Segmentation of Maritime Vehicles from Video Sequences

Maximilian J. Vieweg

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Advisor(s)/Supervisor(s): Prof. Alexandre José Malheiro Bernardino
Prof. Maria Margarida Campos da Silveira

July 2024

DECLARATION

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Abstract

The monitoring of maritime vessels is of great importance due to the significance of shipping in global trade. While shore-based systems, e.g. coastal radar or imaging, are effective for static scenarios, such as for ports, sensor-carrying vehicles, e.g. UAVs or aerial vehicles, are of great value in off-shore scenarios. The use of video sequences for segmentation on these mobile sensor-carrying platforms is promising given the advantages in power, weight, and space of RGB cameras. The additional information obtained from the video segmentation masks offers advantages in identifying the heading of maritime vehicles, as well as for improved human-machine interfaces during monitoring.

The variation in imaging conditions, due to weather, sun, glare, waves, and foreshortening, makes this a nontrivial problem. Some of these conditions depend on the motion of the camera. Information about the position and orientation of a vehicle's camera is often available through its navigational systems. The availability of data, as well as motion correlations, suggests that the inclusion of camera pose information could aid in the performance of computer vision methods. In this master thesis, three methods for including camera pose information to enhance video segmentation quality are investigated. First, a transformer-based model is extended to estimate camera motion parameters in addition to video segmentation masks. Next, two 3D U-Net networks were modified to predict either specific camera pose parameters or the motion field. After extensive experimentation and statistical analysis, we were able to demonstrate that knowledge of camera pose and motion does not improve the segmentation quality of the model when including them as additional tasks during training. Experimentation with different representations of camera movement, different loss functions, and network types showed that there were no statistically significant improvements during segmentation in a multitask learning framework.

Keywords: Video Segmentation, Multitask Learning, Remote Monitoring, Maritime Vehicles

Contents

List of Tables

List of Figures

Acronyms

AP	Average Precision. 12, 40
AVOS	Automatic Video Object Segmentation. 8
bpy	Blender Python API. 20, 22
CNN	Convolutional Neural Network. 4, 8
FFN	Feed Forward Neural Network. 29
IoU	Intersection over Union. 10
ISR Lisboa	Institute for Systems and Robotics. 18, 19
IVOS	Interactive Video Object Segmentation. 8
KNN	K-Nearest Neighbors. 15
LSTM	Long-Short-Term Memory Network. 15, 47
NLP	Natural Language Processing. 5
SVM	Support Vector Machine. 16
SVOS	Semi-automatic Video Object Segmentation. 8
UAV	Unmanned Aerial Vehicle. 16
USV	Unmanned Surface Vehicle. 17
VOS	Video Object Segmentation. 8
VSS	Video Semantic Segmentation. 8

Chapter 1

Introduction

Given the importance of shipping in global trade, the monitoring of maritime vessels is of great importance. The surveillance of ships in ports, during the transport of goods and for the purpose of managing marine traffic, underlines the need for ship detection systems. This master's thesis aims to improve the segmentation of maritime vessels from video data, due to the additional information obtained from the ship contours. To this end, a neural network-based computer vision approach is used.

1.1 Motivation

Currently, approximately 80% of global trade is based on shipping [9]. This results in the need for an accurate way to monitor marine traffic. Additionally, maritime surveillance, law enforcement, and environmental objectives are a concern. Given the increasing automation of ports, there is a constant need for autonomous systems for their surveillance [10]. Similarly, the sustained threat of maritime piracy leads to increased costs for shipping companies, resulting from ransom payments, as well as the cost of armed guards on the ship [11]. This leads to the need of automated solutions, freeing personnel from the manual task of holding lookout.

1.2 Problem Definition

Currently, coastal radar is frequently used to monitor marine traffic. Although it has long-range sensing ability and is good at detecting large metallic bodies, they are susceptible to adverse weather conditions. Nevertheless, large bodies with a small radar cross-section, may still remain undetected [12].

Similarly, approaches that are derived from remote sensing can provide very detailed monitoring of ship positions. However, the low temporal resolution of satellite imagery requires the use of more time-sensitive methods [13].

Monitoring marine traffic using image and video data offers advantages in the form of high temporal and spatial resolution [14]. Although stationary approaches are suitable for applications in ports and harbors, there is a need for mobile surveillance in situations that involve monitoring environmental parameters, law enforcement, and piracy [10]. Aerial vehicles offer the advantage of a large field of view while being able to be readily deployed in a variety of operational scenarios. Fig. 1.1 shows an example of an aerial vehicle designed and manufactured by the Portuguese Air Force Research Center [1], as well as an exemplary image recorded using the same vehicle.

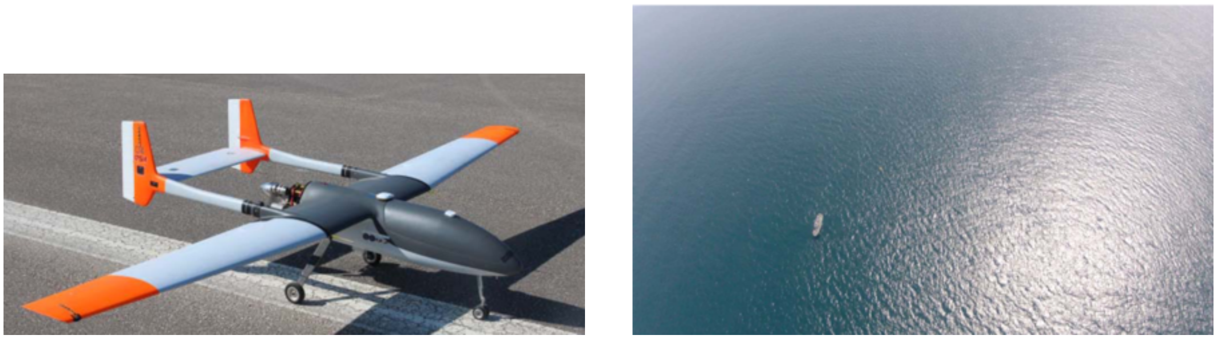


Figure 1.1: The aerial vehicle used for recording the Seagull dataset [1] (left). An example of an image recorded with the aerial vehicle [1] (right).

Segmentation is a computer vision problems that requires image data. Since in this case, the sensor-carrying platform is an aerial vehicle, it is assumed that information on the current altitude, attitude and heading can be paired with the images taken from an RGB video camera. This sequence of images is assumed to be temporally correlated, i.e. previous images help in predicting the following ones. Similarly, a certain frame rate can be taken as given (e.g. 30fps for a regular camera), as well as approximate continuity in the movement of the camera platform.

1.3 Challenges

The problem of segmenting ships from aerial images has several challenges. First and foremost, while the problem may seem trivial during perfect conditions, changes in weather and lighting significantly impact the performance of computer vision approaches. Similarly, phenomena such as ocean waves, white hats, and glare make segmentation more difficult [2]. Fig. 1.2 shows an example of glare encountered in images, as well as the sea foam created during the movement of a boat.

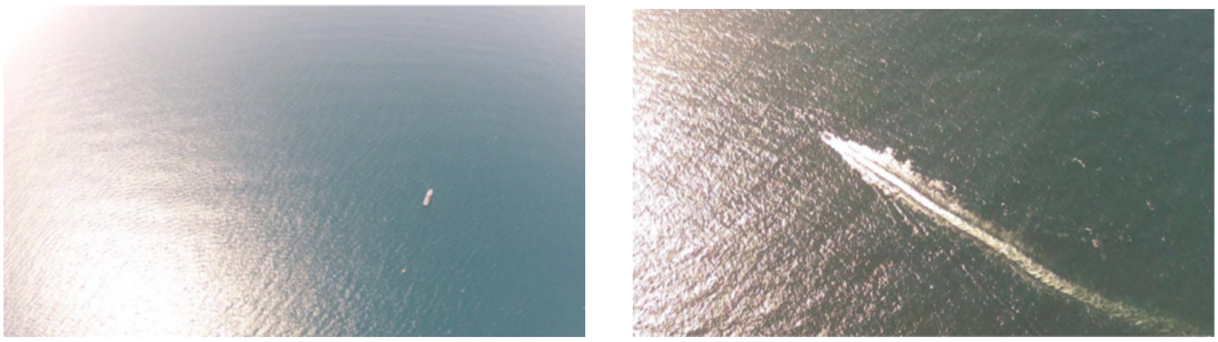


Figure 1.2: Two images from the Seagull dataset depict challenges during segmentation [2]. Glare and sea foam stemming from ship motion improve the difficulty of obtaining accurate results.

Current methods for creating image-based ship segmentation models are based on learning from data. Providing an adequate amount of data is therefore important for the model to work well with previously unseen images. While there are many datasets available, which provide images of ships from an aerial point of view, there may be a need for additional videos for the computer vision model to perform well. This underlines the need to generate an artificial data set for training [2].

1.4 Aims and Objectives

The aim of this thesis is to create a computer vision model, to segment maritime vessels from video data taken by an aerial vehicle. Changes in imaging conditions, due to e.g. weather and glare, are sometimes dependent on the movement of the vehicle's camera. Improvements in segmentation performance to current state-of-the-art models are sought, by incorporating data stemming from the camera pose. Additionally, the creation of synthetic training data might be necessary due to the limited availability of scenario-specific datasets.

1.5 Contribution

The contribution of this master's thesis lies in the creation of a new neural network architecture for the segmentation of image sequences that can incorporate information stemming from the aerial vehicle's attitude and heading, as well as their changes in time.

The research question to be explored is the following:

> *RQ: Does including camera pose information during the training of a neural network improve segmentation quality in a multitask learning framework?*

To this end, multiple methods of including the additional information are explored in the multitask learning framework. They are then extensively evaluated according to statistical tests.

Chapter 2

Background

This chapter gives an introduction to the theoretical background of the proposed thesis. First, neural network architectures, such as convolutional neural networks and transformers, are described. Then two computer vision tasks are introduced, namely object detection, image-, and video segmentation. A background on multitask learning, as well as projection matrices, and motion field is given. Finally, the evaluation of segmentation methods is described.

2.1 Convolutional Neural Networks

As the name suggests, convolutional neural networks (CNNs) mainly consist of convolutional layers: A kernel is convolved on the input, resulting in a feature map that is passed to the next layer. The layers are locally connected, allowing for the sharing of weights in their computation. This weight-sharing results in a reduction in the necessary parameters that need to be learned, making the training process more efficient. Similarly, due to the sliding of the kernel in the convolution operation, the exact location of features becomes proportionally translated between layers, making their representation independent of the position in the initial input. In addition, non-linearities and pooling layers, as well as fully connected layers, are introduced into the model structure. For example, in the case of image segmentation, the input and output of the model consist of a matrix representation of the image. The learned kernels of the model perform convolutions on the input. The final layer then represents the segmentation map [15].

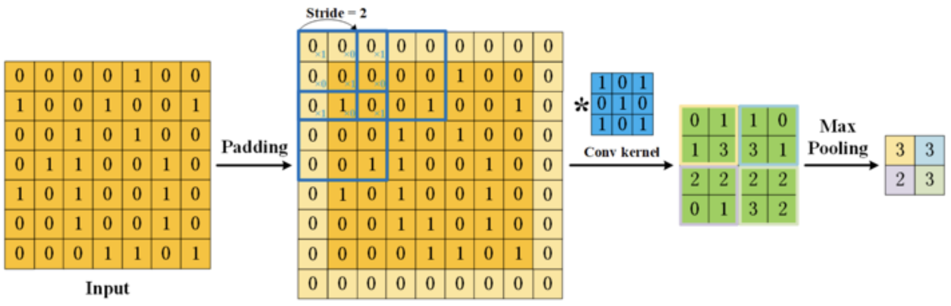


Figure 2.1: An example of a two-dimensional convolution is shown, along with explanations for the padding, kernel, and pooling layers [3].

Fig. 2.1 shows an example of the two-dimensional convolution operation. The input consists of a two-dimensional matrix that represents one channel of the input. The kernel, shown in blue, is convoluted on the input matrix to create an output matrix of different dimensions. The dimensions depend on the dimensions of the padding, stride, and kernel. The intermediate result, shown in green, is then max-pooled; that is, according to the size of a preset box, the maximum value of each box is used for the final output matrix. Multiple arrangements of these layers result in a network, whose weights, i.e. the parameters in the kernels, are trained to perform specific tasks, such as segmentation.

Some of these models are based on an encoder-decoder architecture which consists of two parts. In the first, the encoder, successive convolutional layers compress the input into a dense feature representation. This vector is then passed to the decoder, which generates the segmentation map [16]. By obtaining a semantically high-level representation of the object, important aspects about the input are hoped to be preserved and should aid in the decoding steps. Examples of these types of neural network are the U-Net [4], the V-Net [17], and SegNet [18].

The U-Net architecture is popular for image segmentation due to its performance and simplicity. It consists of several convolutional layers that compress an initial RGB image into a smaller semantic core. This core is then upsampled using deconvolutions to the original image size. To preserve details in the original image, skip connections are used that connect earlier parts of the model to the end. The initial implementation of the U-Net was taken from the following GitHub project[1].

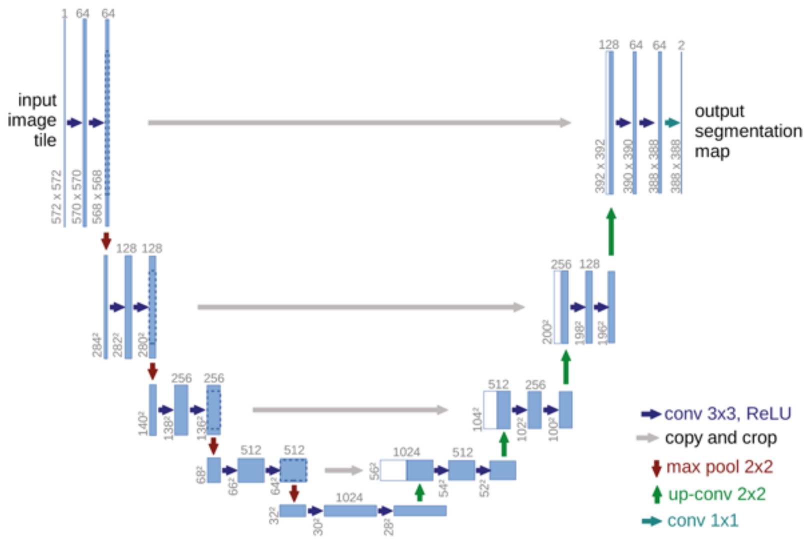


Figure 2.2: The architecture of the original U-Net is shown [4].

Fig. 2.2 shows the process graphically. The U-shape is given by the compression of the initial image tensor with subsequent upsampling through the deconvolutional layers.

2.2 Transformers

The performance of the Transformer architecture in natural language processing (NLP) tasks has led to its use in other domains, such as computer vision [19]. Considering that it is a sequence-to-sequence model, its application to image sequences is natural. Since many of the current state of the art models, as described in Sect. 3.1 are transformer-based, this section will give a brief discussion of the transformer architecture. The Transformer architecture is a sequence-to-sequence neural network, using attention mechanisms [5]. The following section gives a summary of the original transformer paper, as

[1] https://github.com/milesial/Pytorch-UNet

proposed by [5].

2.2.1 Architecture

The architecture is shown in Fig. 2.3 and consists of an encoder and a decoder block. The former converts its inputs into a multidimensional vector, which captures information about the whole sequence. These are information-rich representations of the input data that help to represent the underlying information in a way that makes it easier for the model to learn. Since the main objective of the model is to learn sequences, an input sequence is passed to the encoder one by one, generating embeddings that are used by the decoder. Past outputs of the model are used in addition to the input to obtain new sequences. The model generates the next probable output symbol in the sequence. In the case of language models, these are words; for video segmentation, segmentation masks are built.

The encoder and decoder are composed of multiple layers, represented as a gray box in Fig. 2.3. Each layer consists of two parts. In the first part, an embedding is fed into a Multi-Head attention block. The output of this block is then added to the original embedding and normalized. The normalized output is fed into the following part, where it passes through a feed-forward network. Its result is added to the original normalized input and then is further normalized itself. As indicated in Fig. 2.3, this process repeats Nx times. In the original paper $Nx = 6$ was chosen.

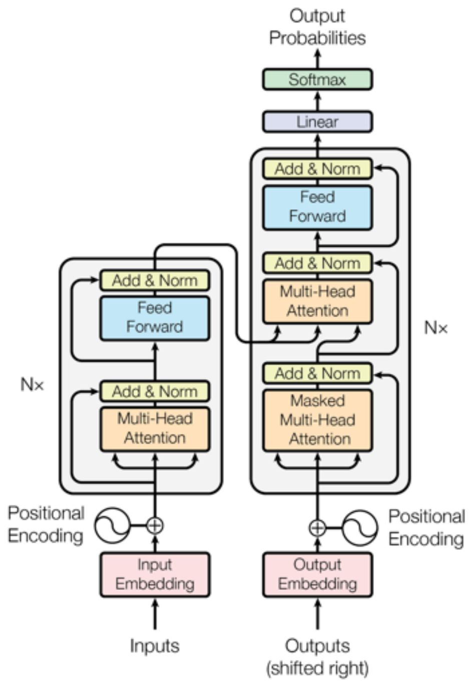


Figure 2.3: Transformer Architecture [5]: The Encoder block on the left generates embeddings that are passed to the Decoder block on the right. Each block consists of Multi-Head Attention layers, feed-forward networks and normalisation functions. To obtain a positional dependence, a positional embedding is added to the input.

2.3 Object Detection

Object detection is concerned with the classification and localization of objects in an image. Taking images as input, a bounding box is obtained, characterized by its width, height, and location (x, y).

Fig. 2.4 shows a ship, with its surrounding bounding box.

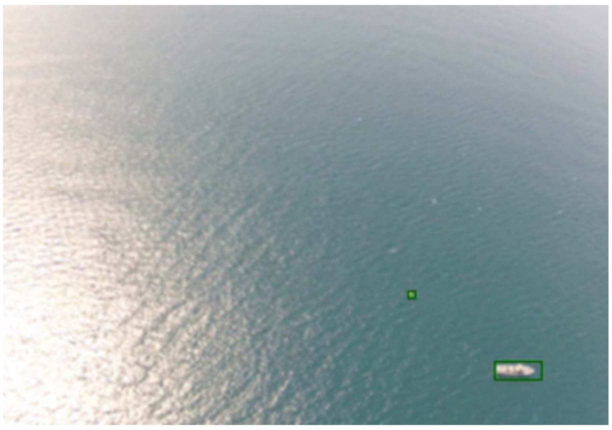


Figure 2.4: An example for the object detection of a ship. The detected object is depicted, surrounded by a bounding box in black [1].

Although traditional methods manually determine significant points in the frame based on predefined rules, they struggle with more complex image understanding. Modern approaches use deep learning-based approaches, which use neural networks and huge amounts of data for detection. The following section describes commonly used approaches [20].

2.3.1 Faster - RCNN

The Faster RCNN network is an end-to-end object detection architecture, which is faster and more accurate than previous approaches [21]. Earlier versions — such as RCNN [22] or Fast RCNN [23] — relied on multistage processes, which were comparatively slow and did not allow for the sharing of information between those stages. Subsequent versions, such as Fast RCNN, aggregate the extraction of features rather than recalculating them for each proposed region, thus saving time. Faster RCNN's integrate the region proposal network with the convolutional feature extraction layers. By sharing information between these elements, the network is significantly faster.

2.3.2 YOLO

YOLO is a neural network approach that frames object detection as a regression task [24]. This results in fast performance, leading to real-time object detection. An image is divided into a grid $S \times S$. From each of these cells, the bounding boxes B are predicted, each of which consists of the following parameters: $(x, y, w, h, confidence)$, where x and y refer to the coordinates of the center point and w and h to the width and height of the bounding box, respectively. In addition, a conditional class probability C is returned. The network is implemented as a CNN, where successive convolutional layers extract features and the final feed-forward network is responsible for the regression. Due to the success of the original architecture, many revisions have been made to the original architecture [25].

2.4 Image Segmentation

The goal of image segmentation is to identify objects in images. Unlike object detection, a pixel-wise correspondence is sought, called a segmentation map [16]. Objects can either be identified semantically, i.e. to what category of object does the pixel belong to, or by instance, i.e. also discriminating between specific objects in the frame. Fig. 2.5 shows an example of the image segmentation: A surfer is identified, and an orange segmentation mask is displayed.

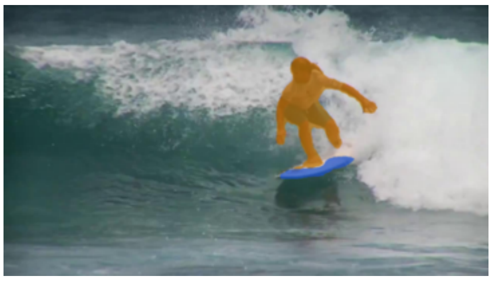


Figure 2.5: An example for the image segmentation of a surfer. The segmentation map is overlaid on the image in orange [6].

This section first describes convolutional neural networks (CNNs). Next, two different types of CNNs are introduced; U-Nets, and the YOLACT++ architecture.

2.4.1 YOLACT++

YOLACT++ is a real-time instance segmentation method that splits the image segmentation task into two simpler parallel tasks. In the first branch, instance-independent prototype masks are generated, which act as base vectors for the final mask generation. A second branch creates coefficients to linearly combine these masks for specific instances. These detections are then cropped and pass through a threshold, to obtain segmentation maps. The main advantage of the YOLACT++ architecture is its speed: Compared to other approaches, it maintains high performance, while achieving real-time results [26].

2.5 Video Segmentation

Video segmentation extends the field of image segmentation by considering not only single images but successive frames in a sequence. The field can be categorized very broadly into video object segmentation(VOS) and video semantic segmentation (VSS). While the former is concerned with discerning which objects in a scene belong to the foreground or background, the latter tries to detect categories of objects in the form of segmentation maps [27].

Video segmentation methods can also be categorized by the amount of human intervention needed. Automatic Video Object Segmentation (AVOS) does not require any human input and automatically chooses the objects in the frame. Semi-automatic Video Object Segmentation (SVOS) usually uses some human input in the first frame to determine which object is being segmented. Finally, Interactive Video Object Segmentation (IVOS) uses continuous user guidance throughout the segmentation process to obtain higher-quality results [27].

An additional categorization lies in the operating principle of the model. Video instance segmentation tasks can be divided into online and offline methods. The former performs instance segmentation on an image level and then tries to associate those instance masks along time. Offline methods directly predict 3D masks, which consist of two-dimensional image space over time [28].

2.6 Multitask Learning

In the multitask learning paradigm, the performance of a deep learning network is improved by learning similar tasks in parallel. Due to their similarity, internal representations of the input might complement each other, leading to an improved quality of output. Multitask learning is thus a method to achieve inductive transfer, that is, the transfer of knowledge between several sources of information [29].

For example, in [29], the primary task consists in predicting the steering direction of a vehicle, given an image. Several additional tasks are added, such as predicting the location of the edge of the road or the intensity of the road surface. Training the network with additional tasks yielded improvements between 15-30% in the steering task.

2.7 Projection Matrix

Projection matrices describe the transformation between three-dimensional points to a two-dimensional frame, as seen by, e.g. a camera. To obtain the pixels in a frame, the following transformation is carried out:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} , \tag{2.1}$$

where (u, v) describes the location of a pixel in the image frame and (x, y, z) describes the location of an image point in the three-dimensional space. The factor λ refers to a scaling factor to normalize the final row of $(u, v, 1)$ due to the representation in homogeneous coordinates. This representation simplifies the calculation by making it possible to write them in one transformation [7].

