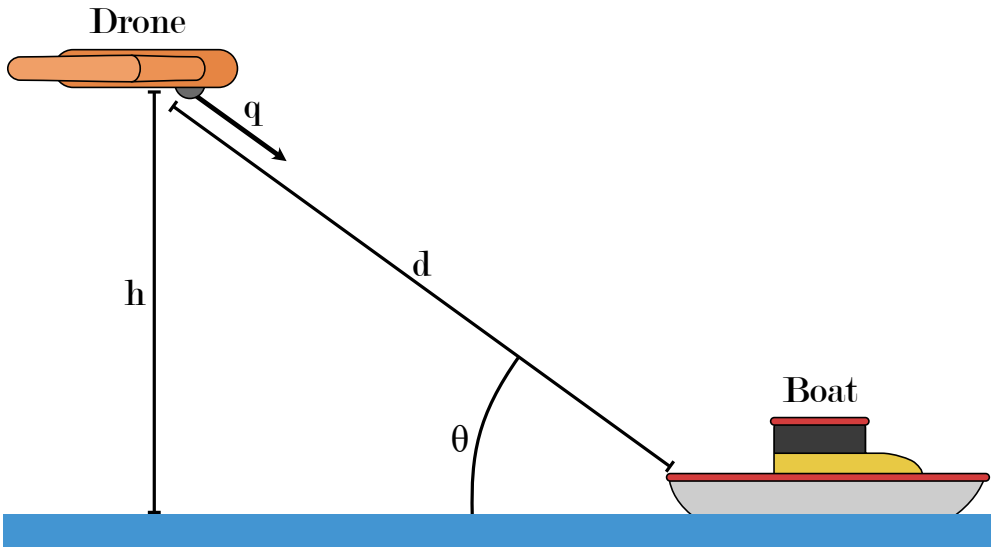


Lightweight Location Estimation of Boats at Sea from Aerial Drone Footage

A modular implementation running on a Raspberry Pi 5

Master's thesis in Complex Adaptive Systems

Lucas Kristiansson
Hampus Olsson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Lightweight Location Estimation of Boats at Sea from Aerial Drone Footage

A modular implementation running on a Raspberry Pi 5

Lucas Kristiansson
Hampus Olsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Lightweight Location Estimation of Boats at Sea from Aerial Drone Footage
A modular implementation running on a Raspberry Pi 5
Lucas Kristiansson, Hampus Olsson

Supervisor: Fredrik Falkman, Svenska Sjöräddningssällskapet
Examiner: Jonas Sjöberg, Professor in Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of a drone and a boat viewed from the side.

Typeset in LaTeX
Gothenburg, Sweden 2025

Lightweight Location Estimation of Boats at Sea from Aerial Drone Footage
A modular implementation running on a Raspberry Pi 5
Lucas Kristiansson, Hampus Olsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis presents a lightweight, modular system for real-time location estimation of boats at sea using aerial drone footage on computationally constrained hardware such as a Raspberry Pi 5, specifically designed for autonomous drone landings.

The architecture comprises three core microservice modules – object detection, tracking, and localization – communicating via a custom Redis-based utility class. Object detection is performed by a YOLO11n model, optimized through transfer learning and deployed within the NCNN framework for efficient inference. A non-neural network tracking algorithm, incorporating a simplified Kalman filter and the Jonker-Volgenant assignment method, manages object association. Two depth estimation techniques used for localization are implemented and compared: one utilizing the drone's altitude and another using the known physical width of the boat.

The developed system achieves a mean end-to-end latency of approximately 183 milliseconds. Comparative analysis revealed that the width-based localization method offers greater accuracy than the altitude-based approach at distances below 60 meters, even considering a mean 0.88-pixel error in the detected boat width from the object detection module (assuming a 1-meter drone altitude error). Furthermore, a proposed simple compensation technique for perspective distortion caused by drone-boat misalignment demonstrated a reduction in mean position error by 73.11%.

Keywords: Object detection, Tracking, Localization, Fixed-wing drone, Aerial imagery, Microservices

Acknowledgments

We would like to thank Jonas Sjöberg for his guidance and inputs during the writing process. We would also like to thank Fredrik Falkman and SSRS for lending us the hardware used for evaluating the performance of our system as well as facilitating the use of their drone and boat for data collection.

Lucas, Kristiansson, Hampus Olsson, Gothenburg, June 2025

List of Abbreviations

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

A-IDC	Average ID Consistency
AI	Artificial Intelligence
AIS	Automatic Identification System
BBox	Bounding box
CNN	Convolutional Neural Network
CPU	Central Processing Unit
IDSW	ID Switch
mAP	mean Average Precision
NED	North-East-Down coordinate frame
ONNX	Open Neural Network Exchange
RPi5	Raspberry Pi CM5
RT-DETR	Real-Time DEtection TRansformer
SSRS	Svenska Sjöräddningssällskapet
YOLO	You Only Look Once

Nomenclature

Below is the nomenclature of variables that have been used throughout this thesis.

$\mathbf{r}$	Detection position array for the tracking module
$\mathbf{r}'$	Predicted position array from simplified Kalman filter for the tracking module
$\mathbf{s}$	Detection size array for the tracking module
$\mathbf{s}'$	Predicted size array from simplified Kalman filter for tracking module
$\mathbf{v}$	Estimated velocity array for the tracking module
dt	Delta time between two video frames
$\mathbb{D}$	Distance matrix for the tracking module
$\mathbb{S}$	Size difference matrix for the tracking module
$\mathbb{C}$	Cost matrix for the tracking module
w_I	Apparent boat width in image measured in pixels
w_W	Physical boat width
l_W	Physical boat length
w_{proj}	Projection of the apparent physical boat width in the perspective of the camera
f_x	The horizontal focal length of the camera in pixels
d	Camera depth (distance from drone to boat along $\mathbf{q}$)
$\mathbf{q}$	Normalized target vector pointing from the drone to the boat
θ	Glide slope (angle between $\mathbf{q}$ and the horizontal plane)
h	Drone altitude from sea level
h_{error}	Measurement error in the drone's altitude
e_{hor}	Horizontal error of the boat's estimated position
ψ	Misalignment angle (angle between the drone and the boat's center-line)
δ	Misalignment distance (perpendicular distance between the drone and the boat's center-line)

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Svenska Sjöräddningssällskapet (SSRS) is developing a drone intended for capturing aerial images of accidents and locating humans in the water [1]. It flies autonomously to waypoints defined by a remote computer on a map interface. The drone sends a real-time video feed back to SSRS which provides an overview that enables the emergency responders to prepare and coordinate the rescue. To maximize battery efficiency, the drone is of the fixed-wing type, meaning it requires a horizontal airspeed to generate lift and cannot land vertically. Sometimes, the accident area is too far away to safely return back to the take-off location with the remaining battery charge. In this situation, the drone is currently ditched and recovered from the water which is not only time consuming but also requires the hardware to be waterproof.

