the necessary dataset. The authors of a Unity-based underwater simulator generously provided the source code and a license. Although comprehending an existing codebase requires time, the simulator was designed in a modular fashion, which significantly simplifies the process of making modifications. Additionally, the authors made time to provide information about the architecture and address any specific questions. Nonetheless, getting familiar with the Unity environment and adapting the existing infrastructure to the needs of this research was a highly time-consuming endeavour. Firstly, the simulator was adapted from a dynamic ROV to a static camera setup. Secondly, the scene was modified to reflect an oceanic environment. The existing physics was revised and updated to meet the simulator’s goals. Various objects were added to the scene, ensuring they adhered to physical laws. Finally, a custom scene generator and data-gathering script were developed. Given the modular setup, it is feasible to start with basic features and progressively add more to enhance visual fidelity and the realism of the physics.

The second subquestion addresses the most optimal approach to embedding domain knowledge into a neural network.

What is the optimal way to couple a deep-learning-based architecture with temporal domain knowledge?

Informed machine learning is a popular research topic, but it has not yet been applied to temporal underwater object detection. This study investigated the use of mathematical models as domain knowledge. It was found that while this approach is the most accurate, it is not the simplest. The models require input parameters that might be difficult to achieve. Additionally, the effectiveness of this approach depends on the type of objects being detected. When objects exhibit distinctly different trajectories, employing simple rules is more efficient. However, when the differences are more nuanced, accurate modelling is recommended. Furthermore, the modular setup of the system allows for straightforward integration of existing modules in the architecture, thereby enabling the inclusion of established and verified research.

The final subquestion addresses the limitations of the research and the generalizability of the findings.

What are the limitations of the study, such as the generalisability of the model on real-world data?

Given the absence of a dataset that meets the required specifications, the conclusions of this research were formulated based on the custom dataset. The approach is anticipated to work in a realistic underwater environment within the assumptions made. The pipeline is designed in a modular fashion, enabling adaptation to changes in the application by modifying certain elements to fit a specific environment.

Currently, the motion extraction module performs optimally when the speed of the object is significantly higher than the environment. This might not always be the case in reality and the module should be enhanced to handle such scenarios. Additionally, the method is developed to detect only single objects at a time, whereas in real-life situations, multiple objects are often present.

This study has proposed a pipeline that combines motion extraction and a domain knowledge-infused training process to develop a trajectory-classifying neural network. This network is designed to be robust against noise and poorly annotated data. This pipeline is however not without its limitations and performs optimally in specific conditions. To enable fully autonomous cleanup, the proposed setup should be augmented with existing state-of-the-art models to improve overall performance.