The projection matrix P can be described as follows:

$$\boldsymbol{P} = [\boldsymbol{K}]\,[\boldsymbol{R} \mid \boldsymbol{T}] = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} . \tag{2.2}$$

The matrix $\boldsymbol{K}$ describes the intrinsic properties of the perspective model. These normally stay constant throughout the scene and can be estimated using camera calibration methods or from reading from the appropriate datasheets. The second entry $[\boldsymbol{R} \mid \boldsymbol{T}]$ describes the extrinsic properties of the camera, i.e. its attitude with respect to an inertial world frame. Throughout a video sequence, these naturally change over time as the camera moves [7].

The entries r_{ij} describe the rotation matrix between the world and camera frame, t_i the translation vector between the world and camera origin. The parameter f describes the focal length, i.e. the amount the incoming image is scaled. The entries c_x and c_y translate the image in the camera frame, to align off-center coordinate systems in the resulting image [7].

2.8 Motion Field

In computer vision, the description of object motion is important for scene understanding. Motion fields give a description of the velocities of points in a three-dimensional space with respect to the camera [30]. They represent an ideal match between the movement in space and the resulting image in a camera frame. Mathematically, they can be described as two corresponding point velocities,

$$\begin{aligned} \boldsymbol{v_o} &= \frac{\partial}{\partial t}\boldsymbol{r_o} \\ \boldsymbol{v_i} &= \frac{\partial}{\partial t}\boldsymbol{r_i} \ , \end{aligned} \tag{2.3}$$

where v_o describes the velocity of in three-dimensional space and v_i in the image frame. The point r_i refers to the location of a point in the image and the point r_o is expressed in three-dimensional space. Their relationship is given by the following perspective transformation:

$$\frac{1}{f}\boldsymbol{r_i} = \frac{1}{\boldsymbol{r_o}^T\hat{\boldsymbol{z}}}\boldsymbol{r_o} \ , \tag{2.4}$$

where f is the focal distance, and $\hat{z}$ is a unit vector on the optical axis of the camera. The movement of these points with respect to the camera can be described with a derivative with respect to time.

These changes in location of object points give rise to differences in intensity, when capturing images with a camera, called optical flow. Although the motion field and the optical flow are closely related, effects such as lighting, (non)-Lambertian surfaces, and shadows greatly influence intensity changes, even if objects have similar motion fields.

2.9 Evaluation of Segmentation Methods

To evaluate the performance of video segmentation methods, multiple metrics have been proposed. Although storage requirements and speed play a crucial value, this section focuses on model accuracy, that is, the raw performance of a model [16].

2.9.1 Intersection over Union

The intersection over union (IoU) is a metric used to determine the quality of a detection or segmentation [27]. The comparison between the areas of a given ground truth and a system's estimation is calculated as follows:

$$IoU = \frac{|A \cap B|}{|A \cup B|} \ , \tag{2.5}$$

where A and B correspond to the areas that are being compared. Generally, a value of $IoU > 50\%$ is considered sufficient for detection [27]. For video input this metric can be extended by comparing the IoU values over time, or by giving the mean IoU ($mIoU$). Given some adaptations, an IoU loss function may be constructed [31].

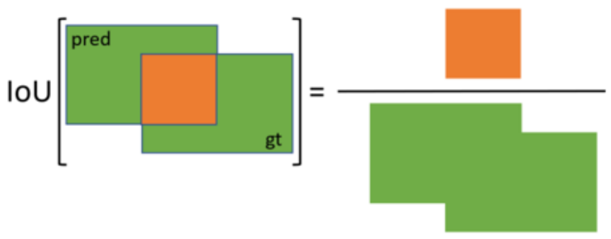


Figure 2.6: This figure shows a graphical depiction of the IoU: The intersection between the predicted and the ground-truth (gt) areas is divided by their union [7].

Fig. 2.6 shows a graphical depiction of the IoU metric. The intersection $A \cap B|$ is obtained between the predicted area and the ground truth and is divided by their union $|A \cup B|$. It gives a useful metric to compare the accuracy of detection and segmentation models.

2.9.2 Dice Score

The dice score measures the similarity of two sets, similar to the IoU score. It is defined as follows:

$$Dice = \frac{2\,|A \cap B|}{|A| + |B|}\ , \tag{2.6}$$

The advantage in using the dice score lies in its differentiability [17]. Therefore, it is possible to utilize the dice score as part of a loss function to increase the measured overlap between two segmentation masks and the commonly used for biomedical imaging tasks [32].

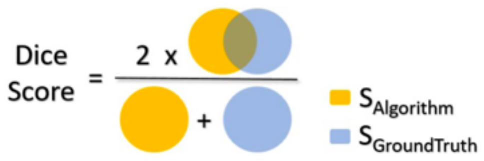


Figure 2.7: This figure shows a graphical depiction of the dice score: The doubled size of the intersection between the predicted and the ground-truth (gt) areas is divided by their respective sizes [8].

Fig. 2.7 shows a visualization of the dice score. First, the respective areas are intersected and doubled, to obtain a score of $dice(A, A) = 1.0$ for complete overlap. The resulting area is divided by the overall sum of the areas.

2.9.3 Precision and Recall

Precision and recall are two fundamental metrics for image segmentation [16]. The following two equations are used:

$$Precision = \frac{TP}{TP + FP}\ , \tag{2.7}$$

and

$$Recall = \frac{TP}{TP + FN}\ . \tag{2.8}$$

where TP, FP and FN refer to true positive, false positive and false negative respectively. Thus, precision is a measure for the amount of correctly identified samples, given all the positively identified ones, i.e. given the current sample, how many of those are correctly classified. In the case of segmentation, a sample corresponds to one pixel, whereas in detection it would refer to a detected object. Recall refers to the overall percentage of correctly classified samples, including those which were mistakenly identified to be negative.

These measures can be used in an object detection setting, where the quality of detection is important. For applications in segmentation, a threshold for overlap between the ground truth mask and the prediction is defined. Segmentation predictions can then be sorted into the upper categories.

2.9.4 AP Score

The average precision AP gives a measure of the quality of an object detection algorithm. During the calculation of the precision score, as seen in Eq. 2.7, the IoU threshold needed for a true positive detection is raised in a set number of steps between $(0.5, 1.0)$. For each value, the precision of the method is evaluated and saved. They are then averaged, to obtain a more descriptive metric that takes differing amounts of overlap into account. For example, the COCO Evaluator [2] is a popular tool that provides a unified interface to evaluate object detection data sets. Although the AP score is used mainly for object detection, its use in video segmentation methods is popular.

2.9.5 Bayesian Testing — Region of Practical Equivalence

Although null hypothesis significance testing is a common procedure to evaluate the baseline performance of a new model, it can sometimes be inadequate to determine statistical significance. A thorough analysis is provided by [33] along with an alternative approach.

In essence, they propose a shift from a frequentist perspective to a Bayesian one in the context of the classification problem. The former asks the following question: Given a hypothesis, how likely are you to encounter the following data? The null hypothesis is that the classifiers used are of the same performance. Since few classifiers are of equal quality, the null hypothesis is rejected most of the time. However, defying the null hypothesis does not give an estimate of the magnitude of the effect. Determining whether and by how much a new approach improves on the current state-of-the-art is, therefore, less certain.

Benavoli et al. therefore adopt the correlated Bayesian t-Test [34]. They give an expression for the prior $p(x|\mu, \Sigma)$, i.e., what is the likelihood of obtaining the data, given the parameter to be estimated μ and its covariance Σ. The parameter μ describes the difference in accuracy between the two classifiers.

The calculation of the posterior $p(\mu|x, \dots)$, allows the generation of a graph of equivalence, as shown in Fig. 2.8

[2]https://cocodataset.org/

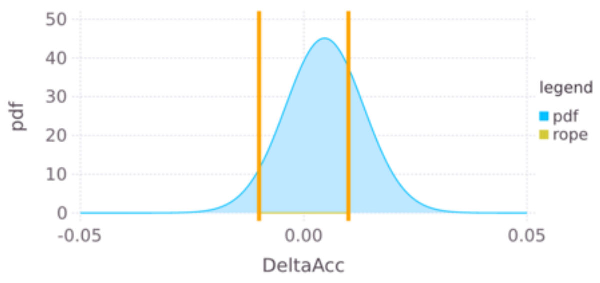


Figure 2.8: Example of a posterior distribution of the difference in classifier accuracies. The yellow bars define the region of practical equivalence. An integral over the probability density function gives the probability for the accuracy difference of the respective region.

The ”region of practical equivalence” (ROPE), indicated by the yellow bars and usually defined as being in the interval of $[-0.01, +0.01]$, shows the probability of whether the difference in precision between the classifiers is 1%. In simple terms: Fig. 2.8 plots the probability distribution of whether classifier A improves on classifier B. The integral over the distribution to the left/right of the ROPE gives the probability that A performs worse than B/B performs worse than A. Integrating in-between the two yellow bars, gives the probability of equal performance.

Chapter 3

State-of-the-Art

In this chapter, different methods for video segmentation in general are discussed. Then, specific methods for segmentation in maritime scenarios are explored. The inclusion of movement information is included next, as well as viable datasets for the training of neural networks.

3.1 Video Segmentation

Table 3.1 shows the main approaches in the literature to video segmentation in general:

Name	Type	Method	Notes
MaskTrack R-CNN [35]	Online	Mask R-CNN tracking head	Created YouTube-VIS
Mask2Former [36]	Offline	Extension from image segmentation	Directly predicts 3D segmentation volumes
MaskFreeVIS [37]	Online	Temporal KNN based loss	Only uses bounding box annotations
VisTR [6]	Offline	CNN embeddings for transformer	Includes instance sequence matching module
SeqFormer [38]	Offline	Deformable attention block	Aggregates box queries

Table 3.1: Comparison of Video Segmentation Methods

Each element is further explained in the following paragraphs.

MaskTrack R-CNN [35] is an extension of Mask R-CNN [39] for video instance segmentation. The original two-stage procedure, consisting of a region proposal network and features extractors, is extended by adding a tracking head, which assigns instance labels to the segmentations obtained on an image level. Its purpose is to match objects in-between the images and is thus an online method for video instance segmentation. Additionally, the paper proposes a new benchmark, the YouTube-VIS dataset, consisting of 2883 videos, along with instance masks for 40 categories.

The Mask2Former [36] is an architecture originally used for image segmentation that was extended to work with video data. Video sequences are treated as 3D spatio-temporal volumes of dimension (T, H, W), which describe the number of frames, height and width, respectively. To allow the model to differentiate between frames, a temporal positional encoding is added, similar to the positional encoding described in Sect. 2.2.1. To enforce causality, a masked attention mechanism is introduced into the image sequence. Therefore, the model is unable to use future images in a sequence to predict the current one during the training process. The result is a direct prediction of a whole 3D volume of two-dimensional images over time.

In MaskFreeVIS [37], only the annotations of the bounding box are used to predict the segmentations of instances in the video. Since labelling segmentations for video are time-consuming, image patches are matched across frames. A temporal k-nearest-neighbors based loss (KNN TK-Loss) is developed, which first identifies patches in the image that can be matched across frames. These are matched according to a one-to-many correspondence by selecting the lowest distance between patches. A final consistency loss between the found correspondences enforces the temporal stability of the masks across frames.

In the VisTR model [6], the sequences of images are first encoded into features by a CNN backbone, specifically a ResNet-50. Then, a positional and temporal encoding is added which encodes the position of pixels in each frame and in the image sequence. These representations are fed into a transformer encoder and decoder in order. The sequence of instance predictions is matched using an FFN, where the loss function encodes a pair-wise matching cost. Additionally, the final mask sequences are output by performing self-attention on the encoder-decoder output.

The SeqFormer [38], first creates embeddings from each frame. Along with an initial instance query, each frame passes through a deformable attention block. Note that each frame is processed individually at this point in the process. The output of each frame, called box queries, is aggregated and used by the mask, class, and box head for the final output.

Transformer-based models seem to be the current state-of-the-art for video segmentation tasks. Although their performance on general video segmentation datasets is well documented, an investigation into a specific application, namely the segmentation of aerial marine vessels, along with the inclusion of additional task-specific data, such as camera movement information, seems unexplored.

3.2 Detection and Segmentation of Marine Vessels

Table 3.2 shows a comparison of the methods applied in the marine use case.

Name	Modality	Type	Scenario	Method	Notes
[40]	Images	Segm.	Shore	Visual attention detection	Low-level features
[13]	Videos	Det.	Air	Convolutional LSTM	Task-specific loss
[2]	Videos	Segm.	Air	YOLACT++	MarSyn creation, real-time, 3D Cond. Rand. Field
[41]	Images	Det.	Shore	CNN + Saliency Map	Coastline prior, contrasts map corrects CNN
[42]	Images	Det.	Air	Convolutional SVM	Needs less training data due to SVM
[43]	Images	Det. / Segm.	Air	U-Net	Cascade model performing segmentation on detected regions

Table 3.2: Comparison between detection and segmentation approaches for marine scenarios.

A more comprehensive explanation of each method is given in the following section.

In [40] a framework for visual attention detection in maritime scenes is proposed. Low-level features, such as edge density and contrast density, are combined with a sea/sky classifier. These features are used to train a Naive Bayes classifier.

In [13], a long-short-term memory network (LSTM) [44] is combined with a pre-trained convolutional neural network for object detection. Thus, not only spatial, but also temporal features are used. Additionally, domain-specific knowledge on the average size is used to improve the modeling of the loss function, to finally improve performance.

The contributions of [2] are two-fold: First, an instance segmentation network, YOLACT++, provides frame-level instance masks. Their quality is improved by post-processing with 3D fully connected conditional random fields, to use information on the temporal correlation of the frames. For evaluation, a synthetic dataset, the MarSyn dataset, is created. It contains videos of maritime surveillance scenarios, namely, of ships from the air, along with accurate segmentation masks.

In [41], convolutional neural networks are used for object detection. First, information on the coast line is extracted using image processing methods, as well as with a saliency map obtained from a contrast-based saliency extraction algorithm. These priors are used together with the CNN for obtaining bounding boxes.

In [42], a method for object detection of maritime vessels, involving unmanned aerial vehicles (UAVs) is found. The model is built from successive layers of convolutional layers, reduction layers, which operate similarly to the pooling layers in CNNs, and ending with a classification layer. The main contribution is an architecture that needs relatively few data, due to the use of a forward supervised learning strategy for the support vector machine (SVM).

The detection and segmentation of marine vessels benefit from domain-specific assumptions taken when processing the respective images and videos. Information on coastlines, horizons, and average target sizes helps to improve model performance. It seems that the use of camera movement information has not yet been explored for the scenario of moving aerial cameras.

3.3 Learning of Movement

Table 3.3 compares the different approaches outlined below:

Name	Modality	Pretext Type	Network Type	Notes
[45]	Video	/	CNN	Unsupervised Learning
[46]	Video	/	CNN	Segm. from Optical Flow
[47]	2 Images	Image Transforms	CNN	Predict camera transform
[48]	Video	Rotation of Video	2DCNN	Pretrain on video rotation
[49]	Video	Label Prediction	CNN	Predict Motion characteristics on grid

Table 3.3: Neural networks that include movement in their training.

In [45] the motion of objects in an image sequence is used to learn object segmentation in an unsupervised manner, since single objects that move in between frames display similar characteristics regarding the optical flow of an image. They create motion cues, that is, superpixels based on motion, through non-local consensus voting [50], and use these to train a convolutional neural network to predict segmentation maps from single images.

In [46] an encoder-decoder model is trained on synthetic video data to obtain segmentation maps from optical flow models. They use a convolutional neural network that learns a representation of the optical flow and then upsamples it to generate high-resolution segmentation maps. The results are consequently improved using conditional random fields.

In [47], egomotion is used as a self-supervision signal to generate useful features for downstream tasks. A Siamese convolutional neural network is used to predict image transformations. This approach is applied once to random translations and rotations applied to the MNIST dataset as a proof-of-concept and then refined on the KITTI dataset. They find that training networks on predicting egomotion before subsequent fine-tuning on the target task improve model performance for specific tasks due to better feature representations.

In [48], a network is trained to recognize the degree of rotation applied to videos. By recognizing the semantics of the video, a spatio-temporal representation of the video content is achieved, which helps with subsequent fine-tuning on small datasets. They demonstrate this by achieving up to 20% improvements in relevant evaluation scores.

Wang et al. propose a method for video representation learning to create spatio-temporal features [49]. They divide each frame into partitions, such as grids, and let a network predict labels, such as the tile with the most amount of motion, etc. Pre-training on these unlabeled videos improves the performance of a three-dimensional convolutional neural network when fine-tuning the model for action recognition problems.

The literature review shows that CNNs are the preferred type of network, when attempting to include camera movement information in a neural network operating on video data. Some of the works try to directly predict motion parameters through the degree of rotations or image transformations applied between multiple images. Others use the movement of objects in the images as an indicator of either scene or camera movement. It seems that the direct inclusion of the camera movement of a mobile vehicle in the training of a neural network has not been extensively explored.

3.4 Maritime Video Segmentation Datasets

A summary of the main existing datasets for maritime video segmentation is shown in Table 3.4:

Name	Total No. Frames	No. Videos	Type	Scenario	Camera Pose
YouTube-VIS 2019 [35]	$\sim$ 400,770	4,453	Segm.	Varied	×
Seagull [1]	150,000	19	Det.	Air	×
MarSyn [2]	25,000	25	Segm.	Air	×
MarDCT [51]	12,547	25	Det.	Ground	×
SeaDronesSee [52]	54,000	208	Det.	Air	×
Airbus Ship [53]	192,556	0	Det.	Satellite	×
MODv1 [54]	4,454	12	Segm.	USV	✓
MODv2 [55]	11,675	28	Det. / Segm.	USV	✓

Table 3.4: Datasets for maritime object detection and segmentation. A focus is set on datasets for marine contexts.

There are several datasets for the detection of objects and the segmentation of maritime vehicles, however, only some of them include camera pose information. For example, the Airbus Ship dataset [53], includes ship images taken from satellite imagery. The Seagull dataset [1] contains images and videos of maritime vessels taken from a UAV. It is annotated for object detection and contains a varied number of scenarios and perspectives. MarDCT [51], provides videos of ships passing canals in Venice, as well as information for the detection, classification, and tracking of these ships. The SeaDronesSee dataset [52], includes maritime imagery taken from UAVs, including information on altitude and angle of the camera. MarSyn, a synthetic dataset created by simulating ships in a 3D rendering environment, consists of video sequences of ships taken by a UAV, as well as segmentation masks [2]. The MODv1 dataset [38] includes camera pose information for the detection of marine vessels to avoid obstacles from the point of view of an unmanned surface vehicle (USV). Similarly, MODSv2 [55] includes video data, along with camera pose information for object detection and segmentation.

To my knowledge, there exists no dataset for video segmentation of maritime vehicles from aerial images that specifically includes camera pose data. In this work, the focus is on datasets depicting maritime scenes from an aerial point of view with annotations for the segmentation of videos. Although videos from different scenarios, such as videos taken from USVs, might improve the recognition of maritime vehicles, the bias originating from points of view close to the water surface does not reflect the operational scenario. Given these observations, Seagull [1] and MarSyn [2] seem the most promising. However, the former only provides ground truth for the object detection task, that is, detection boxes. MarSyn meets all the requirements, while not containing information on the camera pose. Due to the development of this artificial dataset at the Institute for Systems and Robotics (ISR Lisboa), the metadata derived from the simulations is still available. This offers a way to extend the dataset to contain all relevant parameters.

3.5 Literature Gaps

From the literature review, it is clear that the video segmentation field is a relatively new field. Only with the proposition of large-scale datasets, as in [35], advances in the field are accelerated. Given the small corpus of research that applies object detection and segmentation techniques to the domain of maritime vessels, a research gap is found in applying video segmentation to aerial maritime vehicles. One of the main problems in applying segmentation to maritime scenarios comes from weather, the sun, glare, waves, and foreshortening. The application of models using video sequences as input is expected to reduce errors caused by periodic phenomena, such as waves. Similarly, there exists a correlation between the position and pose of the camera, and factors such as waves and foreshortening. A model that takes into account these error sources may improve segmentation performance for maritime scenarios. Additionally, due to the imminent availability of navigational information in these scenarios, the use of camera pose information is a natural way to obtain more information on the scene, potentially increasing the performance of previous approaches.

Chapter 4

Methodology

During this master thesis, a new approach to video segmentation will be created to improve the quality of segmentation of maritime vehicles from an aerial vehicle. Specifically, the integration of camera pose information during training from a synthetic dataset is expected to increase the quality of segmentation maps. Assuming that objects move slowly compared to the camera, knowing the position of the object in one frame, as well as the movement between the next frame, helps in predicting the next segmentation map. Building an internal representation of movement information should then allow the model to simultaneously improve its segmentation predictions.

First, additional ground truth information for camera movement must be extracted from existing segmentation datasets. This information is then used during the training of several types of neural networks, namely one based on transformers and 3D-CNNs. During training, each model predicts the relevant camera pose parameters to build an internal representation, in an effort to improve segmentation performance. This prediction is performed either as part of a regression task based on representations of respective networks or as the prediction of the motion flow of the scene.

In this chapter, first the dataset is described, along with additional data that was extracted from previous simulations. Next, different approaches to the problem, based on transformers and multitask learning are outlined.

4.1 MarSyn Dataset

The existence of a dataset with segmentation maps and camera pose information for videos taken onboard aerial vehicles is necessary to train a network on the desired scenario. Therefore, data for training and evaluation purposes will be synthetically generated. Given the proximity of MarSyn to the task, it is reasonable to extend the dataset with the desired parameters.

Therefore, the first phase of this master thesis consists of associating camera pose information from an aerial vehicle with each image in the video sequence. Given the availability of the MarSyn project code, due to its development at the ISR Lisboa, movement data from the aerial vehicle are obtained. Although the parameters describing the camera motion are not explicitly available in the dataset, they are contained in the files used to generate the simulation.

The MarSyn dataset consists of 25 video sequences, each having a length of 1000 frames [1]. The frame rate of the video is $24\ fps$. The data creation process includes the modeling of three-dimensional ship models, as well as their positioning and movement in Blender[1], an open-source 3D modeling software. Vessels of different types and sizes are simulated to move in an artificial ocean. A synthetic camera then captures images from a set trajectory in heights ranging from 150 to 1000 m. Fig. 4.1 shows a variety of weather, lighting, and operational scenarios that can be configured to capture a diverse range of images. The simulation environment allows highly precise segmentation masks to be obtained, compared to human annotations, without the annotation burden [2].

Fig. 4.1 shows three exemplary images of the dataset. The first image shows three separate maritime vessels sailing in the ocean. The Sun creates strong reflections on the waves, changing with the position of the camera. The second image shows a container ship, whose color closely matches the color of the ocean in some of the training videos. In the last image, a boat and the ocean are seen, as well as parts of the shore.

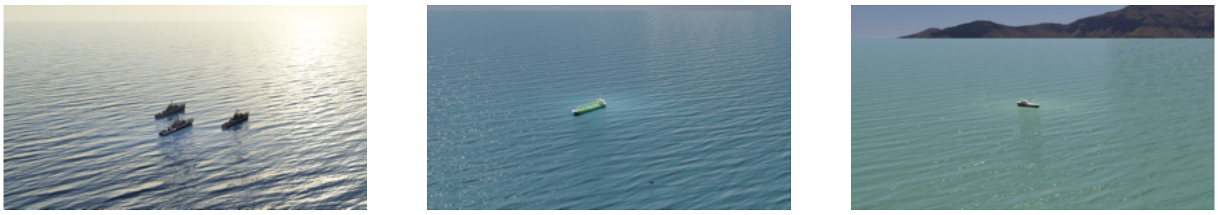


Figure 4.1: Three exemplary images of the MarSyn dataset are shown. Different lighting conditions, diverse background variations due to sea color and shore, as well as multiple ships are found in the dataset.