SSRS aims to develop a solution where the drone can be safely retrieved and suggests landing on a nearby rescue boat autonomously. By instructing the boat to move with a steady course in headwind, the drone can descend onto the boat with a low relative speed. In this thesis, we focus on the perception and localization subproblem: detecting boats using the drone's camera and estimating their position relative to the drone.

All rescue boats are required to have an automatic identification system (AIS) transceiver that transmits their positions [2]. However, the high latency and low update frequency of AIS makes it unsuitable for a real time precision landing on a moving boat. It is also not desirable to retrofit the boats with special localization hardware due to the added cost. We therefore limit the solution to only use the drone's camera and onboard computer (a Raspberry Pi CM5 (RPi5)) in addition to existing boat and drone telemetry for localization. Consequently, the solution will be compatible for landing on all boats equipped with AIS.

To operate in real time, the perception pipeline must be both lightweight and self-contained. Therefore, the program was implemented for local execution only (to minimize latency and avoid reliance on an internet connection). To leave computational headroom to other system processes, it was restricted to run on a single CPU thread of the RPi5.

Additionally, SSRS's drone altitude sensor is known to be imprecise and the error has to be measured in order to compensate for it. However, we were unsuccessful in measuring the exact magnitude of it and we therefore evaluated position estimation under a range of plausible altitude errors.

1.2 Contributions

This thesis presents a complete and modular implementation for detecting and localizing boats at sea from aerial images captured by a drone. The program can run locally on a Raspberry Pi CM5 with an end-to-end latency of 183 milliseconds.

The program is a system comprising three core modules, responsible for object detection, tracking and localization respectively. They communicate through our utility class built on the Redis library that provides methods for subscribing to, sending, retrieving and logging messages.

Two methods were implemented and compared for estimating the depth to the landing target: one leveraging the drone's altitude and the other using the known physical width of the boat.

Lastly, we proposed a way to compensate for a misalignment between the drone and the boat's centerline that caused the boat to appear wider. The compensation reduced the mean position error by 73.11% for angles between 0 and 18 degrees.

2
System Architecture

This chapter provides an overview of the implemented system, its modules and how they interact.

2.1 System Modularity

The system is broken down into three main functions: detection, tracking and localization. They are implemented as microservices in isolated Docker containers (see Appendix A.1 for Docker background) and referred to as *modules*. This creates a more manageable and extendable architecture. Additionally, each container is packaged with its dependencies, providing an ease-of-use deployment and avoids compatibility issues.

2.2 Module Interaction

Due to the models being isolated, a method of transferring data in between them is needed. With Redis (see Appendix A.2 for Redis background) as a foundation, a custom utility class (`RedisClient`)[3] was created in Python and utilized in all modules. It provides a standardized framework for publishing/subscribing to messages as well as adding/retrieving data from a stream with a simple syntax. When the `RedisClient` is instantiated, it will attempt to connect to a Redis server running locally on the system. After a connection has been established, the client can subscribe to message channels and publish messages or stream data, thereby connecting the modules. The content of a message is automatically serialized as a JSON string and sent along with a timestamp.

2.3 System Functions

Figure 2.1 illustrates the information flow between the sensors and the three modules. The chain starts with the *detection module* grabbing the latest image from the camera, detecting all boat objects and drawing bounding boxes around them. The detection boxes are outputted alongside a timestamp from when the image was captured. The *tracking module* takes the detection boxes as an input, assigns unique ID's to them and attempts to maintain the same ID for each object in subsequent images. The output is the detection boxes with associated IDs. The *localization*

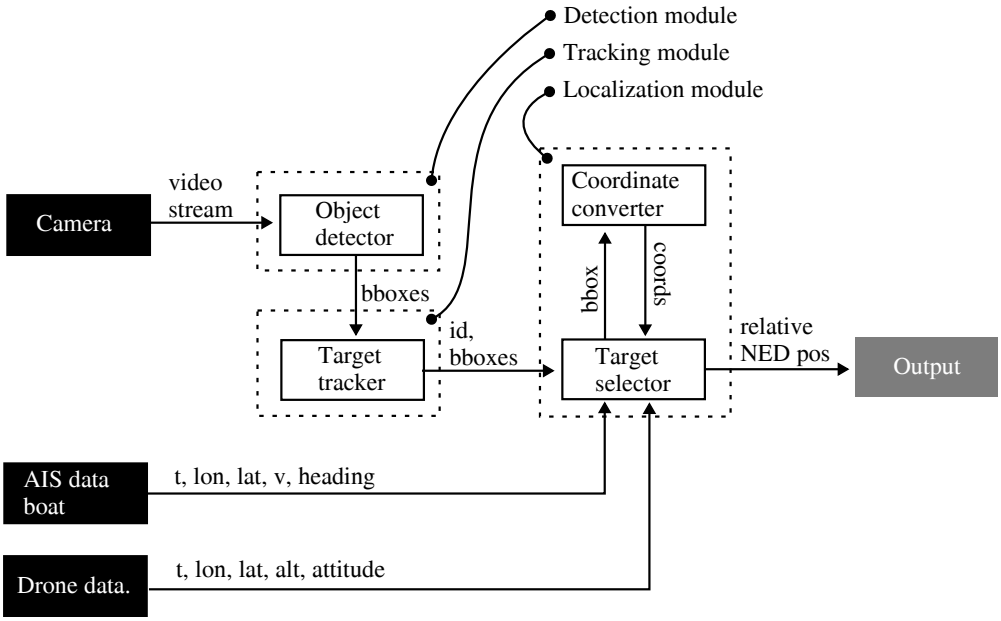


Figure 2.1: An overview of how information flows between the three modules. The black boxes are sensor sources, the white boxes are Python classes and the dashed boxes are the modules. Arrows with labels indicate what information is sent and where.

module selects one detection from the tracking module using a set of conditions and estimates its world-space coordinates relative to the drone. This coordinate is the final output.