Since the ground truth segmentation masks were obtained automatically through a simulator and not hand-annotated, their quality is high. Fig. 4.2 shows an input image on the left, with the corresponding segmentation mask on the right. The details of the masks are seen clearly: The mast and piping, as well as an antenna, stand out from the background.

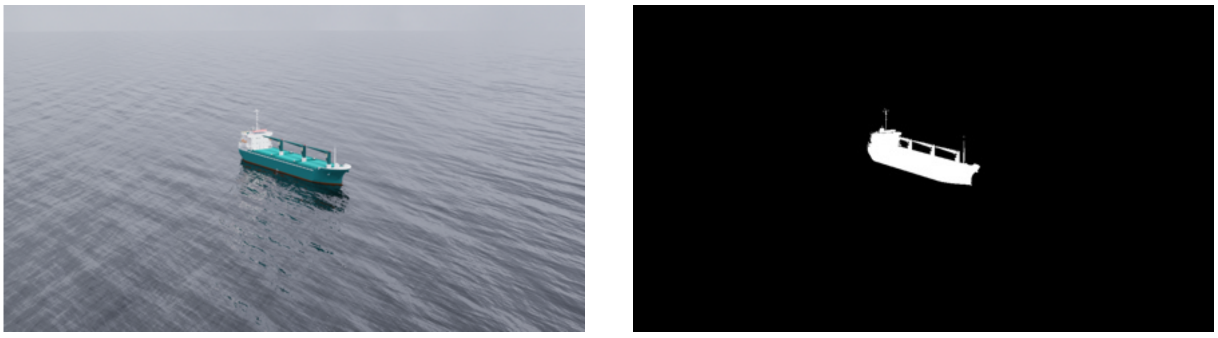


Figure 4.2: Example of an image (left) with the corresponding ground truth mask (right). The quality of the annotation is due to the automatic generation of the ground truth through the simulator.

4.1.1 Extracting the Camera Pose from the MarSyn Dataset

The original MarSyn dataset does not contain any information about the pose of the camera. However, these data can be extracted from the blender simulation files, through the blender Python API (bpy)[2]. This interface allows for the extraction of additional simulation information after run-time. For every frame of the video, information on the location and rotation angles of the camera is extracted. To obtain the linear and angular velocity between two frames, the corresponding values were calculated as follows:

[1] https://www.blender.org/

[2] https://docs.blender.org/api/current/index.html

$$\boldsymbol{v}_{lin} = {}^{w}\begin{bmatrix} v_{x,t} \\ v_{y,t} \\ v_{z,t} \end{bmatrix} = f \cdot {}^{w}\begin{bmatrix} x_{t+1} - x_t \\ y_{t+1} - y_t \\ z_{t+1} - z_t \end{bmatrix} , \tag{4.1}$$

where $f = 24\ fps$ is the frame rate of the video, (v_x, v_y, v_z) and (x, y, z) are velocities and locations in the world frame w, calculated at time t. The conversion to m/s using f was chosen not only to obtain metrics in conventional SI units, but also due to scale. During experiments, it was observed that the network learned these numerically larger values more easily. Finally, the vector is converted to the camera coordinate system.

Similarly, the angular velocity is obtained in the following way:

$$\boldsymbol{v}_{ang} = T_{ang}(({}^{w}\boldsymbol{R}_{t+1} \cdot {}^{w}\boldsymbol{R}_t^{-1})^f) , \tag{4.2}$$

where $\boldsymbol{R}$ describes the rotation matrix at timestep t, again in the world frame. The factor f again describes the frame rate and therefore the conversion to rad/s. The function T_{ang} is a transformation between different representations of angles. There are three ways to represent rotations in the following context: via Euler angles, unit quaternions, or rotation matrices. Although the calculations for deriving these quantities are obtained with rotation matrices, due to their ease of use, the quantities are saved as Euler angles and unit quaternions. It is experimented with which angular representation is easier for the network to learn.

To obtain the angular velocity represented in Euler angles, as well as in quaternions, the function T_{ang} is used, similar to the case of rotation angles. Although a manual conversion is trivial given the relevant identities, libraries such as $scipy$[3] were used to obtain the changes in representation.

Since the videos obtained from each simulation are filmed from the reference frame of the camera, it is beneficial to perform a change of coordinates of the camera pose values to match this representation. This transformation takes place as follows:

$${}^{c}\boldsymbol{R}_w = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} , \tag{4.3}$$

where r_{ij} describes the entries of a three-dimensional rotation matrix. The transformation is performed between the world and the camera frame.

In blender a coordinate system convention is predefined, as can be seen in Fig. 4.3.

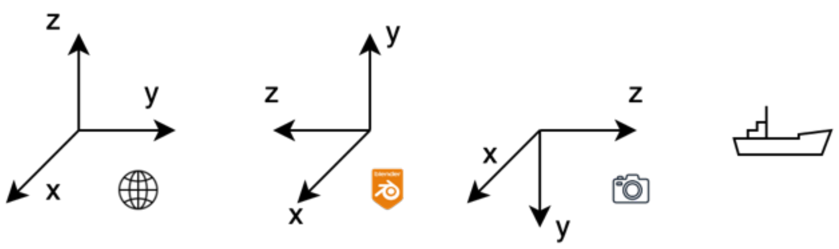


Figure 4.3: Relevant coordinate systems from left to right: The world coordinate system, the blender camera coordinate system, the standard computer vision coordinate system.

[3]https://scipy.org/

Fig. 4.3 shows the different coordinate systems that were used. The locations of the boats and the camera are described in the world coordinate system on the left. When extracting values to generate the rotations and projection matrices from the camera object using bpy, they are represented using the custom blender coordinate system shown in the middle. The z axis points in the inverse direction of the camera view, as indicated by the position of the boat on the right. In this thesis, the conventional computer vision coordinate system is used, where this coordinate usually points in the positive direction. To conserve right-handedness, the y coordinate is similarly inversed, to obtain the camera coordinate system to the right.

The data are modified in a few ways to more accurately represent the camera movement for the specific situation. For example, the absolute (x, y) world coordinates are arbitrarily chosen. Similarly, the initial yaw angle depends on the choice of the coordinate system. To establish a baseline in-between multiple videos, the relevant values are initialized to zero, to align the initial camera position and attitude. Other information, such as height, roll, and pitch angles, is independent of the coordinate representation. For example, the camera pitch includes additional information on the horizon line. The chosen camera pose parameters are shown on Tab. 4.1. Although not all of them are used in the subsequent steps, the extraction of many potentially relevant features allows for experiments.

Parameter	Notation	**Reference Frame**
Name	/	/
Frame	/	/
Location X	x	World
Location Y	y	World
Location Z	z	World
Pitch	r_x	Camera
Yaw	r_y	Camera
Roll	r_z	Camera
Linear velocity X	v_x	Camera
Linear velocity Y	v_y	Camera
Linear velocity Z	v_z	Camera
Pitch velocity	$\dot{r}_x$	Camera
Yaw velocity	$\dot{r}_y$	Camera
Roll velocity	$\dot{r}_z$	Camera
Quaternion w	w	Camera
Quaternion i	i	Camera
Quaternion j	j	Camera
Quaternion k	k	Camera
Quaternion velocity w	$\dot{w}$	Camera
Quaternion velocity i	$\dot{i}$	Camera
Quaternion velocity j	$\dot{j}$	Camera
Quaternion velocity k	$\dot{k}$	Camera

Table 4.1: The camera pose parameters that were extracted from the synthetic dataset are shown.

The upper parameters were extracted for each video in the data set. This allows for a visualization of the camera trajectory. Similarly, the orientation of the camera can be shown, to verify the accuracy of the data. The following figure shows one of the trajectories:

Fig. 4.4 shows a visual representation of the camera trajectory in the second video of the dataset over time. The taken path is marked in blue and shows a circular path, which rises towards the middle and then dives back towards the initial height. An arrow indicates the direction of the camera movement. During a constant sampling in time, the orientation of the camera is shown in red as a camera frustum. In this specific sequence, the vehicle, positioned approximately at $(0, 0, 0)$, is overflown, while the cam-

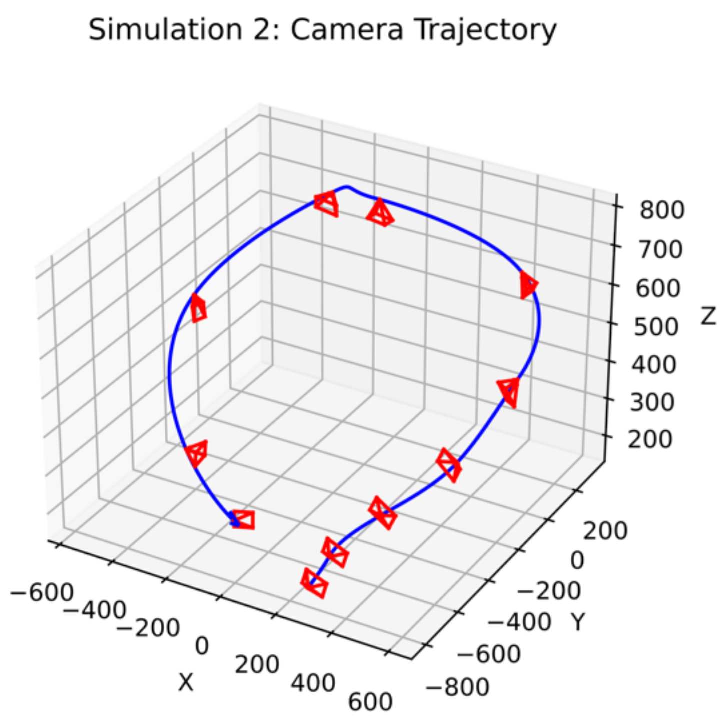


Figure 4.4: Trajectory of the camera for the second video of the dataset. The location over each of the thousand frames is shown in blue, while the camera view is indicated as a camera frustum in red.

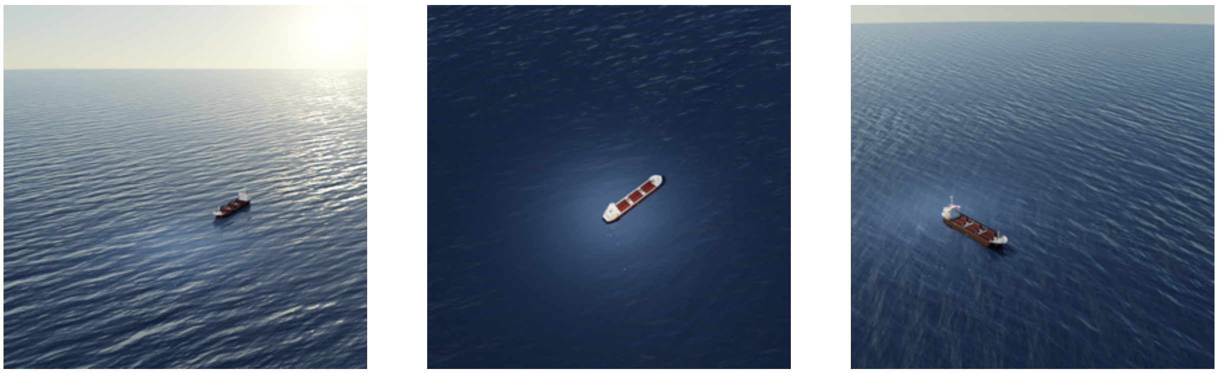


Figure 4.5: To illustrate the images taken along the trajectory, three images from the trajectory from Fig. 4.4 are shown. From left to right the images correspond to the third, eighth, and tenth camera view shown in red.

era consistently points in the direction of the boat. Although the camera views are shown in uniform time steps, the camera velocity changes. Slower velocities can thus be seen when camera views are clustered more closely in the graphic.

To verify the trajectories, three examples of camera trajectories are shown in Fig. 4.6. It can be seen that the trajectories are smooth and free of sudden changes in movement. Additionally, the camera mostly focuses towards a set point, to capture the boats in the video. In the simulation, the location of the boats is not static; they move throughout the sequence. However, this movement is small relative to the velocity of the camera. The full range of trajectories is shown in App. A.1.

A histogram of camera velocities is shown in Fig. 4.7. Linear velocities values are clustered around a mean of $\mu_v = 48.1 m/s$ with a standard deviation of $\sigma_v = 38.51 m/s$. There appears to be a collection of values around zero that stem from sequences with slow acceleration towards the beginning of the videos,

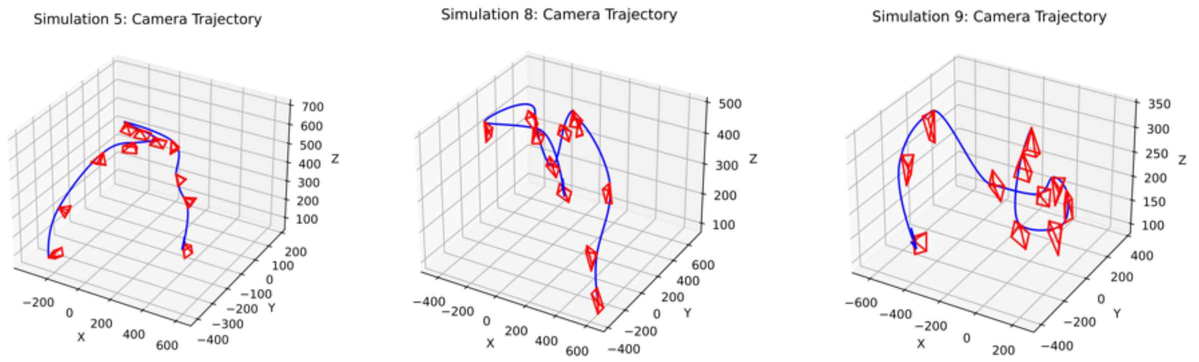


Figure 4.6: Three camera trajectories are shown from the simulations 5, 8, and 9 respectively. A variety of movement patterns is observed.

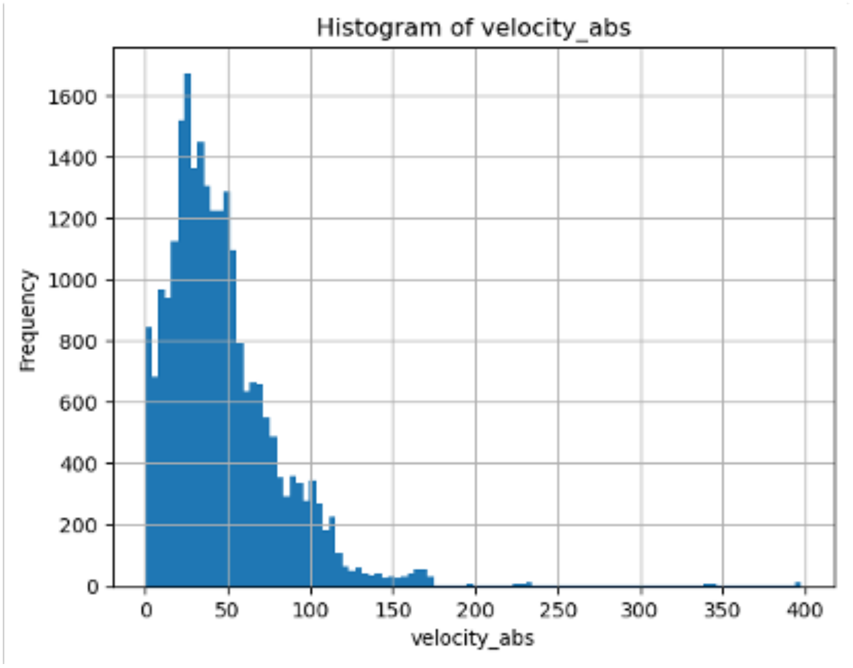


Figure 4.7: Histogram of the absolute velocity values of all the trajectories.

specifically mostly stemming from video 11. In Fig. 4.8, the angular velocity, measured in rad/s, is shown to be spread farther out: The mean is $\mu_a = 0.09rad/s$ and the standard deviation is $\sigma_a = 0.07rad/s$.

Fig. 4.9 shows the change in linear velocity over time for the first three videos. The camera starts without any initial velocity and then accelerates throughout the flight. Importantly, the velocities are noise-free. In datasets obtained from real-life scenarios, noise stemming from sensors and uneven flight paths can be a limiting factor, which requires denoising with e.g. Kalman filters. In this artificial dataset, noise was not added, facilitating data processing.

Fig. 4.10 shows a histogram of the absolute velocities of the first eight videos of the dataset. As can be seen, the range of velocity values ranges up to almost 200 m/s, which could be verified by looking at the timestamped locations of the camera. Values around zero are mostly due to the camera starting out from a resting position and slowly accelerating over the course of multiple seconds.

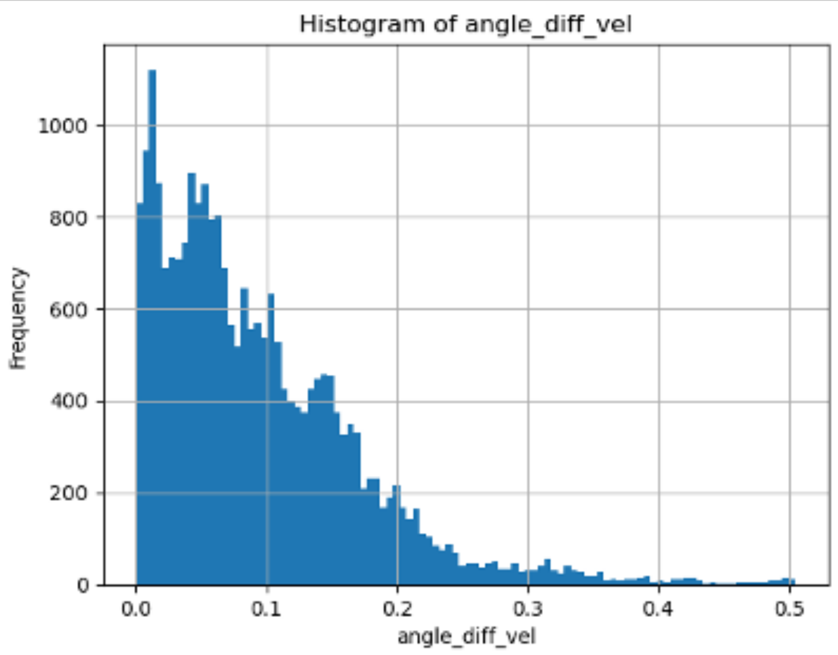


Figure 4.8: Histogram of the absolute angular velocity difference values of all the trajectories.

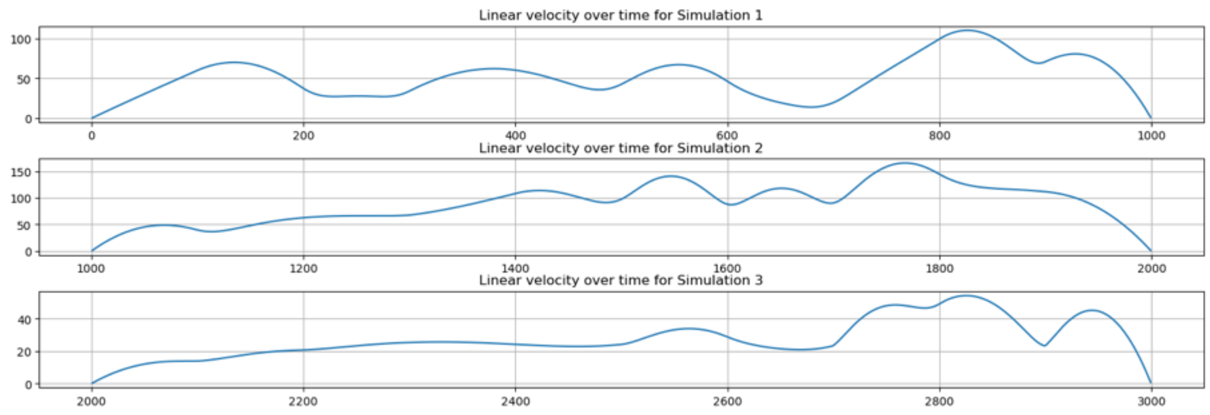


Figure 4.9: Change in linear velocities for selected videos over time in m/s. Each video starts from a resting state and then accelerates. Note, that the velocities are smooth. Due to the generation of the videos in a simulation, and the decision against adding noise to the camera movement, additional filtering is not necessary.

4.2 Mask2Former and Camera Pose Regression

Transformer-based models show good performance in video segmentation tasks because of their performance in modeling sequences. The Mask2Former module was chosen because of its performance on video segmentation benchmarks. Additionally, the model architecture is modular: Different backbone and pixel decoder modules may be used in the original model, which allows versatility in the representation of the video object.

Fig. 4.11 shows the architecture of Mask2Former on the left and the transformer decoder in detail on the right. First, a video of variable length is encoded using a backbone, such as a ResNet-50. This representation is then decoded to different scales, using a pixel decoder. The transformer decoder takes these different scales as an input and outputs the final segmentation masks and class probabilities. On the right, the input of the image feature and the query features, vectors representing image classes, are

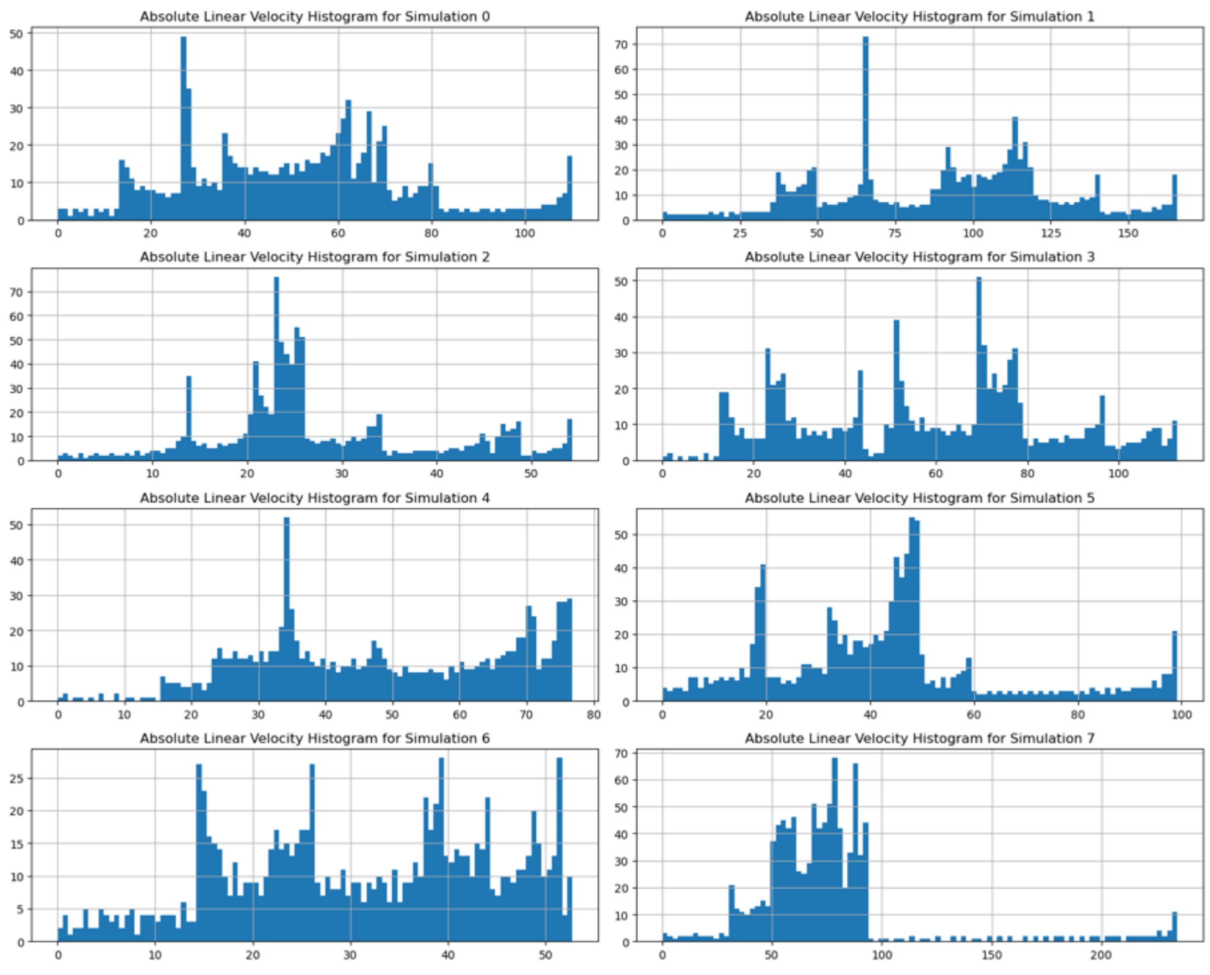


Figure 4.10: Histogram of the absolute value of the linear velocity, shown for every video.

taken as input to the masked attention model. Masked attention represents an additional term in the usual attention block and focuses on regions where previous masks were placed.

The proposed change in the architecture is shown as a red arrow in Fig. 4.11: In addition to predicting the segmentation masks, the current pose of the camera in the camera frame is also predicted. It is hoped that this additional information creates a representation in the layers that the segmentation block can utilize to create improved segmentation. To this end, an additional loss term, an adaptation of the PoseNet loss [56] is introduced. It is adapted by adding additional terms, such as velocities, in addition to the absolute location and rotation of the pose. It is implemented as follows:

$$\mathcal{L}_{pose} = \sum_i \gamma_1 \, \|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2 + \gamma_2 \, \|\boldsymbol{q}_i - \frac{\hat{\boldsymbol{q}}_i}{\|\hat{\boldsymbol{q}}_i\|}\|_2 \tag{4.4}$$

where $\hat{x}_i$ and $\hat{q}_i$ describe the predicted linear and angular positions with respect to the initial reference frame, respectively. The index i is used to iterate between stationary quantities, such as locations and rotations, and velocities, such as linear and angular velocities. These values are summed over the respective camera pose values to be considered, such as location, rotation, and linear/angular velocity vectors. The term $\hat{\boldsymbol{q}}_i$ requires normalization to preserve rotational qualities, such as length. The ground truth values are shown without the accent, and $\|\cdot\|_2$ indicates the L2-norm. The factors γ_i describe the factors used to scale the respective norms to a similar range. They are chosen to be $\gamma_i = 1.$, except for camera locations in three-dimensional space, where they are assumed to be $\gamma_j = 0.01$.

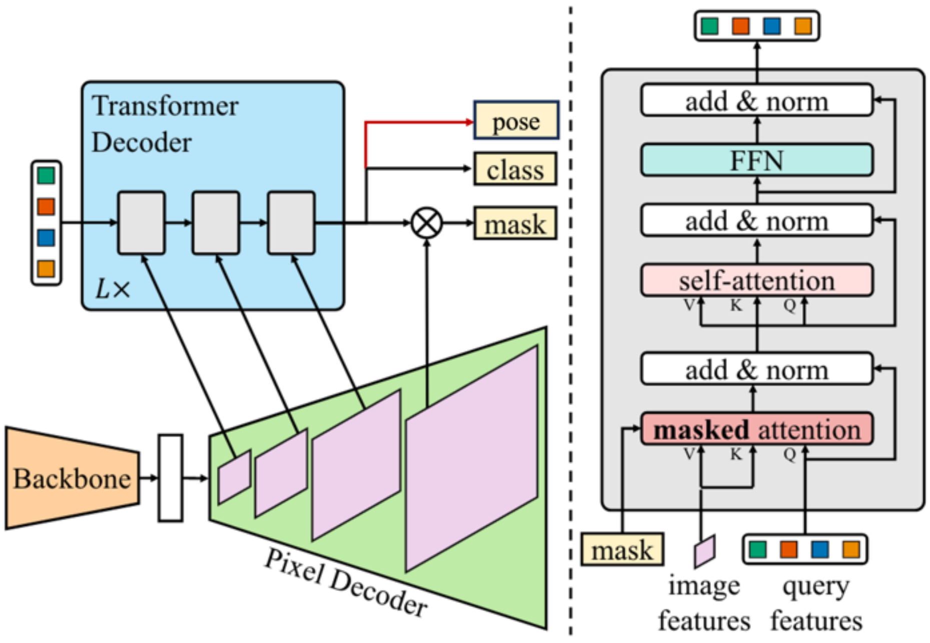


Figure 4.11: Mask2Former Architecture: (left) A backbone encodes a video volume and a decoder scales it to different sizes. These are fed into a transformer decoder, which then outputs the segmentation masks and class probabilities. (right) The transformer decoder block. Query features, describing the possible instances in the frame, and image features from the decoder are both sent into a masked attention module.

Fig. 4.12 shows the loss curves obtained during an exemplary model training run. In the top row, the cross-entropy and dice loss are seen. These two are combined in mask loss, as seen in the bottom right. The new addition of the pose loss is shown to the right. Although the network seems to learn parts of the angular position, the loss seems to reach a plateau.

Parameter	**Value**
Learning Rate	0.0001
Images per Batch	4
Weight Decay	0.05
Gradient Clip	0.01
Iterations	2000
Loss Weight: Class	2.0
Loss Weight: CE	5.0
Loss Weight: Dice	5.0
Loss Weight: Motion	2.0

Table 4.2: Mask2Former Training Parameters

Since Fig. 5.3 shows a slight improvement using the pose loss, an additional investigation was performed. The original model was pretrained on the YouTube-VIS dataset, and subsequently fine-tuned on the MarSyn dataset, extended by the camera pose information. The video sequences used for training are $(1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25)$; for testing the following videos were used: $(2, 13, 14)$ Information on the parameters used during training is shown in Tab. 4.2.

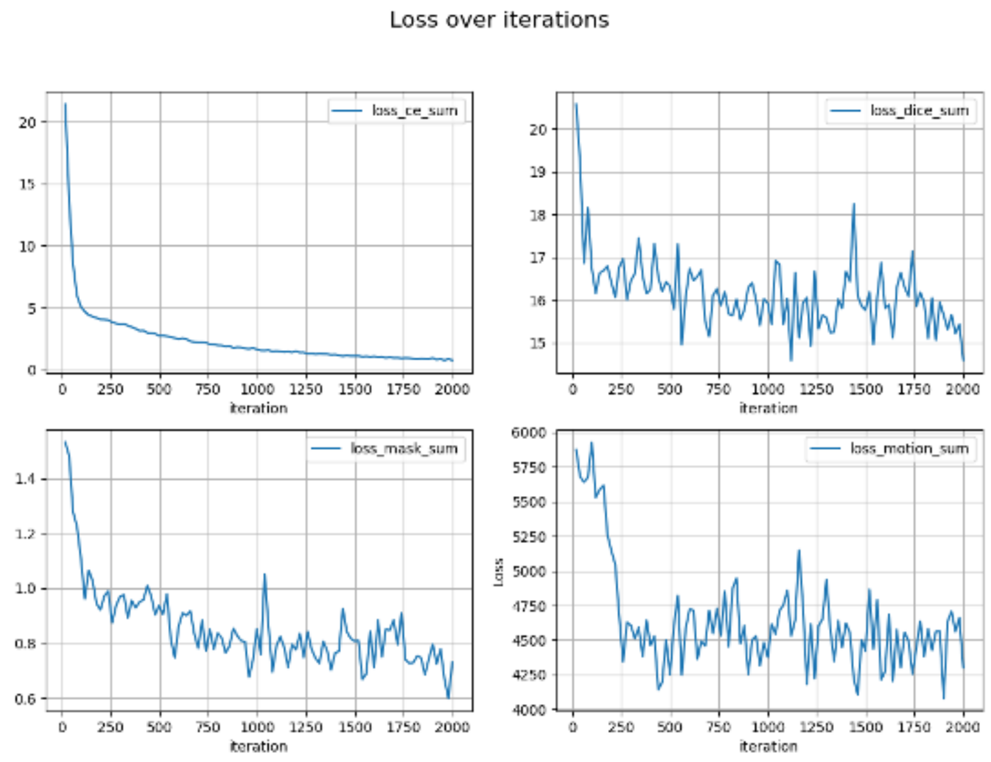


Figure 4.12: Loss curves for the Mask2Former architecture. The cross-entropy loss is shown on the upper left, along with the dice loss to its right. The lower left shows their sum, named the mask sum. On the lower left the pose loss, here denominated motion loss due to the use of the angular velocities, is shown.

4.3 Multitask Learning Approach

Given the additional ground-truth data on the camera movement, a multitask learning approach is chosen for encoder-decoder architectures. The loss function is enriched with additional terms to include information on the motion of the camera. The aim is to let the model represent the camera motion in its parameters to aid the prediction of segmentation masks.

The values chosen for the respective blocks of the U-Net architecture are shown in Tab. **??**.

Name	**In**	**Out**	**Block**
Inc	3	64	DownConv
Down 1	64	128	DownConv
Down 2	128	256	DownConv
Down 3	256	512	DownConv
Down 4	512	1024	DownConv
Down 5	1024	2048	DownConv
Up 1	2048	1024	UpConv
Up 2	1024	512	UpConv
Up 3	512	256	UpConv
Up 4	256	128	UpConv
Up 5	128	64	UpConv
Out	64	1	Conv3D

Table 4.3: The parameters used for the segmentation path of the 3D U-Net architecture.

The $DownConv$ first performs a three-dimensional pooling operation with the kernel size $n_k = 1$, stride $n_{stride} = 2$, and a padding $n_{pad} = 1$. The output then reaches the $DoubleConv$ block, which is made up of two equal subblocks, each consisting of a three-dimensional convolution using a kernel size $n_k = 3$ and a padding $n_{pad} = 1$, followed by a three-dimensional batch normalization and a rectified linear unit (ReLU). The $UpConv$ layers first perform a bilinear upsample operation with a scale factor of $n_{upsample} = 2$, and then pass through the $DoubleConv$ block with the dimensions set in Tab. **??**.

U-Nets are traditionally used for image segmentation. To extend their input to video, the subsequent images of the input are stacked in the form (C, T, H, W), indicating the time, channels, height, and width of the images. The stacking of regular images of dimension (C, H, W) leads to a sequence that can be accessed through the time term T. Additionally, the kernels and pooling layers of the convolution are adapted to be three-dimensional. The pooling layers are chosen to only pool the representation on an image-to-image basis. Otherwise, the number of input frames is limited by the amount of vertical layers. Tab. **??** shows the magnitude of the chosen parameters.

Parameter	Value
Sampling Rate	3
Number of Images	5
Batch Size	1
Learning Rate	0.001
Momentum	0.8
Linear Learning Rate Scheduler	1.0 to 0.1
Epochs	1
Maximum Iterations	2000

Table 4.4: The values of the chosen parameters during training are shown in the table.

The hyperparameters used during the training of the model are shown in Table 4.4. The image sequences are loaded in input samples of $n_g = 5$ consecutive images. Due to the high frame rate, the changes between images appear small, diminishing the possible positive impact of including movement information in a neural network. A sampling rate of $f_s = 3$ is chosen; meaning that every third image is included in the input data.

4.4 Camera Pose Regression as an Additional Task

Fig. 4.13 shows a high level of model representation. The general structure is an encoder-decoder model, where the encoder generates a more semantically compressed feature vector, z, from the incoming video. The vector z is expected to contain higher-level information about the image sequence. Using this information, along with the skip layers from previous convolutional layers, a video decoder outputs a sequence of segmentation masks. As established earlier, the movement of the camera has a direct impact on the location of objects in the frame. It might therefore be beneficial for the model to build an internal representation of the camera pose. The pose regressor module uses the hidden representation z, to predict certain parameters of the camera movement. Alternatively, predicting both the segmentation masks and the camera pose parameters, z should contain information on both segmentation and camera pose, through the backpropagation of errors in the training step.

The regressor module is a Feed Forward Neural Network (FFN), which takes as input a flattened version of the latent representation z. The size of z, depending on the specific dimensions of the convolutional and input layers, can range to 100,000s of parameters for the single layer. Consequently, the change given by an update through the gradient can increase the magnitude of unstable regions, disturbing the

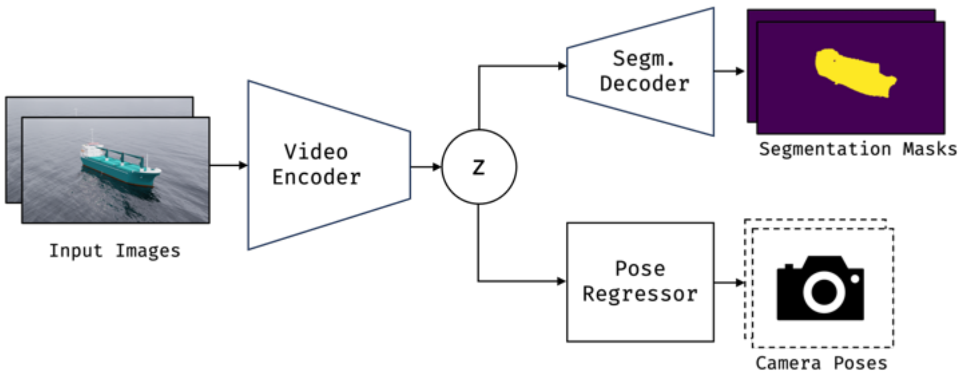


Figure 4.13: Simplified model of the pose regression approach. A series of input images is transformed into a latent representation z using a video encoder. This representation z is then both used to generate a series of segmentation maps and for camera pose regression, by the corresponding decoder blocks.

training. During experiments, downsampling the representation z to a constant length $dim(z_r) = (1000,)$ led not only to memory benefits through a decreased parameter count of the regressor but also to results with higher accuracy after training.

The trajectory of a camera directly impacts the movements of objects in the camera frame. Linear and angular velocity values affect the change in vessel location in several images. Similarly, the absolute values of the pitch and roll angles, as well as the height of the camera, give information on the positions of the object and the horizon. Throughout this thesis, a combination of these values is predicted through the network as an additional task, which we expect to be beneficial for the model to predict.

For training, the following composed loss function is used:

$$\mathcal{L} = \mathcal{L}_{ce} + \beta_1 \, \mathcal{L}_{dice} + \beta_2 \, \mathcal{L}_{pose} \ . \tag{4.5}$$

The goal is to update the segmentation and pose losses simultaneously, using the terms $\mathcal{L}_{ce}$ for cross-entropy, $\mathcal{L}_{dice}$ and $\mathcal{L}_{pose}$. Factors β_i are multiplied by the dice score $\mathcal{L}_{dice}$ and the pose loss $\mathcal{L}_{pose}$, to balance the ambiguities of the initial scale and to have a factor determining the relative importance between the two losses. During training, these were set to $\beta_1 = 1.0$ and $\beta_2 = 0.5$ to reflect the importance of the segmentation task.

For segmentation, there are multiple commonly used loss functions. Binary cross entropy, dice, L1, and L2 losses are the most widely used. To remain consistent with implementation of the U-Net architecture, a cross-entropy loss is used:

$$\mathcal{L}_{ce} = -\frac{1}{N} \sum_{i=1}^{N} p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i) \ , \tag{4.6}$$

where p_i refers to the ground truth target value of a pixel i, and $\hat{p}_i$ refers to the model output logit. In general, binary cross-entropy loss is used as a measure of the difference between two probability distributions for classification problems [57]. In this case, we classify between two different categories; boats and non-boats. This can be easily extended by defining categories for different types of ships, the horizon, or land.

The dice loss is given as follows:

$$\mathcal{L}_{dice} = 1 - 2 \cdot \frac{1 + \sum_i^N \hat{y}_i y_i}{1 + \sum_i^N \hat{y}_i^2 + \sum_i^N y_i^2} \,, \tag{4.7}$$

where x_i and y_i describe the output of the model and the ground truth target label for a pixel i, respectively. Through the additional terms of 1 in the numerator and denominator, the dice score is bounded to the interval $(0, 0)$, where a higher score indicates a worse quality segmentation. By inverting the score and repositioning it in the same interval, a dice loss $\mathcal{L}_{dice}$ is found, which rewards a high overlap between the predictions and the ground truth values. Although the value IoU is often used as a quality indicator for segmentation, it is not differentiable, making it impossible to use in a deep learning-influenced backpropagation setting.

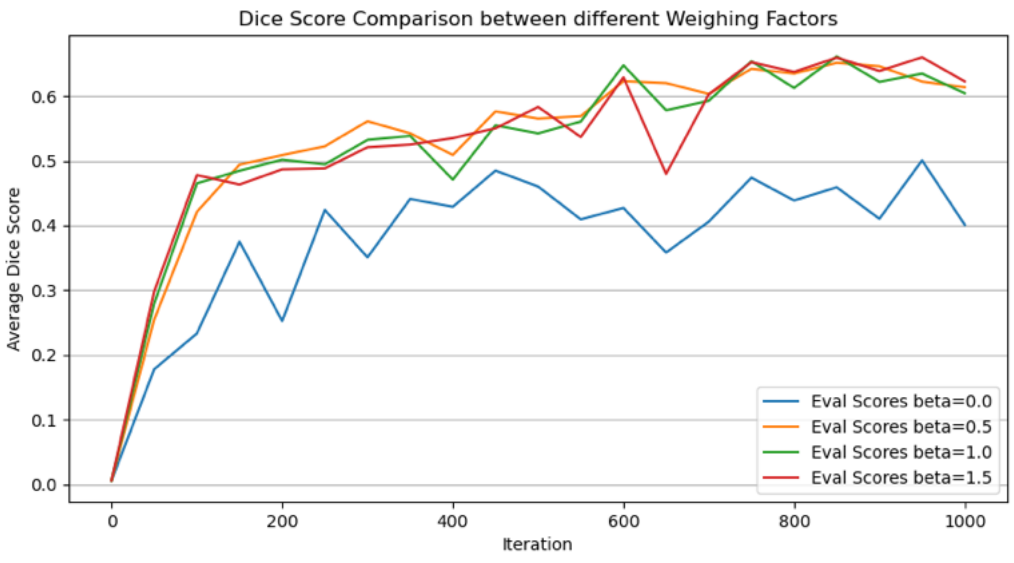


Figure 4.14: The effect of dice loss factors on the average dice score of the model over training

Fig. 4.14 shows the influence of including dice loss in the performance of a model during a training run by varying the magnitude of the prefactor β_1. The metric used for comparison is the average dice loss, further explained in Sect. 5.3. It can be seen that including the dice loss significantly improves the performance of the model. Furthermore, the influence of the prefactor β_1 is small. During subsequent training runs, the factor is therefore set to $\beta_1 = 1$.

Lastly, the pose loss is defined, as seen in the following equation:

$$\mathcal{L}_{pose} = \sum_i \gamma_1 \, \|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_2 + \gamma_2 \, \|\boldsymbol{q}_i - \frac{\hat{\boldsymbol{q}}_i}{\|\hat{\boldsymbol{q}}_i\|}\|_2 \tag{4.8}$$

where, similar to Eq. 4.4, $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{q}}_i$ describe the predicted linear and angular quantity, respectively. The ground truth values are shown without the accent, and $\|\cdot\|_2$ indicates the L2-norm. Here, both linear and angular values are calculated separately and summed to resolve issues in scale between the two measures. The origin of this loss function lies in the PoseNet loss [56], where the absolute location and angles of the camera are predicted. In our scenario, multiple combinations between absolute locations, linear velocities, as well as different angular representations, such as Euler angles and quaternions, are experimented with.

Fig. 4.15 shows an exemplary loss curve during training for binary cross-entropy $\mathcal{L}_{ce}$. A fast decrease is observable; the loss decreases consistently until the end of training at 1500 iterations. The dice loss $\mathcal{L}_{dice}$ similarly decreases over the training cycle. The variation in loss magnitude over the training cycle is big; however, earlier experiments, as shown in Fig. 4.14, confirm the utility in addition to using the dice loss.

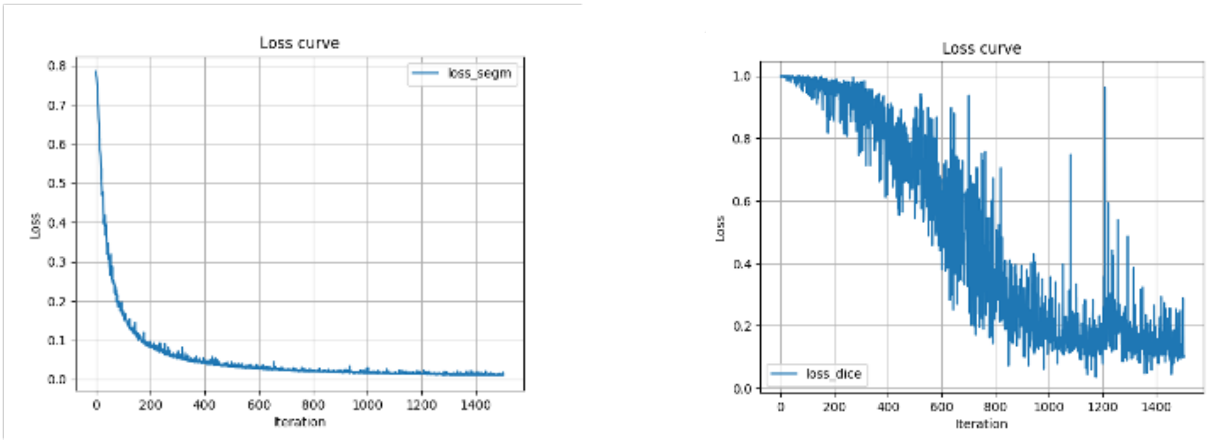


Figure 4.15: Loss Curve of the binary cross-entropy and dice losses.

Fig. 4.16 shows the evolution of the adapted PoseNet loss $\mathcal{L}_{pose}$ using only the angular velocity values over 1500 iterations. A sudden increase of the loss can be observed, until it steadily decreases. To better illustrate the scale, the figure is shown with a logarithmic scale. The error remains steady after around 200 iterations, at a value of 10^{-2}.