3
Detection Module

This chapter covers the implementation of the detection module, analysis of existing object detection models, benchmarks and our motivation for selecting one of the models.

3.1 Description of the Module

The detection module is responsible for processing the live feed from the drone's camera by detecting individual boats in an image and drawing bounding boxes around them. The inputs and output for this module are listed in Table 3.1 and the source code is available at [4].

Table 3.1: Data input and output from the detection module. Camera feed is from the onboard camera. Detection speed is a variable for how frequently the object detector should process an image.

Input	**Output**
Camera feed Detection speed	Detection data

The object detector updates at a set interval. The interval can be dynamically adjusted or set to -1, thereby disabling the detection completely when it is not needed. Each update, the detection's bounding boxes, the image resolution and capture time are sent as a Redis message.

3.2 Analysis of Detection Models and Frameworks

The process of implementing a detection module included a literature study of existing object detection models. We also analyzed various frameworks that the AI-based detection models can be deployed in.

3.2.1 Object Detectors

Due to the widespread use and strong implementation of existing models in industry [5], we analyzed and benchmarked several of them to evaluate their suitability for our system, rather than developing a novel solution from scratch. The models analyzed were YOLO, RT-DETR and a filter-based algorithm.

You Only Look Once (YOLO) is a popular CNN-based object detector that is known for its real-time object detection and can be employed in scripts using Ultralytics' own python package [6], [7]. Real-Time DEtector TRansformer (RT-DETR), conversely, is a transformer-based object detector which the authors claim to be superior to YOLO in terms of precision and speed [8]. Lastly, filter-based algorithm [9] is a non NN-based detection algorithm specifically designed for detecting vessels in aerial footage using image filtering methods.

3.2.2 Frameworks for Neural Network Inference

YOLO and RT-DETR are based on neural network processing and can be deployed using different frameworks. They were designed in the PyTorch [10] framework, which is not optimized for low latency or resource constrained systems. Therefore, the models were also tested after converting them to the ONNX and NCNN framework.

Open Neural Network Exchange (ONNX) enables a model conversion that is optimized across a variety of different platforms, improving both speed and reducing model size [11]. NCNN is a high-performance inference framework that is specifically optimized to run on devices using ARM64 architecture, such as Raspberry Pi [12].

3.3 Test Setup for Neural Network Inference

To select the most suitable object detector and deployment framework, we evaluated the models on a dataset containing aerial footage of boats where, for each picture, the detected bounding boxes was compared to the ground truth bounding boxes. The three metrics used for evaluating the performance was inference time (time from the object detection model receiving an image to outputting a result), detection precision (measured as mAP, explained in Appendix B) and ease-of-use in implementation. The most critical metric was inference time , since the system had to be lightweight.

From early testing of the filter-based algorithm, we found it to be easily overfitted and unreliable in detecting boats. Additionally, it had higher inference time than YOLO. Therefore, this model was eliminated and not included in the next chapter.

YOLO and RT-DETR were tested using publicly available, pre-trained weights from Ultralytics and converted into the three frameworks: PyTorch, ONNX and NCNN. We included the current version of YOLO as well as older versions, all with their respective smallest model sizes and the RT-DETR-L model. An overview of each model can be found in Table 3.2.

The models were validated on a custom dataset provided by SSRS containing aerial drone footage of a boat cruising at sea, using Ultralytics' library for model vali-

Table 3.2: An overview of evaluated object detection models: Lists every model that was evaluated on a test dataset together with its framework. Params represents how many learnable parameters each network has in millions. More parameters typically indicate a longer inference time. Note that not every model is NCNN compatible.

Model	Params [M]	PyTorch	ONNX	NCNN
YOLOv8n	3.2	✓	✓	✓
YOLOv9t	2.0	✓	✓	✓
YOLOv10n	2.0	✓	✓	-
YOLO11n	2.6	✓	✓	✓
RT-DETR-L	45.3	✓	✓	-

dation. The models were tested on a single thread of an RPi5. After testing and assessing the models, we trained the best performing model on a custom dataset to confirm that transfer learning would improve its precision.

3.4 Test Results

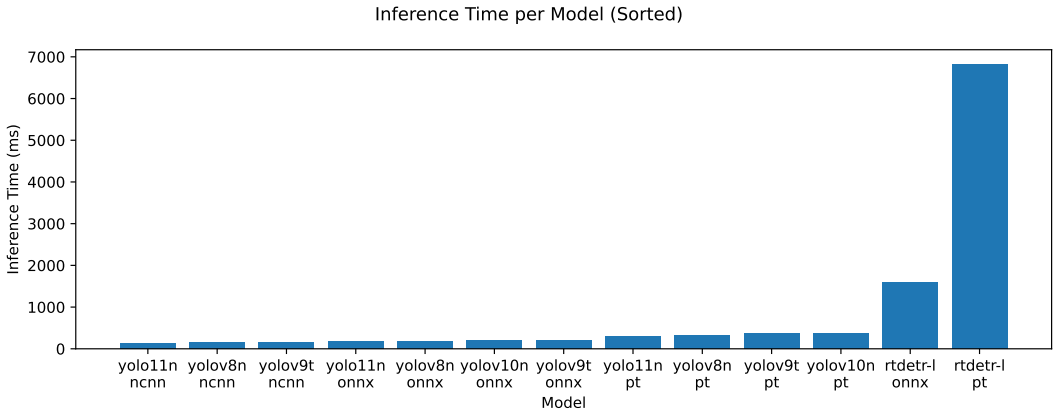


Figure 3.1: Comparison of object detection models evaluated on inference time using a single CPU thread. Lower is better.