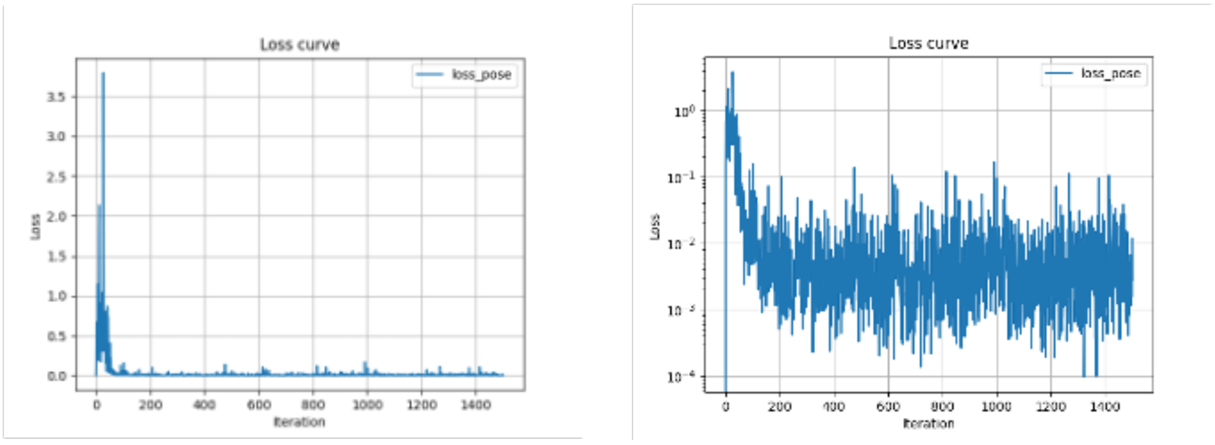


Figure 4.16: Loss Curve of the adapted PoseNet loss: Linear Scale (left), Logarithmic scale (right)

Different camera pose regression models are compared in Fig. 4.17. During an exemplary training run, every 50 iterations, the average dice score was calculated based on image input from the test set. First, parameters that are relevant to explaining movement in-between frames were used in the loss function. These include the height, rotational angles, and angular and linear velocities of the camera. Since the movement of segmentation maps in the output more directly corresponds to changes in rotation and location, the second model in green only includes linear and angular velocities in the pose loss function. Finally, only angular velocities were included. Roll, pitch, and yaw velocities have a big influence on the location of the maritime vehicles in the image, especially at close distances. As can be seen in Fig. 4.17, only including this information aides the network the most in representing motion.

To show the improvement in angular prediction in the test set, evaluation runs were performed during training every 50 iterations, as shown in Fig. 4.18. In blue, the increase in segmentation quality is

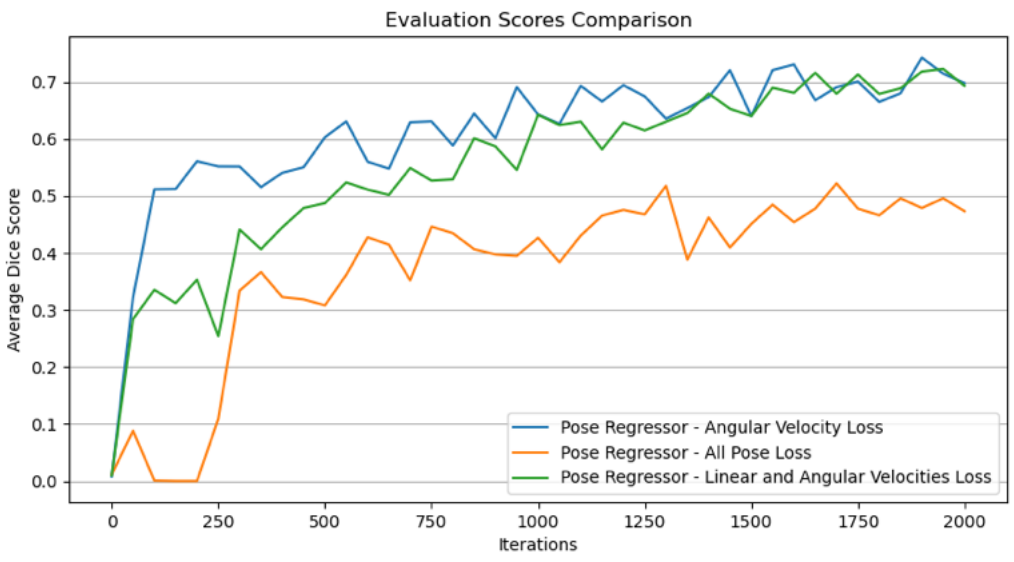


Figure 4.17: A comparison between different camera pose configurations during training. Including all investigated parameters seems to perform the worst, while only including angular velocity values is the best. The models are evaluated every 50 iterations on the test set.

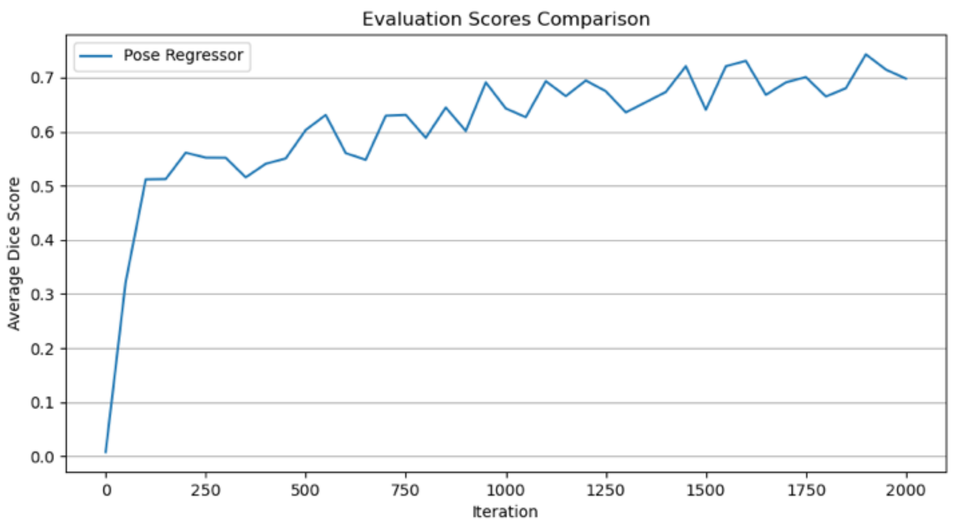


Figure 4.18: The evaluation of the dice score during training is shown above, for the camera pose regression model predicting angular velocities.

seen on the test set, as indicated by the increase in the average dice score when evaluating over 100 iterations test video samples. The difference in angular velocity is shown in Fig. 4.19 in yellow, where a clear learning effect is observed. After the specified iterations, errors below $0.1\ rad/s$ are seen. The next section discusses a different pretext task that aims at incorporating camera pose information differently.

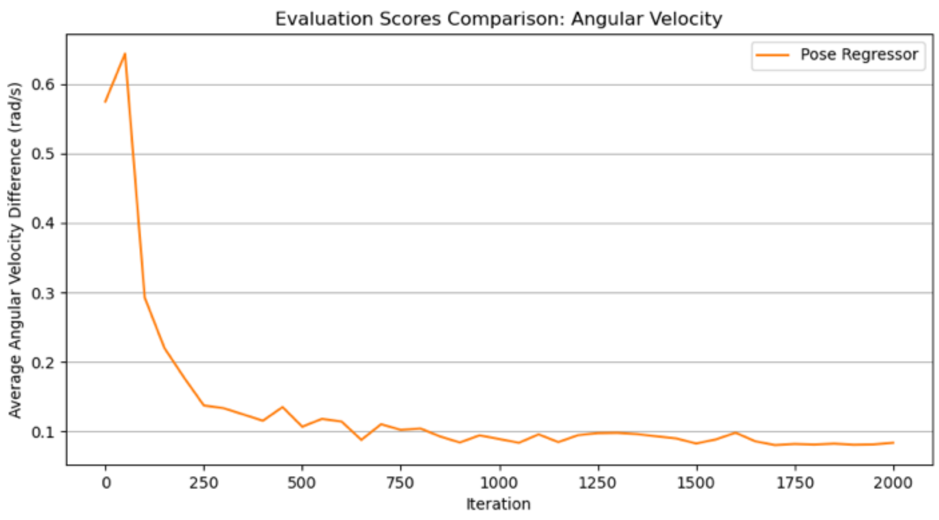


Figure 4.19: The average angular velocity difference is shown during training. The shown model uses the camera pose regression predicting angular velocities as an additional task.

4.5 Motion Flow as an Additional Task

Similarly to Sect. 4.4, the additional task is devised as a side branch of the U-Net, where the hidden representation z is used for the prediction of the motion field. It is theorized that the use of camera motion in the form of a motion field gives a more informationally rich representation for the model to learn from, due to the large amount of data contained in each field. Since directly regressing camera motion data from the compressed representation z did not seem to create a motion representation in the model, this alternative path is chosen. Fig. 4.20 shows the adaptation of the model for the prediction of motion flow.

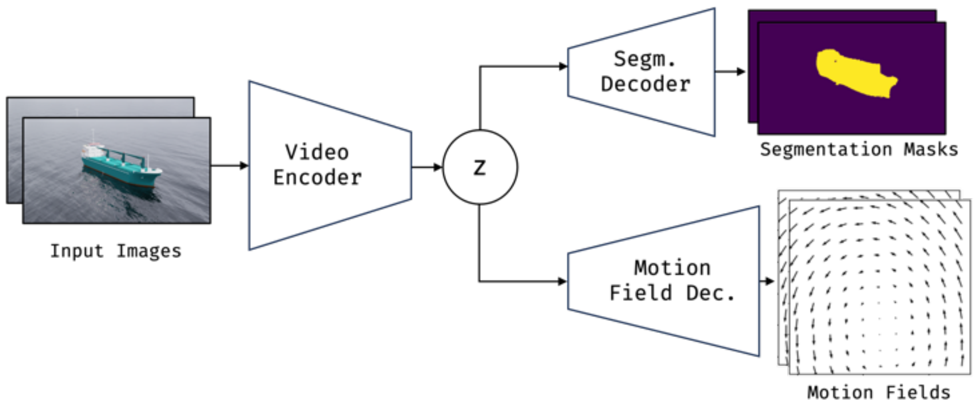


Figure 4.20: Simplified model of the motion flow prediction approach. A series of input images is transformed into a latent representation z using a video encoder. This representation z is then both used to generate a series of segmentation maps and motion fields, by the corresponding decoder blocks.

Ground truth information on the motion field is not available from the video data. By making geometric idealizations and using the camera motion information obtained in Sect. 4.1, ground-truth motion fields are created. Due to the negligible height of waves, as well as backgrounds which mainly consist of the horizon, the ocean is modeled as a plane at $z = 0$. The points in the plane, when seen from a mobile camera, create a motion field.

The motion flow embeds information about the movement of the camera. Given the points in the camera frame, their apparent movement due to the motion of the camera is to be investigated to obtain the motion flow of the image. The motion flow of the scene consists of the difference between the location of pixels in the time step t, compared to the subsequent time step $t+1$. Each pixel is first projected back into the water plane $z = 0$ using the inverse homography matrix $\boldsymbol{H}_t^{-1}$. After the movement of the camera between subsequent frames, the change in location and orientation of the camera changes the projection matrix $\boldsymbol{P}_{t+1}$, as well as the corresponding homography $\boldsymbol{H}_{t+1}$, defined in Eq. 4.9. Now each of the original pixels in the three-dimensional space is projected to the camera to obtain the final location of the pixels. Using the projection matrix described in Sect. 2.7, the plane points can be matched to pixels in the camera frame.

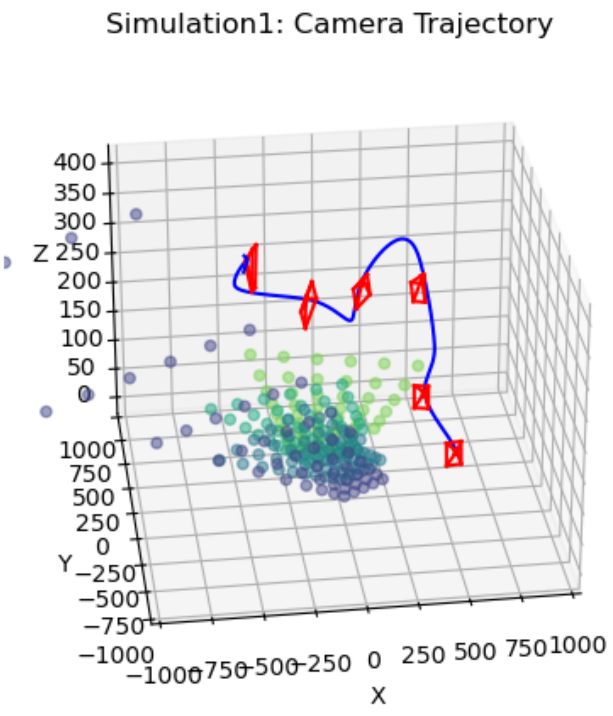


Figure 4.21: Backprojecting the camera pixels: The trajectory of the camera, along with sampled camera orientations are shown. The corresponding backprojection of camera frame pixels into three-dimensional space is indicated by the blue/green pixels. Pixels get progressively lighter, throughout the whole trajectory. The location of pixels in space is projected into the next state of the camera, allowing for the calculation of motion flow throughout a video.

The following equation describes the projection of camera pixels into three-dimensional space:

$$\lambda \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \boldsymbol{H}_t^{-1} \cdot \boldsymbol{x_c} = \begin{bmatrix} h_{11} & h_{12} & h_{14} \\ h_{21} & h_{22} & h_{24} \\ h_{31} & h_{32} & h_{34} \end{bmatrix}_t^{-1} \begin{bmatrix} u_t \\ v_t \\ 1 \end{bmatrix}, \tag{4.9}$$

where $\boldsymbol{H}_t$ describes the homography matrix at time t, considering the geometric constraint. The variable $\boldsymbol{x}_c$ describes the location of the pixel in the camera frame. The parameters h_{ij} describe the projection matrix parameters. Due to the geometric constraint of lying on the ground plane, the third column of the projection matrix is be removed to create the homography matrix $\boldsymbol{H}_t$; all points lie on $z_t = 0$. To obtain

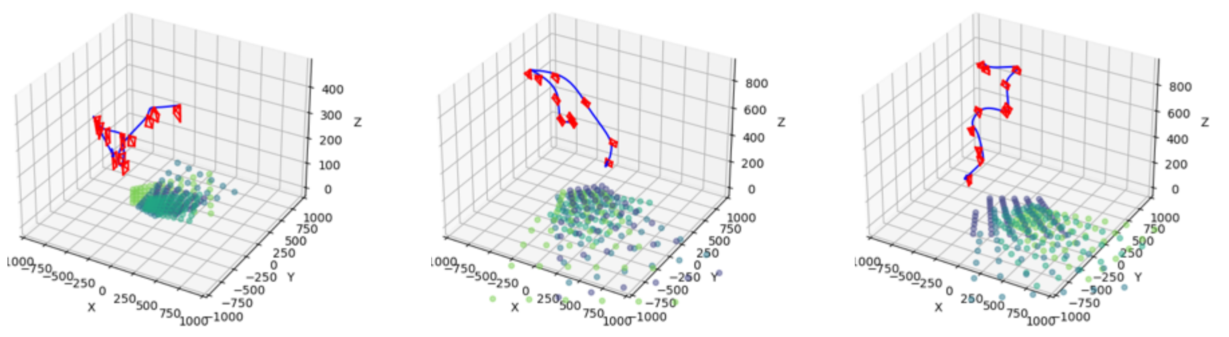


Figure 4.22: The trajectories of videos 10, 12, and 15 are shown above. In addition, the backprojection of pixels from the camera view onto the ocean plane is indicated by the blue/green dots. Dark pixels indicate the backprojection towards the beginning of the trajectory, whereas lighter colors indicate a later projection.

the location of points back in the camera frame, the projection is applied in the next timestep.

$$\lambda \begin{bmatrix} u_{t+1} \\ v_{t+1} \\ 1 \end{bmatrix} = \boldsymbol{P}_{t+1} \begin{bmatrix} x_t \\ y_t \\ z_t = 0 \\ 1 \end{bmatrix} , \tag{4.10}$$

The motion flow is finally obtained as follows,

$$\begin{bmatrix} u_{motion,t} \\ v_{motion,t} \end{bmatrix} = \begin{bmatrix} u_{t+1} \\ v_{t+1} \end{bmatrix} - \begin{bmatrix} u_t \\ v_t \end{bmatrix} , \tag{4.11}$$

where the same points in space are projected differently from the camera frame, due to the motion of the camera between time t and $t+1$. The difference in location results in the motion flow.

The upper procedure thus implicitly encodes a number of parameters of the camera movement. In particular, each projection matrix is calculated using the location and angular orientation of the camera. Due to the change in time between the backprojection and reprojection, the linear and angular velocities are implicitly encoded in the result.

Fig. 4.21 shows a visualization of the process. The trajectory and orientation of the camera are shown for the third video in the dataset. Each point of the current frame is projected into three-dimensional space, where the backprojected points get increasingly more light throughout the sequence. In timestep $t+1$, the location of these points in the new camera frame is recorded to calculate the optical flow. Note that the projections are subsampled, to obtain a clear visualization of the process; To obtain the motion field data, this process is executed for each time pixel and step t. Several other trajectories with exemplary backprojections are shown in Fig. 4.22.

Fig. 4.23 shows an example of the motion flow obtained. A rotating motion of the camera can be clearly observed. As a confirmation, the ground truth input images are shown on the side. To improve the visualization of the motion, the second image was chosen to be approximately one second after the initial one.

For the prediction of motion fields, several loss functions may be used. During this thesis, the focus was on the L2 error. Geometrically, it can be interpreted as the square of differences between pixel velocities

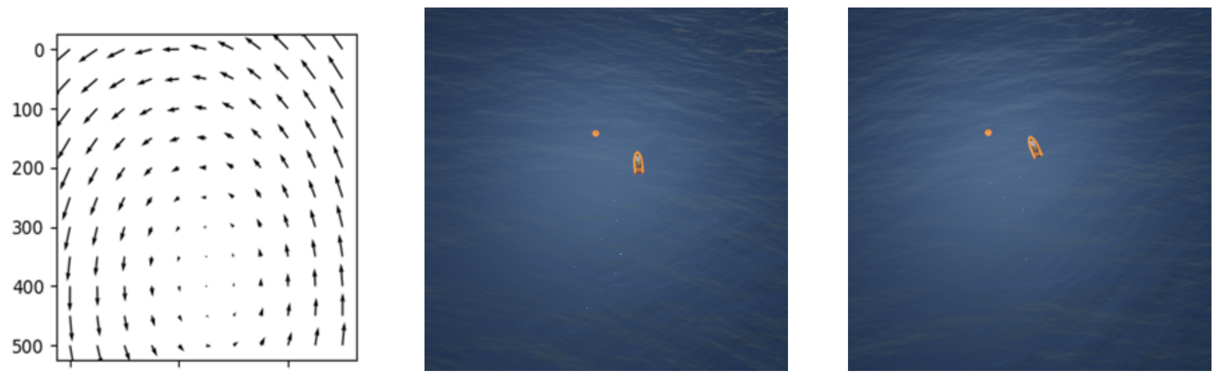


Figure 4.23: Example of Motion Flow: The first image shows the difference between pixels at time $t-1$ and t for video 3, frame 591. The next two images show the movement of the camera; note that for visibility, frames 591 and 620 are shown.

and can be expressed as follows:

$$\mathcal{L}_{motion\_flow} = \|\boldsymbol{V}_{gt} - \hat{\boldsymbol{V}}\|_2 \ , \tag{4.12}$$

where V_{gt} and $\hat{V}$ refer to the ground truth and estimated motion fields, respectively, and $\|\cdot\|_2$ is the L2 norm. Specifically, the dimension of the motion field vectors here is $(2, T, H, W)$, referring to the time, height, and width dimensions of the image. The horizontal and vertical components of the flow vector are encoded in the first dimension of the motion field vector.

The decoder module generates motion fields using the latent representation z as input. The choice of decoder has a large influence on the quality of the output. During this thesis two approaches were attempted: First, we reuse the video segmentation branch already provided through the U-Net implementation. To this end a copy of the deconvolutional branch, defined in Tab. **??** is made, while replacing the final output dimensions with $d_{out} = 2$, to calculate (u_{motion}, v_{motion}). Given the existing implementation and due to the combination between high-level representations and skip-connections, along with the locality of the motion field data, it is reasonable to attempt.

Second, an existing optical flow CNN, FlowNet, was used [58]. It consists of a contracting and expending part that predicts the optical flow between two images. In this context, FlowNet could be used in two different ways. The first approach applies it directly to the latent representation z and scales the results to the input image dimensions. The low rank of the desired output, given the comparatively low number of parameters needed to generate a motion field, made the upscaling plausible.

A representative loss curve from training for the use of the 3D U-Net is shown below in Fig. 4.24. The model cannot predict the motion flow of the image series used as input. The loss is not decreasing and seems to show large variations in some training iterations. Fig. 4.25 shows an example of the prediction. The first two of the five predictions are depicted below, while the ground truth is shown on top. The rotating motion of the camera is clearly seen; however, the model is unable to predict this result.

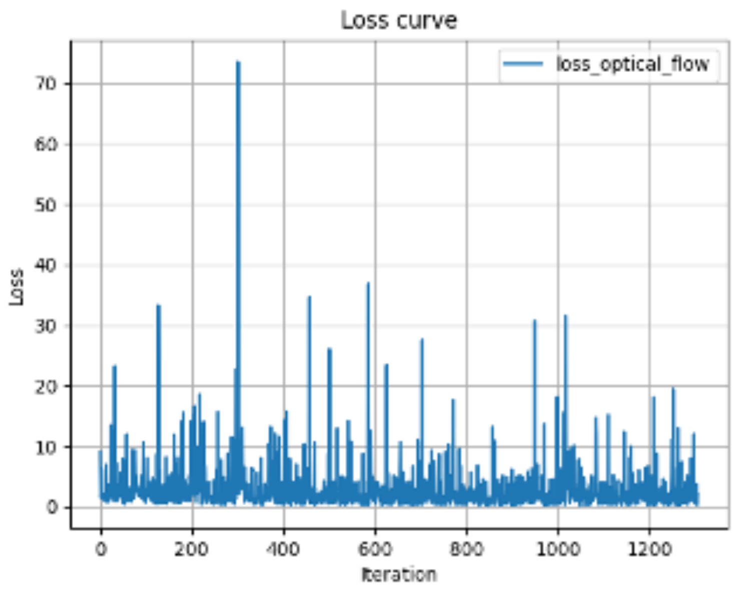


Figure 4.24: Loss curve during motion field prediction: The loss has a large variance, while not decreasing throughout the training process, as seen on the left.

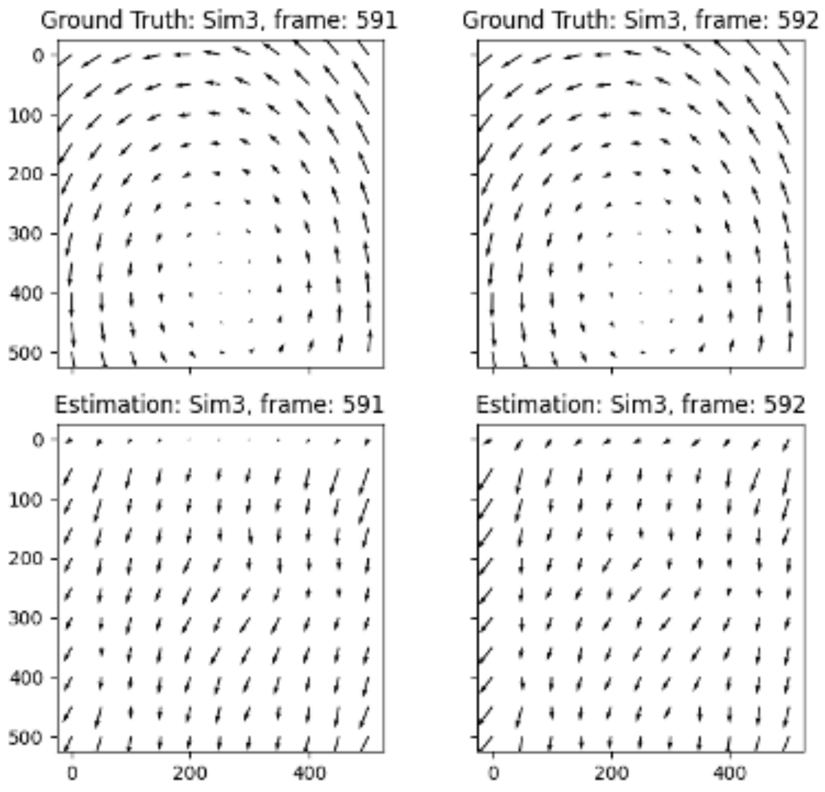


Figure 4.25: An example of the prediction, with the corresponding ground truth, is shown to the right.

Chapter 5

Results and Evaluation

This chapter gives an overview of exemplary outputs of the 3D U-Net and Mask2Former models, including the adaptations described in this work. A series of evaluation runs during training is presented as well as a comparison between the models that were trained in previous sections. Specifically, the addition to the Mask2Former model, as well as the extension of the 3D U-Net with the additional task of pose regression, and motion flow prediction are shown. Finally, a statistical evaluation based on Bayesian t-testing is performed for the most promising model, namely the 3D U-Net that predicts the angular velocity as an additional task.

5.1 Experimental design

The hyperparameters used for training the Mask2Former and 3D-U-Net models are seen in Table 4.2 and Table 4.4 respectively. Similarly, the structure used for the 3D-U-Net is shown in Table **??**.

The models are first trained on a subset of available data, the training set, comprising 80% of the videos, and then evaluated on the test set, consisting of the remaining 20%. The following plots show the model evaluations in one specific train/test split. Specifically, the following split was tested: The videos chosen for the train set are $(1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25)$ and for the test set $(2, 13, 14)$. Due to missing simulation data, video 20 could not be used for the extraction of camera movement and was therefore excluded from training.

To illustrate the change in model performance over training, evaluation runs are performed on the test set for every fixed number of iterations, here chosen to be $f_{eval} = 50$, that is, during the maximum iterations of $n_{max} = 2000$, 40 evaluation runs are performed. Each evaluation run consists of predicting segmentation masks for a number $n_{eval} = 100$ of video volumes, and then obtaining the dice score with respect to the ground truth masks. Averaging these dice scores gives a metric for the prediction performance of the model on the test set.

5.2 Results

Fig. 5.1 shows an example of the output of the 3D U-Net model, trained with the added pose regressor predicting angular velocity values. On the left-hand side, the logits of the model are seen. This example shows clear vessel-shaped activations, while the rest of the image is deactivated. To generate segmentation masks for the model estimation, a sigmoid function is applied, to force the output to a range between $(0.0, 1.0)$. To obtain the final masks, a threshold of $t = 0.5$ is applied. The third image shows the first input image of the sequence, depicting two orange boats. The ground truth segmentation mask is shown to the right.

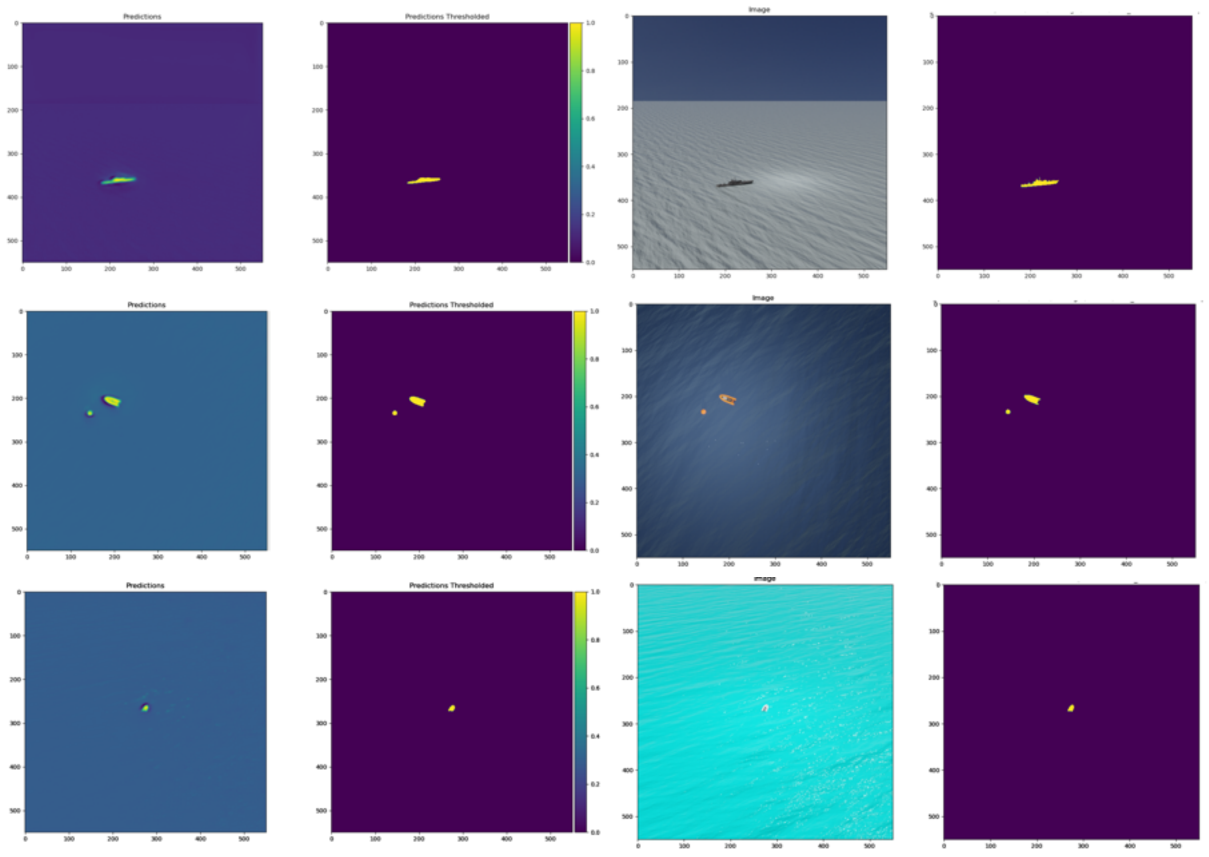


Figure 5.1: Results showing the prediction and ground truth for the first image of a prediction using the pose model trained with angular velocities. From left to right: Logits, predicted segmentation mask, input image, and ground truth.

An exemplary output of the Mask2Former model with the additional task of pose regression for angular velocities is seen in Fig. 5.2. The segmentation mask, shown in pink, is displayed on top of the boat. The label shows a classification score of 100% since we only accept boats as a category for the classification.

5.3 Evaluation

5.3.1 Mask2Former Evaluation

Fig. 5.3 shows the comparison between the Mask2Former model with and without the additional pose loss function and without pretraining. Their difference in performance during the training run, as seen in Fig. 5.3, does not appear to be significant. A further evaluation is performed using the AP scores, to judge whether the modified version outperforms the original Mask2Former.

Evaluating the model in terms of AP scores reveals the scores shown in Tab.5.1. The original transformer model still outperforms the extended model.

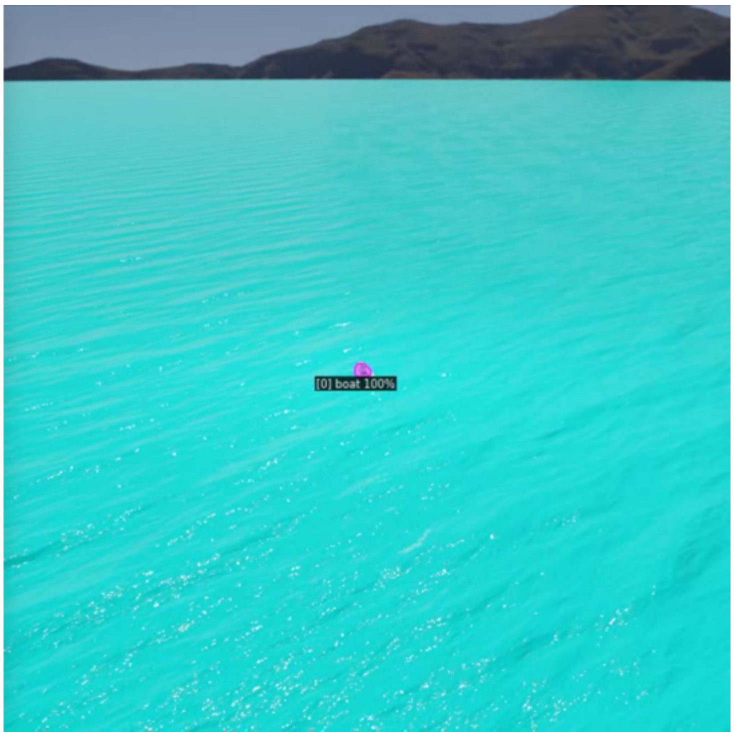


Figure 5.2: An image depicting the results from the video segmentation of the Mask2Former with the additional task of camera pose regression of angular velocity values.

	Mask2Former	**Mask2Former + Pose**
AP Score	41.68%	35.55 %

Table 5.1: No Pose: Cross-validation results for cross-validation runs and five folds

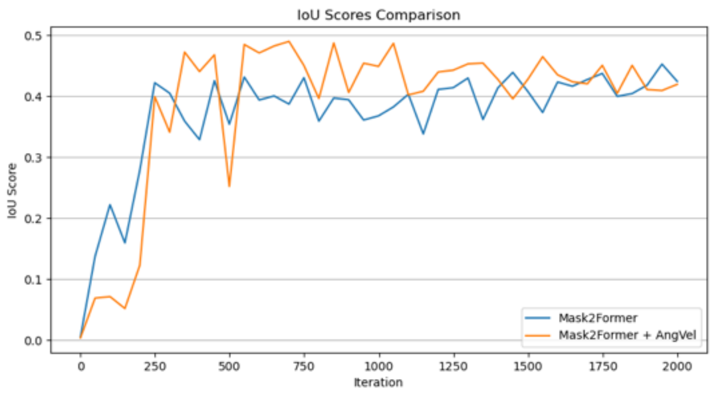


Figure 5.3: Comparison between the original Mask2Former and the modified one for the first 2000 training iterations for semantic segmentation. The difference in evaluation accuracy is negligible. The IoU scores are evaluated every 50 iterations on the test set.

5.3.2 Comparison between the 3D U-Net and the Mask2Former

For comparison, Fig. 5.4 shows the performance of a three-dimensional U-Net model. It can be seen that the performance of the U-Net increases faster. This is believed to be due to the comparably low parameter count, which makes it easier for the model to learn this dataset. It is expected that, given enough training data, the transformer model will overtake the U-Net. Details on the implementation of

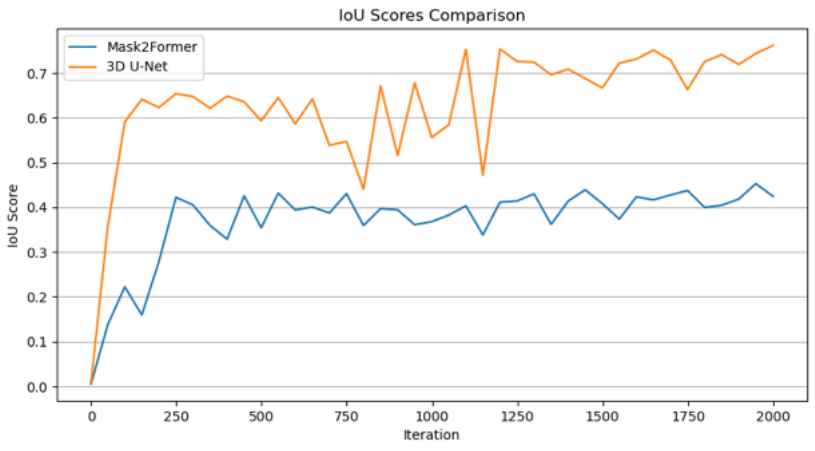


Figure 5.4: Comparison between the original Mask2Former and a 3D-U-Net. The IoU scores are evaluated every 50 iterations on the test set.

the 3D U-Net are given in Sect. 4.3.

The Mask2Former thus reaches limits with regard to training data and the amount of computational resources required for training and inference of the Mask2Former model. Furthermore, given the scenarios for maritime surveillance outlined in Sect. 1, inference on aerial vehicles becomes difficult, given the need for large transformer model sizes. The rest of this thesis thus explores the possibility of using camera movement data in encoder-decoder models, such as the three-dimensional U-Net. To this end, a multitask learning approach is used.

5.3.3 3D U-Net Models Evaluation and Comparison - Pose regression and Original Models

The following figures show comparisons between the trained 3D U-Net models.

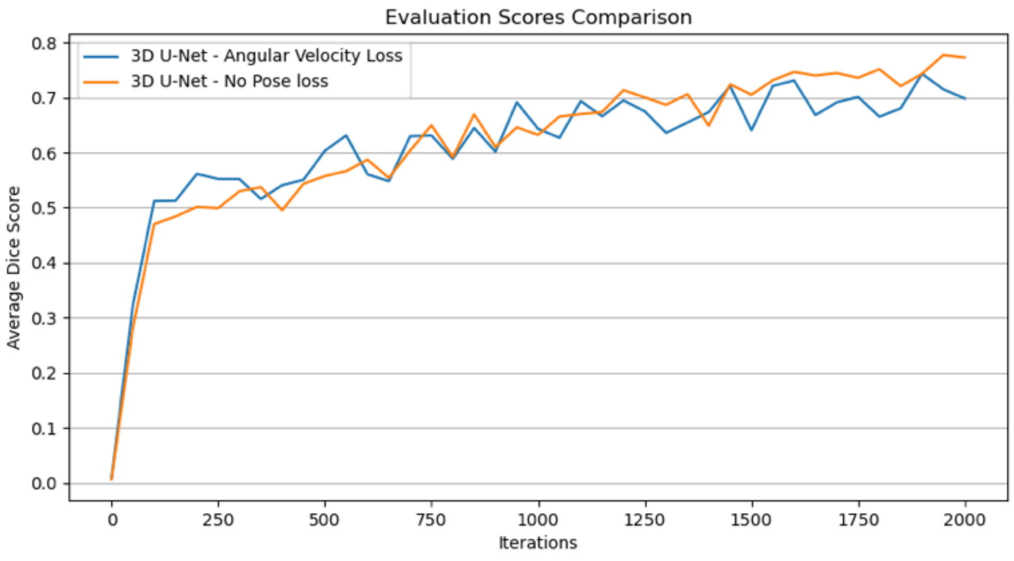


Figure 5.5: Comparison between the unmodified 3D U-Net model and the model including pose regression during training.

Fig. 5.5 shows a comparison between the original 3D U-Net model and the model that includes the pose loss based on the angular velocities. Again, evaluation runs are performed every 50 iterations during the training of either model. The accuracy of both models seems to be similar; however, towards the end of the training run, the unmodified 3D U-Net shows a slight increase in accuracy. This suggests that including a prediction of the camera pose as a simple regression task on the compressed representation does not improve the model performance for these particular training data.

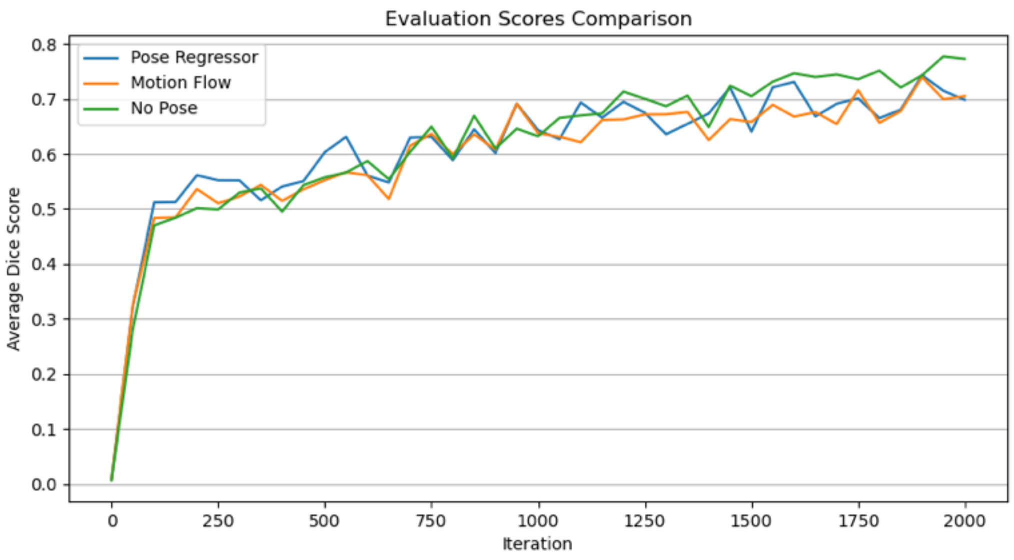


Figure 5.6: Evaluation runs over training for different models.

Fig. 5.6 shows a comparison over one training run between all 3D U-Net-based models that were investigated. As can be seen, the addition of movement losses in general does not have a major impact on the performance of the models. Slight increases and decreases in accuracy regarding the No-Pose model can be seen at the beginning and end of training, respectively.

5.3.4 Bayesian t-Testing

When comparing the accuracy of neural networks on video segmentation tasks, an accurate evaluation is important. In the deep learning paradigm, the performance of an approach is heavily dependent on training and test data. For this purpose, the MarSyn videos were randomly divided into an 80/20 train test split. Due to the arbitrary nature of splitting data like this, k-fold cross-validation is used to obtain a more statistically significant result. In $m = 4$ iterations, the input videos were divided into $k = 5$ different groups, while keeping the same random seed. Each of these groups was once used for evaluation while training with the rest. This split alleviates concerns of arbitrariness when working with one dataset.

The results of the k-fold cross-validation are shown in the following tables. Tab. 5.2 and Tab. 5.3 show the average dice scores obtained over subsequent runs of 100 video fragments on the relevant test set, for the pose regression and non-pose regression model respectively. When comparing averages, the non-pose model seems to perform slightly better. A detailed statistical test is shown below.

As shown in Sect. 2.9.5, correlated Bayesian t-tests are useful to compare the performance of different classifiers. They actively give probabilities of one method surpassing the other. The following results were obtained by evaluating the ensemble of run and fold accuracies obtained above;

Run	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Run 1	0.665	0.809	0.722	0.708	0.770	0.735
Run 2	0.815	0.736	0.691	0.742	0.780	0.753
Run 3	0.669	0.723	0.789	0.706	0.777	0.733
Run 4	0.772	0.768	0.671	0.702	0.674	0.717
Overall Average						0.734

Table 5.2: Pose: Cross-validation results for four cross-validation runs and five folds

Run	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Run 1	0.689	0.792	0.721	0.699	0.785	0.737
Run 2	0.777	0.818	0.693	0.718	0.783	0.758
Run 3	0.661	0.706	0.774	0.744	0.778	0.733
Run 4	0.767	0.773	0.680	0.716	0.683	0.724
Overall Average						0.738

Table 5.3: No Pose: Cross-validation results for cross-validation runs and five folds

$$\begin{aligned} P(Pose > NoPose) &= 0.17 \;, \\ P(rope) &= 0.51 \;, \\ P(NoPose > Pose) &= 0.32 \;, \end{aligned} \tag{5.1}$$

where each metric represents the probability that the performance of the new classifier will improve, match, or worsen with respect to the old one. As can be seen from this analysis, the probability that both classifiers have the same performance is the highest. Additionally, it is unlikely that the model including camera pose outperforms the non-pose one.

Fig. 5.7 shows the process graphically. The probability density function shows the probability of a difference between the original and the pose model. The probabilities obtained in Eq. 5.1 can be graphically interpreted as an integral over the appropriate sections of the probability density function.

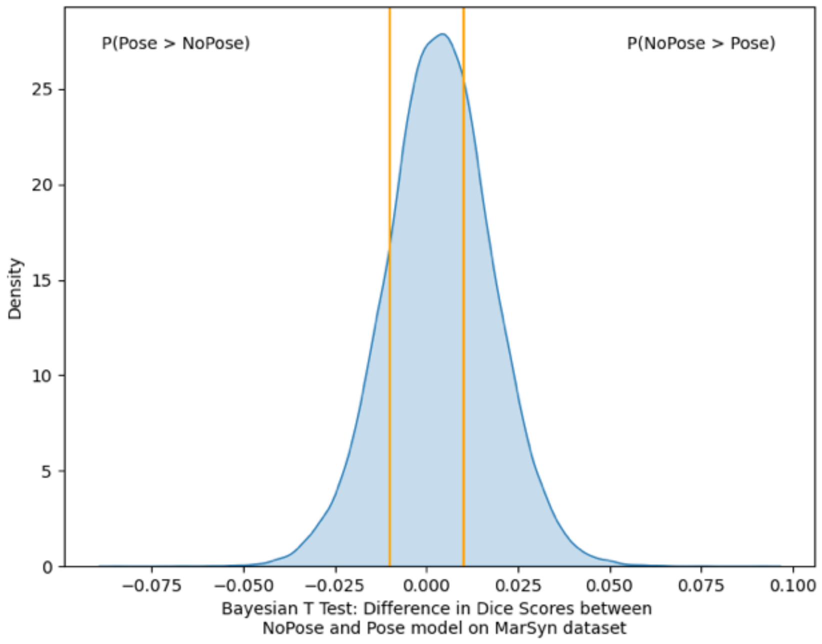


Figure 5.7: Bayesian T Test comparing the average dice scores between the 3D-Unet with and without the added pose loss. The yellow bars indicate the region of practical equivalence. A probability density function of the difference between the NoPose and Pose model is shown. The probability of an improvement is low, as indicated by the the small area on the left of the orange vertical line.

Chapter 6

Conclusion and Future Work

The following chapter gives a conclusion to the thesis work. Each approach is first summarized and then discussed in brief. Finally, possible avenues for future work are discussed.

6.1 Conclusion

This work explored different approaches to using camera pose information to improve video segmentation performance. Information on camera movement, in combination with high-quality segmentation masks, is difficult to obtain, especially for specific scenarios, such as maritime surveillance. Simulation data from a synthetic dataset, MarSyn, is therefore first extracted, to create ground-truth values for reference. Then, several approaches are tested to make neural networks predict these values, to improve segmentation performance.

The first approach is described in Sect. 4.2, where a transformer-based model, the Mask2Former, was explored. A branch was added to predict camera pose values to create internal representations of camera motion, which might help improve the segmentation of maritime videos. The analysis shows a decrease in performance compared to the original model. This is due to the inability of the model to accurately predict camera motion using this branch. Due to the size of computational and data resources necessary for transformer-based models to show good performance, smaller neural networks are investigated.

Sect. 4.3 then shows the extension of a 3D U-Net with a camera pose regression decoder. The compressed representation z created by the encoder-decoder architecture is fed into an FFN, to directly predict camera pose parameters. Through the prediction of the camera movement solely from the representation z, the model seeks an internal structure for movement. This additional information is theorized to improve the quality of video segmentation. After testing the inclusion of several groups of parameters, using only the values for angular velocity seems to be the most promising approach. However, statistical tests, based on Bayesian T testing, reveal that it is unlikely to improve over the unmodified model.

Lastly, in Sect. 4.5, the prediction of the motion flow is used as a pretext task to include camera movement in the 3D U-Net. Since the motion of the camera is inherently linked with the flow of pixels in a scene, predicting these might also create internal representations for movement. Idealizations about the scene were made, to obtain ground truth values for prediction, after which a second decoder of the 3D U-Net started predicting the motion flow. During experiments on the MarSyn dataset, the model was unable to accurately determine the motion flow given a series of subsequent images.