Figure 3.1 shows the average inference time for each model-framework combination. It is evident that RT-DETR-L is computationally heavy, leading to significantly higher inference time. Therefore, it is not suitable for single threaded inference and was therefore eliminated as a candidate. Figure 3.2 shows the same results as Figure 3.1, excluding the RT-DETR model.

Figure 3.2 reveals a trend with the NCNN framework being the most efficient in terms of inference time, followed by ONNX and lastly PyTorch. The conclusion of this finding was that using a NCNN framework is the most ideal for ARM64 based processors. The conversion was also found to be simple and streamlined, thus resulting in NCNN being the framework we continued to analyze. Table 3.3 shows the resulting precision scores and inference times for the NCNN models.

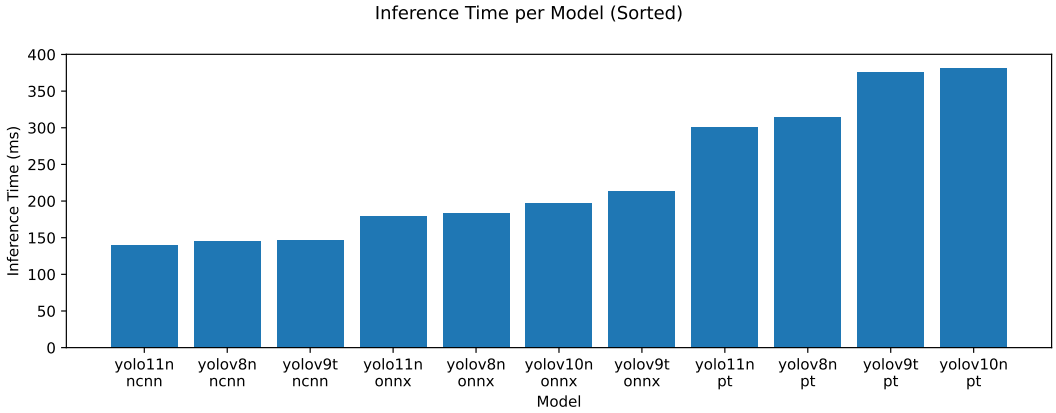


Figure 3.2: Comparison of object detection models (excluding RT-DETR-L) evaluated on inference time using a single CPU thread. Lower is better.

Table 3.3: Precision score and inference metrics of the detection models including a self-trained version running in NCNN framework. Higher mAP is better. Lower time is better.

NCNN model	mAP@50	mAP@50:90	Time [ms]
YOLO11n	0.2170	0.1352	**144.62**
YOLOv8n	0.2440	0.1682	151.99
YOLOv9t	0.1922	0.1318	152.32
YOLO11n (ours)	**0.7633**	**0.4565**	148.58

In Table 3.3, we see that YOLO11n is the fastest model. However, it does not have the highest precision score, but we assessed that the mAP difference is negligible, due to the possibility of improving precision with transfer learning. After training the model on images of boats, specifically the ones SSRS uses, we were able to achieve a higher precision, listed in the bottom row of Table 3.3. We selected YOLO11n with the improved parameter set from transfer learning as the model of choice in the detection module.

4

Tracking Module

This chapter covers our implementation of an object tracker and the results from testing it in a simulated environment.

4.1 Description of the Module

The tracking module processes detection data to associate objects across consecutive images (frames) by assigning unique IDs to them based on their position- and size-differences. The input and output are listed in Table 3.1 and the source code is available at [13].

Table 4.1: Data input and output from the detection module.

Input	**Output**
Detection data	Detection data with associated IDs

Ultralytics – the creators of YOLO11 – provides two AI-based tracking models in their Python package. However, they are only deployable in the PyTorch framework, thus restricting the use of NCNN. Testing the two methods revealed an average inference time of 441.57 ms. Compared to using the chosen NCNN detection model (recall Section 3.4), this would add roughly 300 ms of extra inference time. Furthermore, both tracking models demonstrated unreliable performance at the tested frame rates: ID assignments were inconsistent, and some detections were not assigned any ID at all. As a result, we concluded that these models were unsuitable for our application and instead chose to implement a non-neural-network-based tracking solution.

The implementation was inspired by [14] where a Kalman filter and iterative-Hungarian algorithm was used for real-time tracking of objects. Our tracking algorithm utilizes a simplified Kalman filter that stores previous detections as states in its memory. A schematic overview of our implementation is illustrated in Figure 4.1 and explained in detail below.

When tracking in the current frame (i), the algorithm organizes the incoming detection data (Figure 4.1, (1)) into a position array ($\mathbf{r}_i$), a size array ($\mathbf{s}_i$) and the capture time (t_i), as follows:

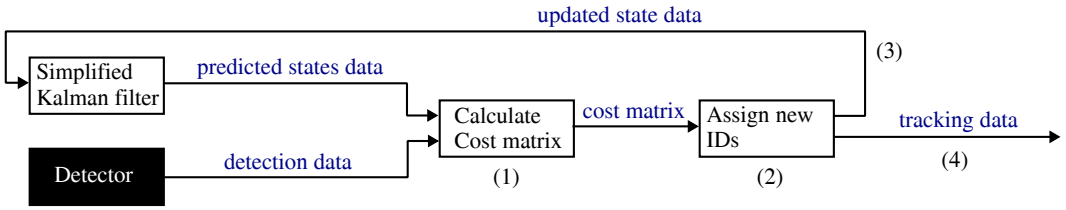


Figure 4.1: Schematic overview of the tracking algorithm: Detection data and predicted states data is fed to the tracker. It combines them to a weighted cost matrix (1) and then assign IDs to the new detection boxes based on the cost matrix. Unmatched detections are assigned new IDs (2). The tracker then updates the states of the simplified Kalman filter (3), and sends the new data as output (4).