The inability of several different types of network to predict camera motion suggests the following possible findings. First, it is possible that the MarSyn dataset is not suitable to predict motion between frames under the assumptions taken. The texture originating from waves in the ocean is a recurring phenomenon that changes over time. Given that the majority of the image frame is composed of waves and water, it is possible that deriving movement information is further made difficult.

6.2 Future Work

Possibilities for future work include the comparison of including camera pose-based losses for different forms of neural networks. Since neural networks implicitly represent information in different ways, contrasting the inclusion of camera movement representations might help to gain more insight into the effectiveness of this approach. For example, long short-term memory (LSTMs) architectures may be adapted to predict segmentation maps for video data. Similarly, obtaining real-life data from missions using aerial vehicles presents an avenue for adapting these systems to realistic use cases. Although the collection of these data is time-consuming, the benefits of obtaining information in these scenarios will definitely improve model accuracy. This would ensure that the data used for model training closely corresponds to the real-life use case. For example, noise-related problems could be taken into account during the training of the new model.

6.3 Resources

The data and code used in the experiments for this thesis are openly available to promote transparency and reproducibility. The code used is available at the following links:

For obtaining the simulation files, as well as the video data, along with the ground truth, the following address is useful:

`https://vislab.isr.ist.utl.pt/marsyn-dataset/`

The modified version of the Mask2Former is available at the following address:

`https://github.com/majvie/video_segmentation`

The data processing to obtain camera movement data, as well as the modified versions of the 3D U-Nets are available in the following repository:

`https://github.com/majvie/ship_segmentation`

For setup and usage instructions, refer to the README files in the repositories. Contact the author at maximilian.vieweg@gmail.com for further assistance.

Bibliography

[1] R. Ribeiro, G. Cruz, J. Matos, and A. Bernardino, “A Data Set for Airborne Maritime Surveillance Environments,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, pp. 2720–2732, Sept. 2019.

[2] M. Ribeiro, B. Damas, and A. Bernardino, “Real-Time Ship Segmentation in Maritime Surveillance Videos Using Automatically Annotated Synthetic Datasets,” *Sensors*, vol. 22, no. 21, 2022.

[3] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 6999–7019, Dec. 2022.

[4] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), Lecture Notes in Computer Science, (Cham), pp. 234–241, Springer International Publishing, 2015.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[6] Y. Wang, Z. Xu, X. Wang, C. Shen, B. Cheng, H. Shen, and H. Xia, “End-to-End Video Instance Segmentation with Transformers,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Nashville, TN, USA), IEEE, June 2021.

[7] R. Szeliski, *Computer Vision: Algorithms and Applications*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.

[8] D. Rao, P. K, R. Singh, and V. J, “Automated segmentation of the larynx on computed tomography images: a review,” *Biomedical Engineering Letters*, vol. 12, pp. 1–9, Mar. 2022.

[9] X. Chen, X. Wu, D. K. Prasad, B. Wu, O. Postolache, and Y. Yang, “Pixel-Wise Ship Identification From Maritime Images via a Semantic Segmentation Model,” *IEEE Sensors Journal*, vol. 22, pp. 18180–18191, Sept. 2022.

[10] N. Tsolakis, D. Zissis, S. Papaefthimiou, and N. Korfiatis, “Towards AI driven environmental sustainability: an application of automated logistics in container port terminals,” *International Journal of Production Research*, vol. 60, pp. 4508–4528, July 2022. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207543.2021.1914355.

[11] A. Sandkamp, V. Stamer, and S. Yang, “Where has the rum gone? The impact of maritime piracy on trade and transport,” *Review of World Economics*, vol. 158, pp. 751–778, Aug. 2022.

[12] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, “Video Processing From Electro-Optical Sensors for Object Detection and Tracking in a Maritime Environment: A Survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 1993–2016, 2017.

[13] G. Cruz and A. Bernardino, “Learning Temporal Features for Detection on Maritime Airborne Video Sequences Using Convolutional LSTM,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, pp. 6565–6576, Sept. 2019.

[14] M. M. Marques, P. Dias, N. P. Santos, V. Lobo, R. Batista, D. Salgueiro, A. Aguiar, M. Costa, J. E. Da Silva, A. S. Ferreira, J. Sousa, M. De Fatima Nunes, E. Pereira, J. Morgado, R. Ribeiro, J. S. Marques, A. Bernardino, M. Grine, and M. Taiana, “Unmanned aircraft systems in maritime operations: Challenges addressed in the scope of the SEAGULL project,” in *OCEANS 2015 - Genova*, (Genova, Italy), pp. 1–6, IEEE, May 2015.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[16] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image Segmentation Using Deep Learning: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 3523–3542, July 2022. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[17] F. Milletari, N. Navab, and S.-A. Ahmadi, “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation,” in *2016 Fourth International Conference on 3D Vision (3DV)*, (Stanford, California, USA), pp. 565–571, 2016.

[18] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[19] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in Vision: A Survey,” *ACM Comput. Surv.*, vol. 54, Sept. 2022. New York, NY, USA.

[20] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep Learning for Generic Object Detection: A Survey,” *International Journal of Computer Vision*, vol. 128, pp. 261–318, Feb. 2020.

[21] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” (San Francisco, CA, USA), pp. 580–587, 2014.

[23] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, Dec. 2015. ISSN: 2380-7504.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” (San Juan, PR, USA), pp. 779–788, 2016.

[25] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, “A Review of Yolo Algorithm Developments,” *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022.

[26] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, “YOLACT++ Better Real-Time Instance Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 1108–1121, Feb. 2022.

[27] T. Zhou, F. Porikli, D. J. Crandall, L. Van Gool, and W. Wang, “A Survey on Deep Learning Technique for Video Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, pp. 7099–7122, June 2023.

[28] J. Wu, Q. Liu, Y. Jiang, S. Bai, A. Yuille, and X. Bai, “In Defense of Online Models for Video Instance Segmentation,” in *Computer Vision – ECCV 2022* (S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds.), (Cham, Switzerland), pp. 588–605, Springer Nature Switzerland, 2022.

[29] R. Caruana, “Multitask Learning,” *Machine Learning*, vol. 28, pp. 41–75, July 1997.

[30] B. Horn, *Robot vision*. The MIT electrical engineering and computer science series, Cambridge, Mass. : New York: MIT Press ; McGraw-Hill, mit press ed ed., 1986.

[31] F. Van Beers, A. Lindström, E. Okafor, and M. Wiering, “Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation:,” in *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, (Prague, Czech Republic), pp. 438–445, SCITEPRESS - Science and Technology Publications, 2019.

[32] R. Zhao, B. Qian, X. Zhang, Y. Li, R. Wei, Y. Liu, and Y. Pan, “Rethinking Dice Loss for Medical Image Segmentation,” in *2020 IEEE International Conference on Data Mining (ICDM)*, (Sorrento, Italy), pp. 851–860, IEEE, Nov. 2020.

[33] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, “Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis,” *Journal of Machine Learning Research*, July 2017.

[34] G. Corani and A. Benavoli, “A Bayesian approach for comparing cross-validated algorithms on multiple data sets,” *Machine Learning*, vol. 100, pp. 285–304, Sept. 2015.

[35] L. Yang, Y. Fan, and N. Xu, “Video Instance Segmentation,” (Seoul, Korea), pp. 5188–5197, 2019. Proceedings of the IEEE/CVF International Conference on Computer Vision.

[36] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, “Masked-attention Mask Transformer for Universal Image Segmentation,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (New Orleans, LA, USA), pp. 1280–1289, IEEE, June 2022.

[37] L. Ke, M. Danelljan, H. Ding, Y.-W. Tai, C.-K. Tang, and F. Yu, “Mask-Free Video Instance Segmentation,” pp. 22857–22866, 2023. Vancouver, BC, Canada.

[38] J. Wu, Y. Jiang, S. Bai, W. Zhang, and X. Bai, “SeqFormer: Sequential Transformer for Video Instance Segmentation,” in *Computer Vision – ECCV 2022* (S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds.), vol. 13688, pp. 553–569, Cham, Switzerland: Springer Nature Switzerland, 2022. Series Title: Lecture Notes in Computer Science.

[39] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask R-CNN,” (Cambridge, MA, USA), pp. 2961–2969, 2017. Cambridge, MA, USA.

[40] T. Albrecht, G. A. West, T. Tan, and T. Ly, “Visual Maritime Attention Using Multiple Low-Level Features and Naïve Bayes Classification,” in *2011 International Conference on Digital Image Computing: Techniques and Applications*, pp. 243–249, Dec. 2011.

[41] Z. Shao, L. Wang, Z. Wang, W. Du, and W. Wu, “Saliency-Aware Convolution Neural Network for Ship Detection in Surveillance Video,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, pp. 781–794, Mar. 2020.

[42] Y. Bazi and F. Melgani, “Convolutional SVM Networks for Object Detection in UAV Imagery,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, pp. 3107–3118, June 2018.

[43] C. Pires, B. Damas, and A. Bernardino, “An Efficient Cascaded Model for Ship Segmentation in Aerial Images,” *IEEE Access*, vol. 10, pp. 31942–31954, 2022.

[44] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[45] D. Pathak, R. Girshick, P. Dollar, T. Darrell, and B. Hariharan, “Learning Features by Watching Objects Move,” in *Venice, Italy*, pp. 2701–2710, 2017.

[46] P. Tokmakov, K. Alahari, and C. Schmid, “Learning Motion Patterns in Videos,” (Venice, Italy), pp. 3386–3394, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[47] P. Agrawal, J. Carreira, and J. Malik, “Learning to See by Moving,” (Santiago, Chile), pp. 37–45, Proceedings of the IEEE International Conference on Computer Vision, 2015.

[48] L. Jing, X. Yang, J. Liu, and Y. Tian, “Self-Supervised Spatiotemporal Feature Learning via Video Rotation Prediction,” Apr. 2019. arXiv:1811.11387 [cs].

[49] F.-E. Wang, H.-N. Hu, H.-T. Cheng, J.-T. Lin, S.-T. Yang, M.-L. Shih, H.-K. Chu, and M. Sun, “Self-supervised Learning of Depth and Camera Motion from 360 degree Videos,” in *Computer Vision – ACCV 2018* (C. Jawahar, H. Li, G. Mori, and K. Schindler, eds.), Lecture Notes in Computer Science, (Cham), pp. 53–68, Springer International Publishing, 2019.

[50] A. Faktor and M. Irani, “Video Segmentation by Non-Local Consensus Voting,” 2014.

[51] D. D. Bloisi, L. Iocchi, A. Pennisi, and L. Tombolini, “ARGOS-Venice Boat Classification,” in *2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, (Karlsruhe, Germany), pp. 1–6, IEEE, Aug. 2015.

[52] L. A. Varga, B. Kiefer, M. Messmer, and A. Zell, “SeaDronesSee: A Maritime Benchmark for Detecting Humans in Open Water,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, (Waikoloa, Hawaii), pp. 2260–2270, Jan. 2022.

[53] M. Jeff Faudi, “Airbus Ship Detection Challenge,” 2018.

[54] M. Kristan, V. Sulić Kenk, S. Kovačič, and J. Perš, “Fast Image-Based Obstacle Detection From Unmanned Surface Vehicles,” *IEEE Transactions on Cybernetics*, vol. 46, pp. 641–654, Mar. 2016.

[55] B. Bovcon, R. Mandeljc, J. Perš, and M. Kristan, “Stereo obstacle detection for unmanned surface vehicles by IMU-assisted semantic segmentation,” *Robotics and Autonomous Systems*, vol. 104, pp. 1–13, 2018.

[56] A. Kendall, M. Grimes, and R. Cipolla, “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization,” (Santiago, Chile), pp. 2938–2946, 2015.

[57] E. Gordon-Rodriguez, G. Loaiza-Ganem, G. Pleiss, and J. P. Cunningham, “Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning,” pp. 1–10, PMLR, Feb. 2020. ISSN: 2640-3498.

[58] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, 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)*, (Santiago), pp. 2758–2766, IEEE, Dec. 2015.

Appendix A

Appendix chapter

A.1 Camera Trajectories

The following figures show the camera trajectories collected as well as the camera orientation on each video of the dataset. Video 20 is omitted due to corrupted simulation data.

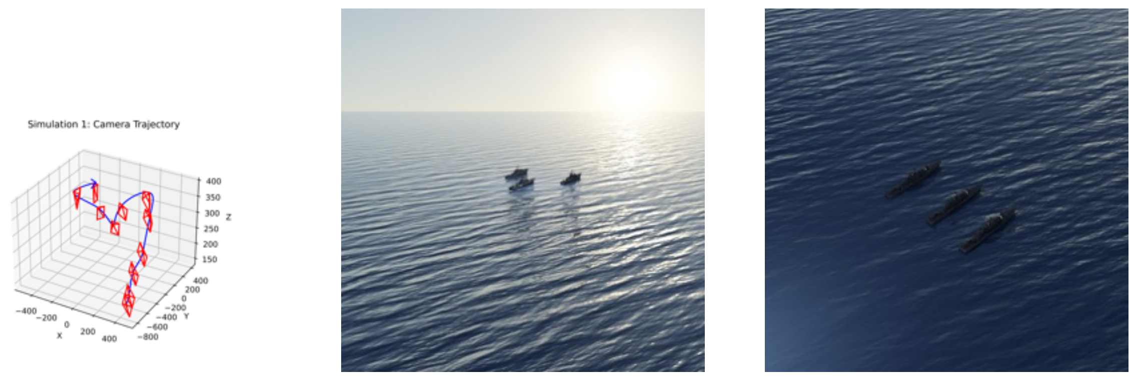


Figure A.1: The trajectory of the camera for video 1 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 1 are shown.

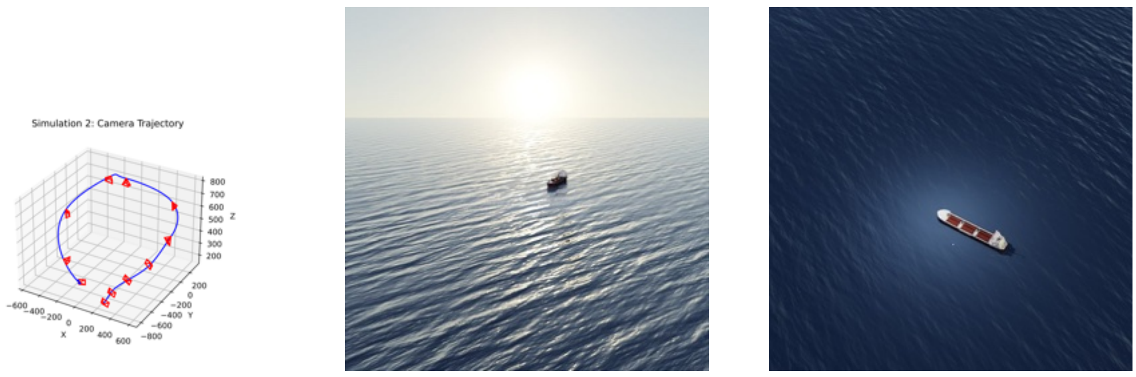


Figure A.2: The trajectory of the camera for video 2 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 2 are shown.

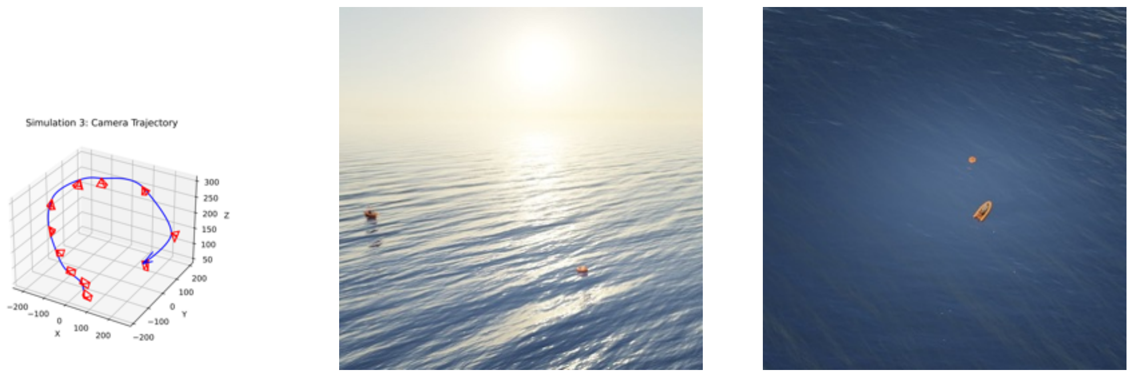


Figure A.3: The trajectory of the camera for video 3 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 3 are shown.

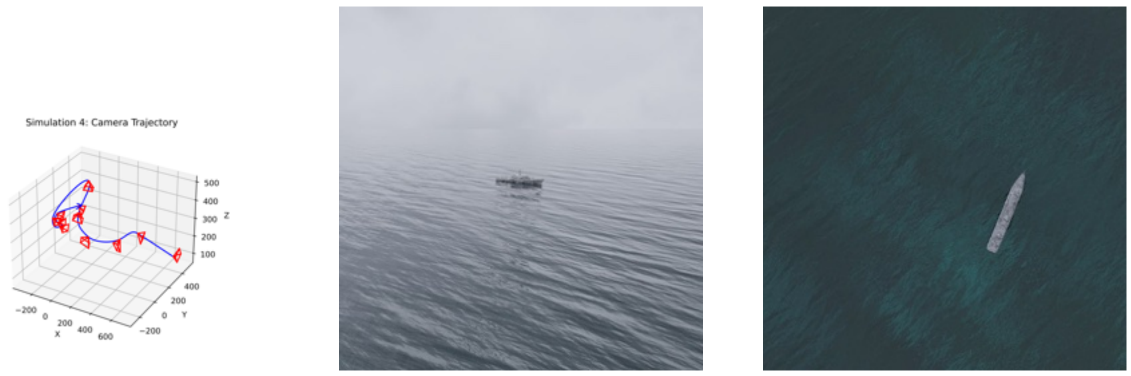


Figure A.4: The trajectory of the camera for video 4 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 4 are shown.

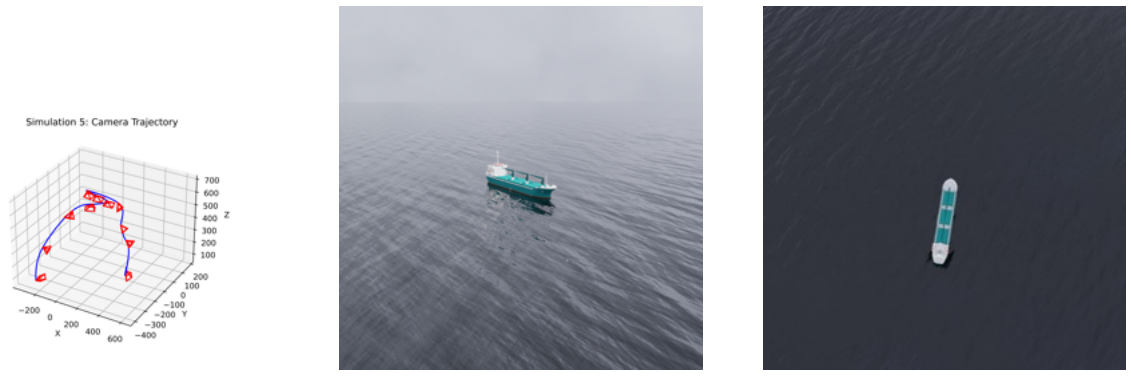


Figure A.5: The trajectory of the camera for video 5 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 5 are shown.

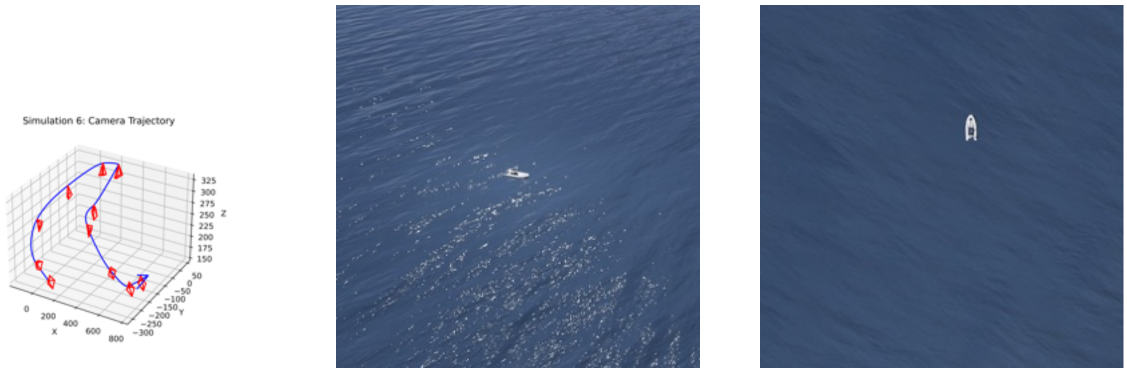


Figure A.6: The trajectory of the camera for video 6 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 6 are shown.

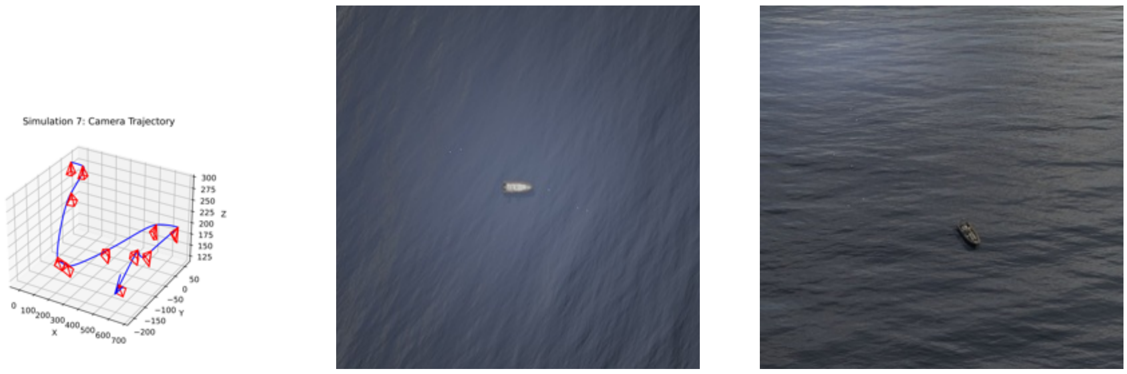


Figure A.7: The trajectory of the camera for video 7 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 7 are shown.

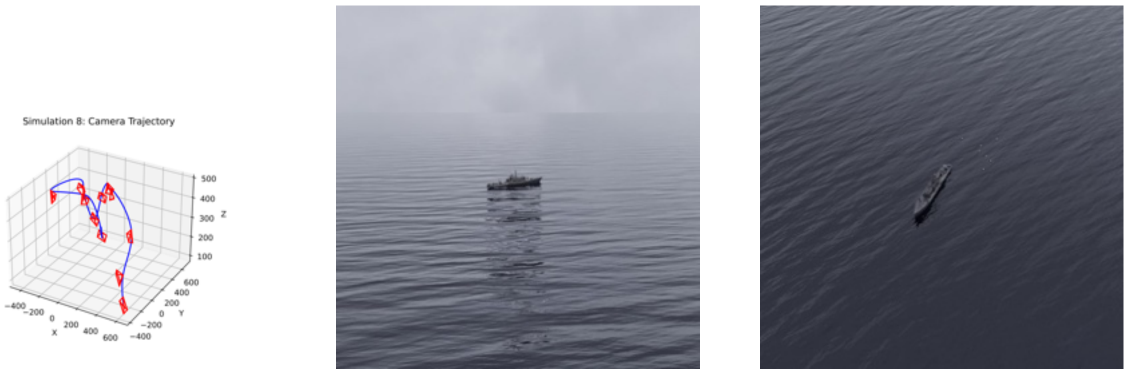


Figure A.8: The trajectory of the camera for video 8 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 8 are shown.