$$\mathbf{r}_i = \begin{bmatrix} x_1, & y_1 \\ x_2, & y_2 \\ \vdots & \vdots \\ x_n, & y_n \end{bmatrix}, \quad \mathbf{s}_i = \begin{bmatrix} w_1 \cdot h_1 \\ w_2 \cdot h_2 \\ \vdots \\ w_n \cdot h_n \end{bmatrix}, \quad t_i, \tag{4.1}$$

where n is the number of detections and i is the index of the current frame. Using the capture time, the algorithm predicts the future states $(\mathbf{r}'_i, \mathbf{s}'_i)$ of previous detections through a simplified Kalman filter:

$$\mathbf{r}'_i = \mathbf{r}_{i-1} + \mathbf{v}_{i-1} \cdot dt, \quad \mathbf{s}'_i = \mathbf{s}_{i-1}, \quad dt_i = t_i - t'_{i-1}, \quad \mathrm{ID}_i = \mathrm{ID}_{i-1}, \tag{4.2}$$

where $\mathbf{v}_{i-1}$ is the estimated velocities of the detections from the previous frame, dt_i the time between the the two latest frames and ID_i the array of unique assigned IDs for previous states. The distance matrix ($\mathbb{D}$) and the size-difference matrix ($\mathbb{S}$) are then defined as:

$$\mathbb{D} = || \mathbf{r}'_i - \mathbf{r}_i ||, \tag{4.3}$$

$$\mathbb{S} = | \mathbf{s}'_i - \mathbf{s}_i | . \tag{4.4}$$

The matrices are element-wise normalized to the range [0-1]:

$$\mathbb{D}_{norm} = \frac{\mathbb{D} - \min(\mathbb{D})}{\max(\mathbb{D}) - \min(\mathbb{D})}, \tag{4.5}$$

$$\mathbb{S}_{norm} = \frac{\mathbb{S} - \min(\mathbb{S})}{\max(\mathbb{S}) - \min(\mathbb{S})}. \tag{4.6}$$

The cost matrix $\mathbb{C}$ is then defined as:

$$\mathbb{C} = \alpha \cdot \mathbb{D}_{norm} + \beta \cdot \mathbb{S}_{norm} = \begin{bmatrix} c_{11}, & c_{12}, & \dots, & c_{1n} \\ c_{21}, & c_{22}, & \dots, & c_{2n} \\ \vdots & \vdots & & \vdots \\ c_{n'1}, & c_{n'2}, & \dots, & c_{n'n} \end{bmatrix}, \tag{4.7}$$

where α and β are weight coefficients for the distance and size differences, respectively. They are complementary such that $\alpha + \beta = 1$, resulting in the cost $c \in [0, 1]$. The shape of the cost matrix is dictated by the number of detections, n, and the number of predictions from the Kalman filter, n'.

To find the matches between detections and predictions that minimizes their total cost, we used SciPy's function for linear sum assignment [15], based on the Jonker-Volgenant algorithm. This allowed for a non-square $\mathbb{C}$, in contrast to the Iterative-Hungarian method used in the original algorithm. This is beneficial when a new object enters the view and the number of detections is more than the number of predictions.

The ID of each prediction is transferred to its matched detection. Unmatched detections indicate that new objects have appeared in the view and should be assigned new, unique IDs (Figure 4.1, (2)). Similarly, unmatched elements in the predictions indicate that previously tracked objects have disappeared from the view. If they remain undetected for a certain period of time, their states are removed from the memory and will not be considered for future matching.

We introduced a threshold parameter, γ, to discard matches with costs greater than the threshold. This is important when a previously undetected boat appears in the view and an old boat has disappeared but still is present in the memory. The threshold ensures that the new boat is assigned a new and unique ID instead of the old boat's ID.

The detections and their IDs are fed back to the Kalman filter (Figure 4.1, (3)) and used as output for this module (Figure 4.1, (4)).

4.2 Validation of Tracker

The tracker's performance was evaluated based on its ability to assign the same ID to the same object during a period of time. Traditional multi-object tracking metrics such as MOTA or IDF1 were not applicable as they include the detection precision in the performance calculation. Our tracker is isolated from the detection process and we therefore created our own metric named Average ID Consistency (A-IDC), that only considers the tracker's performance.

A-IDC quantifies the average frame-to-frame ID consistency over a period of time based on the proportion of ID switches for each frame. For frame i, an ID switch is counted for every object whose assigned ID is not matching its ID in the previous frame $(i-1)$. The total number of ID switches (IDSW) divided by the total number of visible objects in frame i gives the ID consistency of that frame and the A-IDC takes the average over multiple frames:

$$\text{A-IDC} = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{\text{IDSW}_i}{\text{Objects}_i} \right), \tag{4.8}$$

where N is the total number of frames in the time period.

This metric emphasizes consistent ID matches and penalizes ID swaps between two objects more than it penalizes assigning new IDs to individual objects. We also recorded the sum of IDSW for the time period as a secondary indicator of performance, thus giving an absolute measurement of the number of errors made by the tracker.

4.3 Simulation & Results

To validate the implemented tracker, we simulated detection boxes that moved around randomly with a possibility of moving in and out of the visible space. In each frame, the tracker was fed the position and size of the visible boxes.

First, all combinations of α and β with increments of 0.05 were evaluated with γ set to 1. Table 4.2 shows the combinations with the best A-IDC and IDSW scores averaged over 100 simulations. Each simulation contained 10 objects and ran for 20 seconds with a tracking update rate of 5 Hz (similar to the object detector inference rate achieved in Section 3.4). The number of objects was set to 10 to stress test the tracker for complex scenarios. In ideal real-life conditions, much fewer objects are expected to be in view.

Table 4.2: Score metrics for best performing α and β combinations. The result is from 100 simulations of 10 objects for 100 frames.

Rank	α	β	A-IDC	IDSW
1	0.95	0.05	0.9184	55.21
2	0.90	0.10	0.9055	63.27
3	0.85	0.15	0.8947	69.73
4	0.80	0.20	0.8888	74.52
5	0.75	0.25	0.8807	78.42

The result from Table 4.2 shows a trend of α being significantly higher than β, indicating that the spatial distance is more relevant for tracking than the size difference.

Table 4.3: Score metrics for best performing γ values. $\alpha = 0.95$, $\beta = 0.05$. The result is from 100 simulations of 10 objects for 100 frames.

Rank	γ	A-IDC	IDSW
1	0.15	0.9475	35.97
2	0.2	0.9424	39.14
3	0.25	0.9396	40.68
4	0.1	0.9386	41.57
5	0.3	0.9337	45.12

During this simulation γ was set to 1, resulting in no constraint in high cost entries. By further testing the best performing α-β pair with varying γ, we obtained the results detailed in Table 4.3. This shows that introducing the threshold γ reduced the number of ID switches significantly, as compared to Table 4.2. The A-IDC score also improved, albeit less significantly. This was due to the metric being averaged over a large number of frames in which no ID switches occur.

A second test was conducted to evaluate how sensitive the tracking computation time is to the number of tracked objects present in the scene. Seven scenarios were tested, each with a different number of objects. The average per-frame tracking time for each scenario is presented in Table 4.4.

Table 4.4: Calculation time of the tracker - the average time it takes for the tracking algorithm to track one frame.

Number of objects	Time (ms)
1	0.3295
5	0.5553
10	0.8549
50	3.3417
100	7.8234
500	*121.9332*
1000	*487.8311*

The results show that the tracker achieves high speeds in the expected operational range, where typically a very few objects are visible per frame. Computation time does increase exponentially with the number of tracked objects but remains negligible relative to the detection modules inference time (recall Figure 3.2) below 100 objects. Beyond this point, processing time is significantly higher. Nevertheless, these scenarios are not expected in the deployed system, as having more than 10 objects visible on screen would indicate an overly crowded scene – this is undesirable for precision landing applications.

4.4 Discussion of Simulation Results

The results show that our tracker performs well across a wide range of parameter settings, with generally high A-IDC scores for high α values and lower γ values. While there are measurable differences between the combinations, the difference in tracking performance among the top performing combinations is small. This suggests that there is some leniency in the parameter value selection and room for adjustment to account for edge cases in the specific application.

Higher values of α indicate that spatial distance is a more informative feature than size difference in the tested scenarios. However, in the intended application, it may

still be beneficial to retain some emphasis on size. For example, at lower drone altitudes, other boats may cross paths with the target, causing their bounding boxes to overlap. In such situations, relying solely on spatial proximity may result in incorrect ID assignment as the distances between the objects can be small and the risk for ID swaps high. Lastly, the threshold parameter γ has proven effective for reducing the amount of ID switches and should be kept at a relatively low value.

Based on this evaluation, the final configuration used in the system was:

$$\alpha = 0.9, \quad \beta = 0.1, \quad \gamma = 0.15.$$

These values serve as a good starting point but might need to be adjusted for the specific environment the drone is intended to operate in.

5
Localization Module

This chapter covers the implementation, tests and performance of the localization module.

5.1 Description of the Module

This module selects one detection from the tracking module using a set of conditions and estimates its world-space position relative to the drone. Its input and outputs are listed in Table 5.1 and source code is available at [16].

Table 5.1: Data input and output from the localization module.

Input	**Output**
Tracked detections Drone position and attitude Camera attitude Target boat's MMSI	Boat position

To select a landing target in the list of detections from the tracking module, a decision list is traversed and the first satisfied condition determines the method of target selection. There are three conditions, each listed below alongside their respective method:

1. **Condition:** One of the incoming detection IDs corresponds to the saved target ID from the previous update.*
   Method: Continue to use the same target.
2. **Condition:** A Maritime mobile service identity (MMSI) has been set as the intended target.
   Method: Convert all detections to world-space and select the one closest to the AIS position of the MMSI target.**
3. **Condition:** Default. The above cases was not satisfied.
   Method: Select the centermost detection in the image as the new target.

* In the very first update, the previous target is undefined and condition 1 can not be satisfied.

** An MMSI can be set through a Redis stream in which case the last transmitted position of that boat will be acquired from an AIS database.

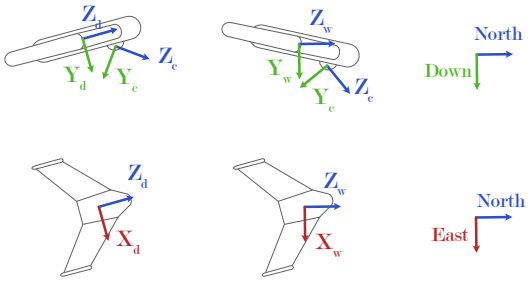


Figure 5.1: Reference frames. On the top is a side-view of the drone and on the bottom is a top-view of the drone. The subscripts stand for drone (d), world (w) and camera (c). The camera-frame rotates with the gimbal camera and it's Z-axis is aligned with the optical axis. The world-frame uses the same axes as the NED coordinates. They are aligned with the cardinal directions and has a local, geodesic origin, in this case the drone.

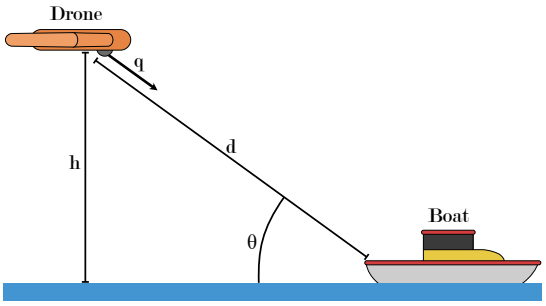


Figure 5.2: Side-view showing the target vector ($\mathbf{q}$), altitude (h), glide slope angle (θ) and camera depth (d).

Once a detection is selected as the target, its ID is stored for use in subsequent updates. The screen-space coordinate of the target – defined as the bottom-center point of its bounding box – is then converted into a world-space position relative to the drone, expressed in the North-East-Down (NED) coordinate frame. An overview of the coordinate systems and reference frames involved is shown in Figure 5.1.

The coordinate conversion process follows the pinhole camera model described in Appendix C. The result is the target vector $\mathbf{q}$, pointing from the camera origin toward the target (Figure 5.2). Note that $\mathbf{q}$ is expressed in image plane coordinates, where $Z_c = 1$, as explained in Appendix C. To obtain the estimated 3D position of the target, this vector must be scaled by the unknown depth d.