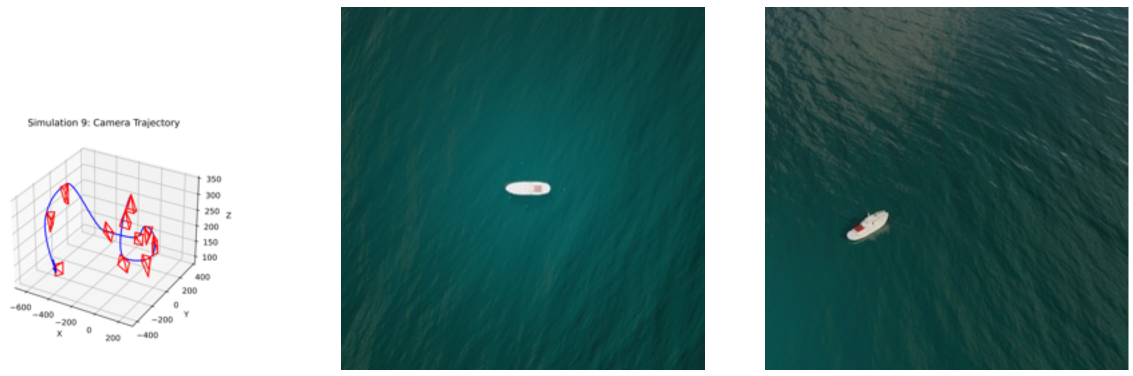


Figure A.9: The trajectory of the camera for video 9 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 9 are shown.

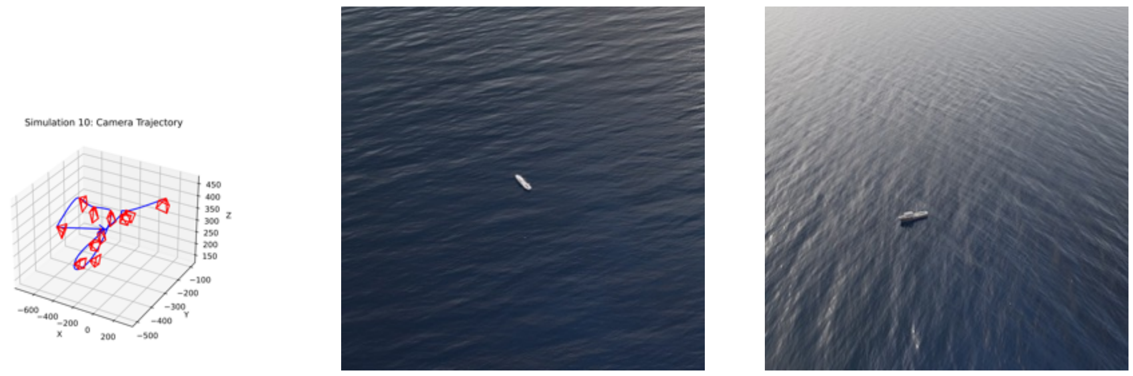


Figure A.10: The trajectory of the camera for video 10 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 10 are shown.

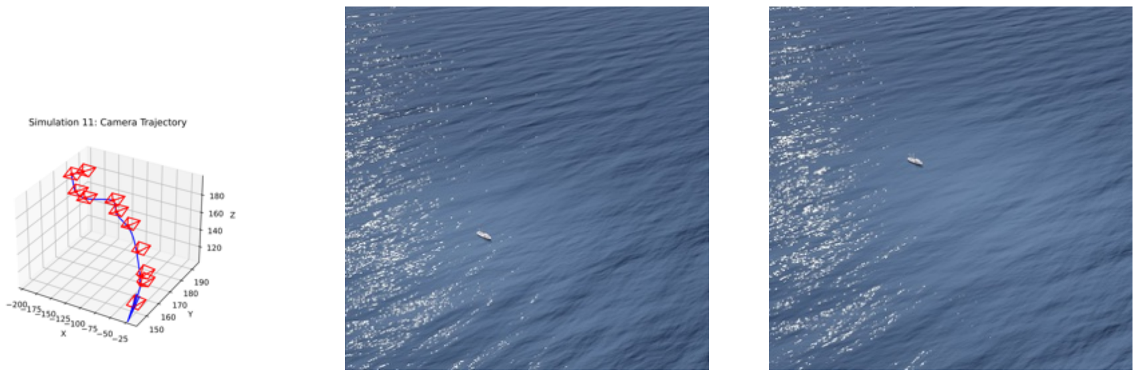


Figure A.11: The trajectory of the camera for video 11 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 11 are shown.

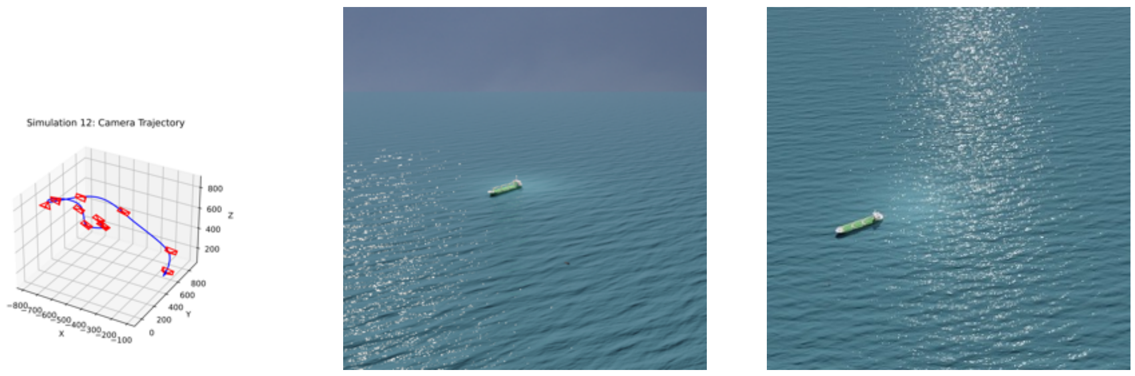


Figure A.12: The trajectory of the camera for video 12 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 12 are shown.

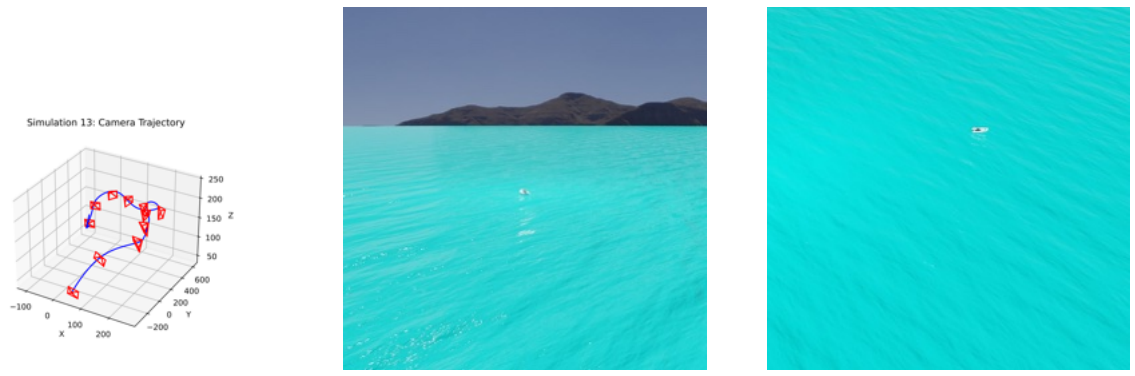


Figure A.13: The trajectory of the camera for video 13 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 13 are shown.

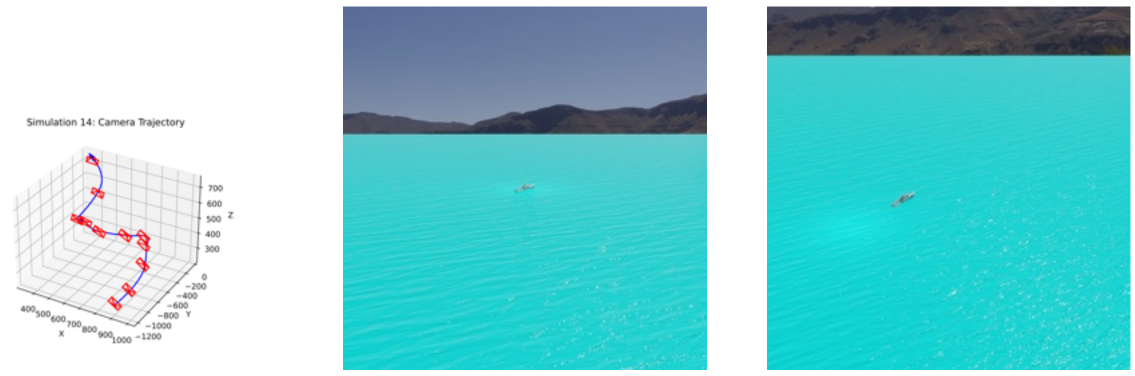


Figure A.14: The trajectory of the camera for video 14 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 14 are shown.

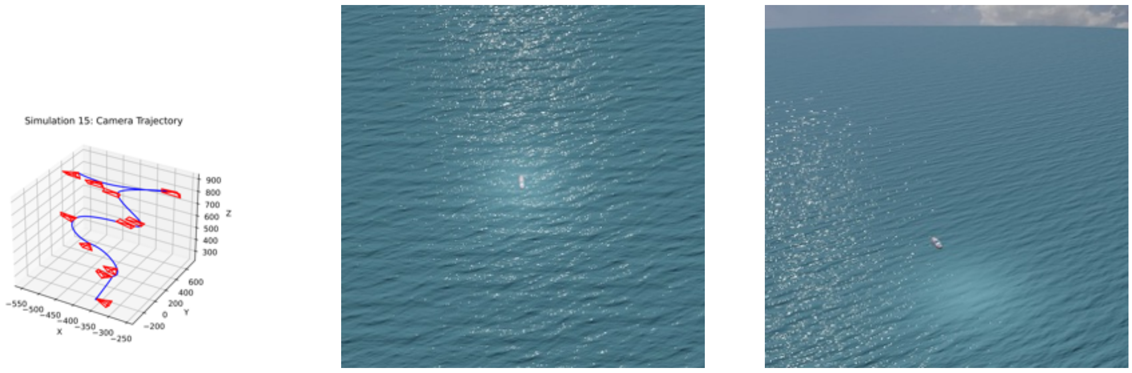


Figure A.15: The trajectory of the camera for video 15 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 15 are shown.

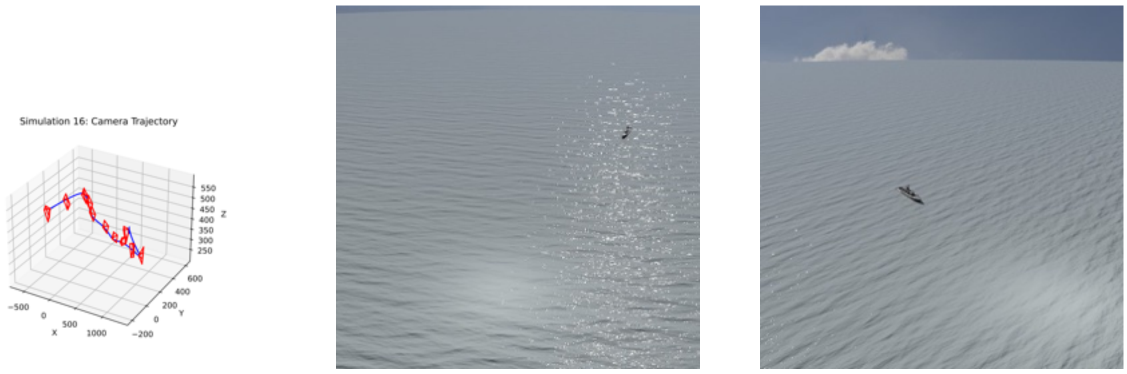


Figure A.16: The trajectory of the camera for video 16 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 16 are shown.

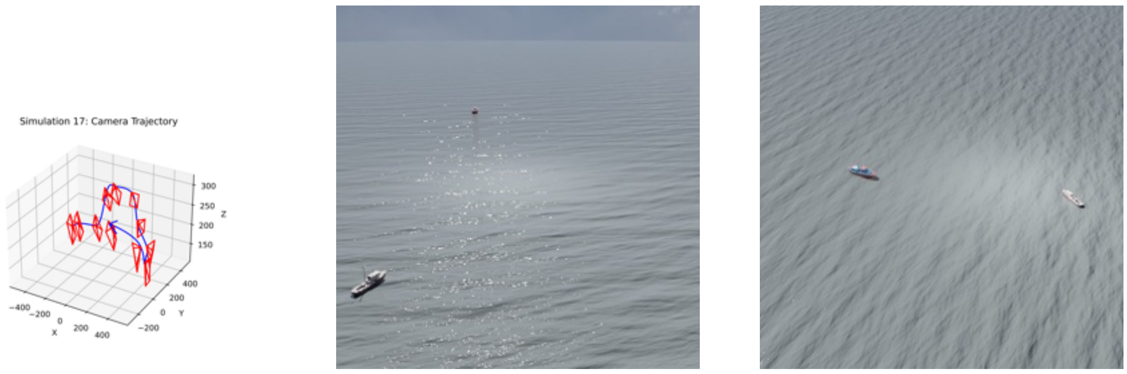


Figure A.17: The trajectory of the camera for video 17 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 17 are shown.

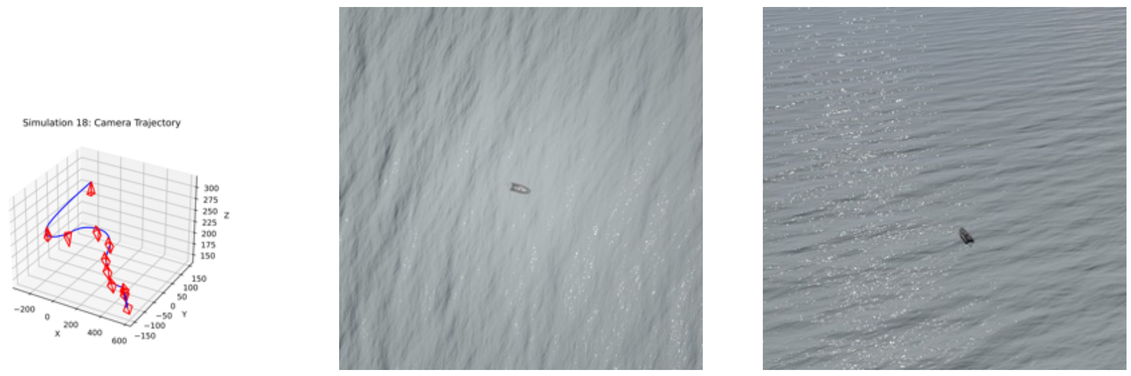


Figure A.18: The trajectory of the camera for video 18 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 18 are shown.

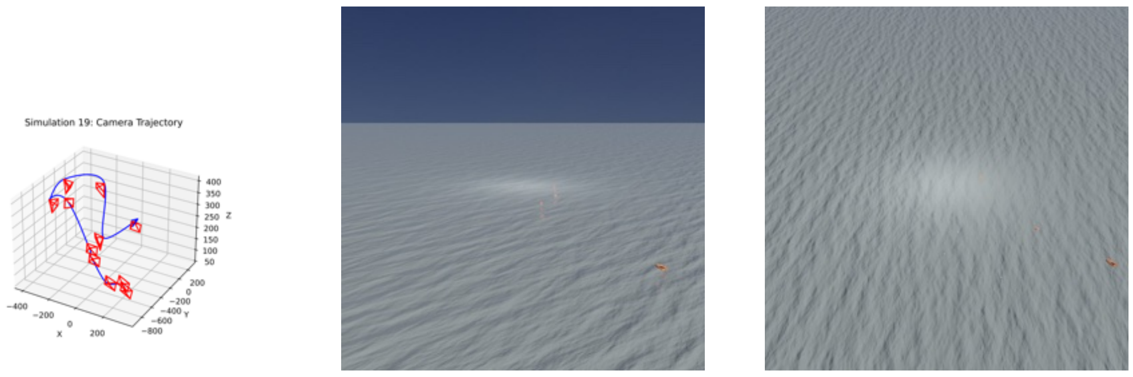


Figure A.19: The trajectory of the camera for video 19 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 19 are shown.

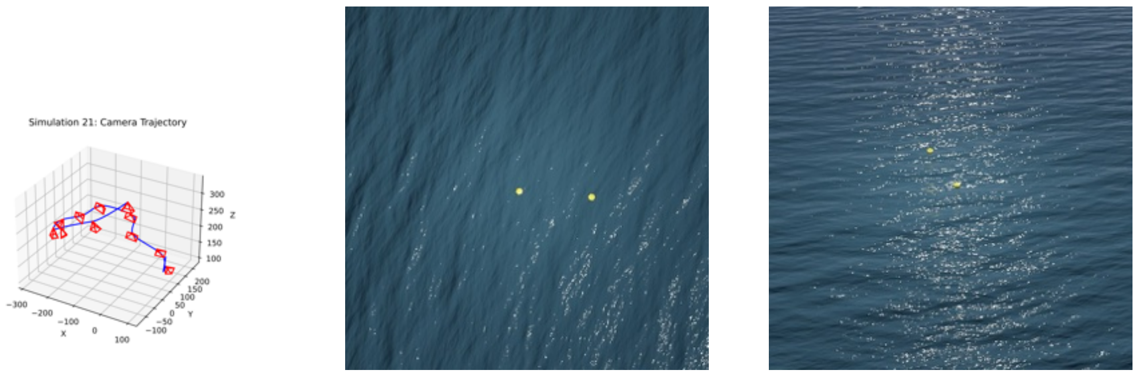
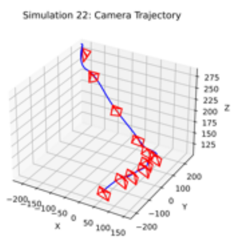
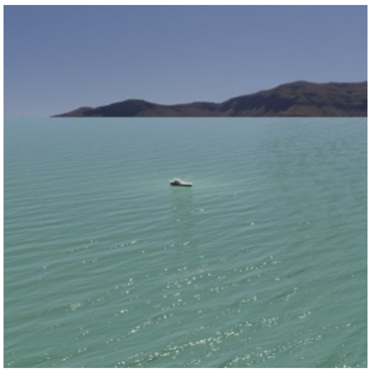


Figure A.20: The trajectory of the camera for video 21 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 21 are shown.





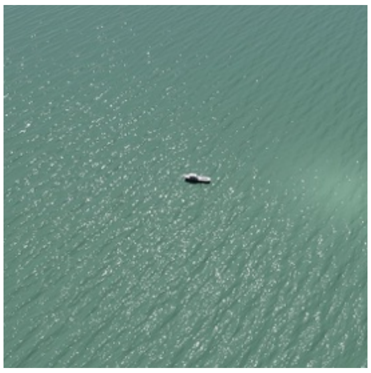
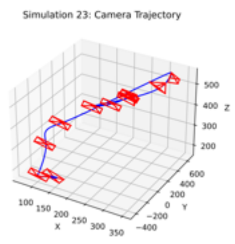
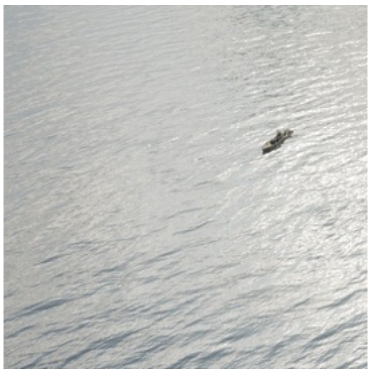


Figure A.21: The trajectory of the camera for video 22 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 22 are shown.





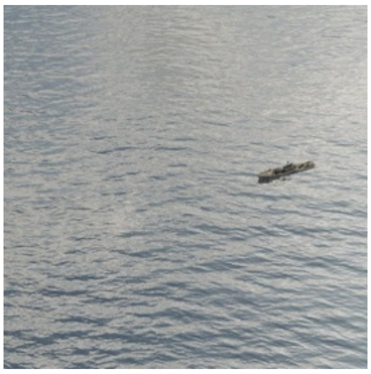
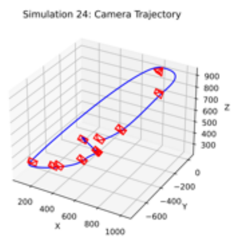
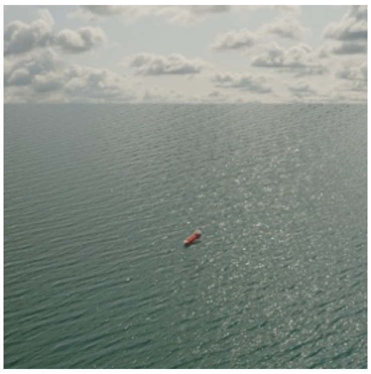


Figure A.22: The trajectory of the camera for video 23 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 23 are shown.





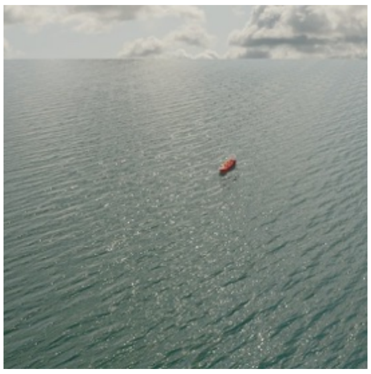


Figure A.23: The trajectory of the camera for video 24 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 24 are shown.

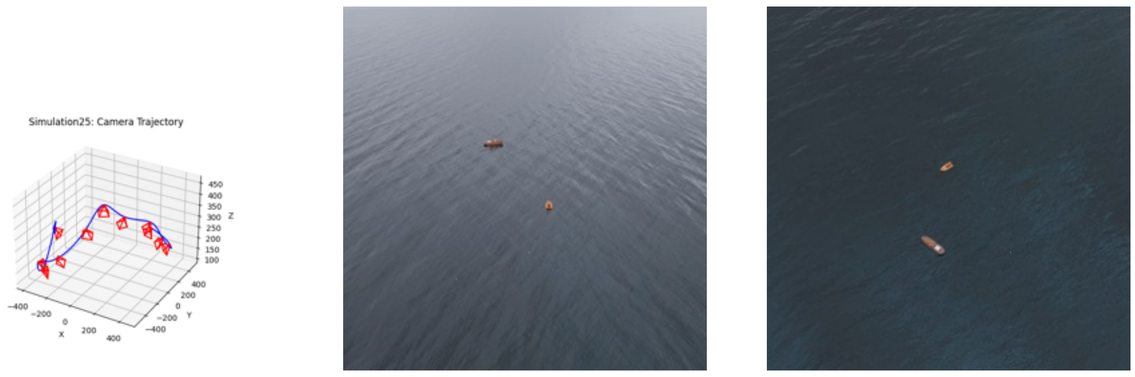


Figure A.24: The trajectory of the camera for video 25 of the dataset is shown on the left. To illustrate the images taken along the trajectory, images Nr. 0 and Nr. 500 from video 25 are shown.