We evaluate two methods for estimating d using data from sensors already available on SSRS's drone: the onboard camera image and the estimated altitude. The first option, proposed by [17], is to assume that a boat will always be located at sea level ($Y_w = h$) and calculate the depth as:

$$d = \frac{h}{Y_{\bar{\mathbf{q}}}} \tag{5.1}$$

where h is the camera altitude (drone's altitude adjusted for the offset between the drone center and the camera) and $Y_{\bar{\mathbf{q}}}$ is the vertical component of the normalized target vector $\bar{\mathbf{q}}$. We refer to this as the *altitude-based method*.

The second option is to acquire the width of the boat, which is transmitted as part of AIS and referred to as the *width-based method*. By using the ratio between the real width in meters (w_W) and the bounding box width of the detection in pixels (w_I) alongside the camera's horizontal focal length in pixels (f_x), the depth can be

calculated using the pinhole camera model:

$$d = f_x \frac{w_W}{w_I} \tag{5.2}$$

It is important to consider which physical part of the boat the measured image width, w_I, corresponds to. If the boat's actual width is constant and the camera is positioned behind the boat, the stern will appear widest in the camera's perspective due to its proximity. Consequently, the estimated depth will correspond to the distance between the camera and the stern. To limit the scope of the problem, we assume that the boat width reported by AIS messages refers to the stern width, and that the stern always appears widest from the camera's point of view when the camera is positioned behind the boat along its centerline.

5.2 Test Environment

The positional accuracy of the width- and altitude-based methods was tested using a virtual environment constructed in the game engine Godot [18]. This allowed for flexibility in testing a variety of scenarios and ensured exact and reliable values in the ground truth positions. The environment contained a boat on a body of water and a movable camera. Whenever an image was captured, the boat's world position, screen position and heading was recorded as well as the camera's world position and rotation. The images were captured with a resolution of 640×640 (the internal resolution of our YOLO model) and the bounding box width (w_I) of the boat was collected by manual measurements.

5.3 Test Results

This section presents coordinate conversion errors from using the width- and altitude-based methods and evaluations of how altitude, angle, distance and bounding box inaccuracies effect positional accuracy.

5.3.1 Altitude-based Method

When using the exact screen position and drone altitude from the virtual environment, the position error was negligible and likely only caused by floating point errors. We did not expect the indicated altitude (h) measured by a real drone to be exact and evaluated the localization's sensitivity to an altitude error (h_{error}). As visualized in Figure 5.3, the altitude error forms a right triangle with the horizontal component of the position error (e_{hor}) and the glide slope (θ). From that, we can estimate the horizontal position error with Equation (5.3). The vertical component of the position error will just be h_{error}. Figure 5.4 shows the effect of different θ and h_{error} on the total position error.

$$e_{hor} = \frac{h_{error}}{\tan(\theta)} \tag{5.3}$$

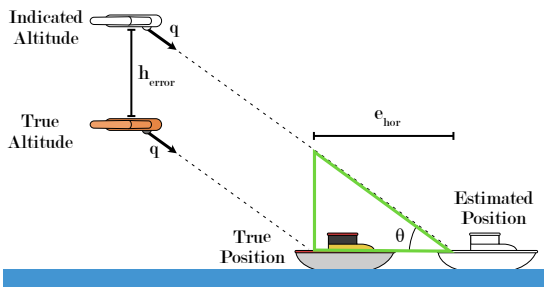


Figure 5.3: Side-view showing how an altitude error (h_{error}) causes a horizontal position error (e_{hor}). The altitude error forms a right triangle with the horizontal component of the position error (e_{hor}).

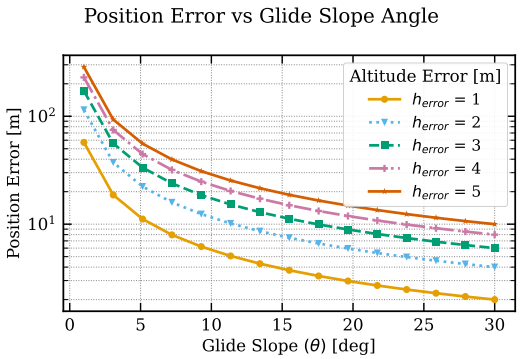


Figure 5.4: Position error's dependency on the glide slope angle (θ) for multiple altitude errors (h_{error}).

In order to minimize the position error, θ should be close to 90 deg. Normally, the angle must be significantly lower to manage vertical speed, however, in our case, the landing target (boat) will be moving – the motion of the boat in combination with a possible head wind can reduce the minimum horizontal relative speed to zero without stalling the drone. In this case, it is possible to approach the boat at a higher θ.

5.3.2 Width-based Method

Figure 5.5 shows how the position error from coordinate conversion depends on the horizontal distance between the boat and the camera. The camera was positioned along the boat's center-line, 15 meters above it. We observe that the error is small – less than 1 meter – when the distance is below 60 meters. In this region, the position estimation would be precise enough to use as a landing target for a drone. At distances greater than 60 meters, the error becomes quasi-periodic with high amplitudes. The local minimums observed in Figure 5.5 could be a result of the boat's bounding box width (w_I) being discrete. At some distances, the correct width is an even number of pixels and the discretization has no effect.

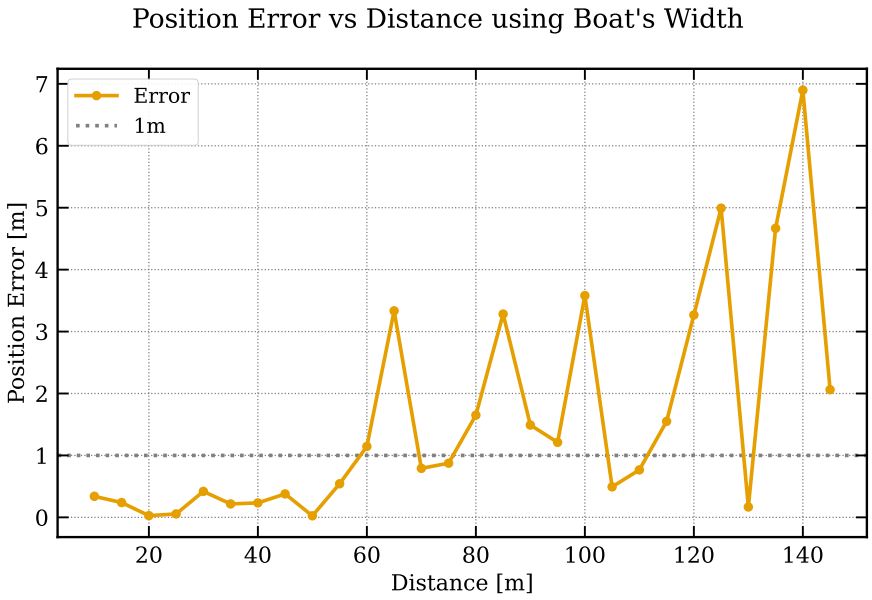


Figure 5.5: Position error's dependency on distance when estimating depth from width.

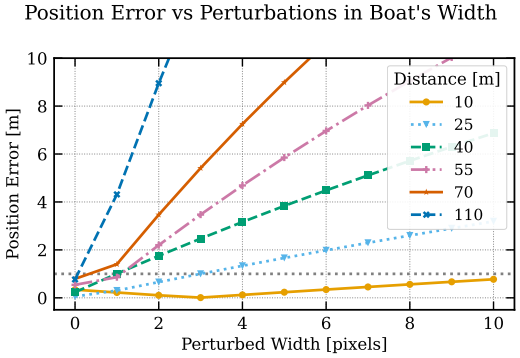


Figure 5.6: The positional error's dependency on a small increase in the boat's bounding box width (perturbation).

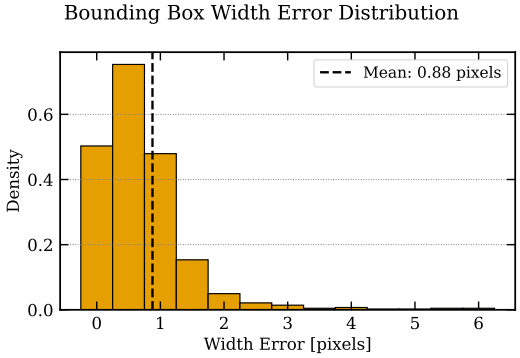


Figure 5.7: A distribution of the width error for bounding boxes created by our trained detection module. It was evaluated on 858 images of an SSRS boat. The ground truth was created by a larger YOLO model (YOLO11x).

Rounding error from discretization is not the only contributor to a width error – it can be caused by image compression or inaccuracies of bounding box sizes from the object detection model. Figure 5.6 shows how the position error depends on perturbations in w_I at multiple distances. Because the depth – and thus the position error – is inversely proportional to w_I (recall Equation (5.2)), it is more sensitive to errors when w_I is small. Naturally, the boat's width appears smaller in the image when it is further away (cf. Figure D.1). Additionally, a one pixel perturbation is proportionally larger for smaller w_I values. We concluded that this method was viable (less than 1 meter error) when the drone was closer than 60 meters if w_I could be reliably measured and suspected that increasing the horizontal resolution of the image would increase the viable distance. Our evaluation of the detection module revealed a mean width error of 0.88 pixels (Figure 5.7) which lowered the

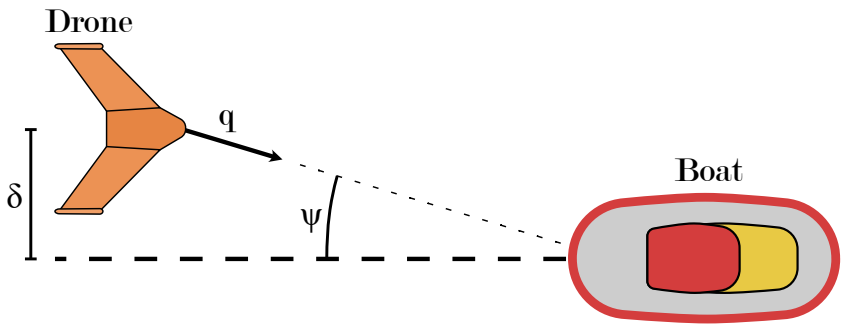


Figure 5.8: Top-view showing the drones misalignment angle (ψ) and misalignment distance (δ) from the boat's center-line.

viable distance to approximately 55 meters.

Another source of error occurred when the camera was not positioned directly behind the boat. We call the perpendicular offset between the boat's centerline and the camera the misalignment distance (δ) and the angle between the boat's center-line and the drone the misalignment angle (ψ), see Figure 5.8. We found that when δ was smaller than half of the boat's width, w_I – and thus the position error – remained unchanged. When δ was larger, the side of the boat was visible in the camera's view, increasing w_I. Figure 5.9 illustrates this phenomenon and Figure 5.10 shows the positional error's dependency on the misalignment. We concluded that it is safe for the drone to be slightly misaligned without effecting the accuracy of the localization. When δ is larger, a strong correlation between it and the position error is observed. In addition to an increased w_I, the bounding box's center would also be shifted by the misalignment, contributing further to the position error.

In practice, the misalignment distance (δ) is unknown to the drone. We therefore studied whether or not the misalignment angle (ψ) – which can be calculated from the target vector ($\mathbf{q}$) – could be used to correct the bounding box's width. Figure 5.11a shows how the position error depends on ψ for multiple distances and by how much w_I changes. Since the distance to the boat – and thus the depth (d) – is independent of ψ, w_I should ideally remain constant as well. The topic of width compensation could be explored in great detail, but was out of scope for this thesis. We did, however, attempt a simple compensation by projecting the boat's width (w_W) and length (l_W) onto a plane, normal to $\mathbf{q}$. The projected width (w_{proj}) can be written as Equation (5.4) and following manual adjustments of the terms, we found that Equation (5.5) minimized the mean position error (Figure 5.11b). We also found that it was best to only use the corrected width when ψ was greater than

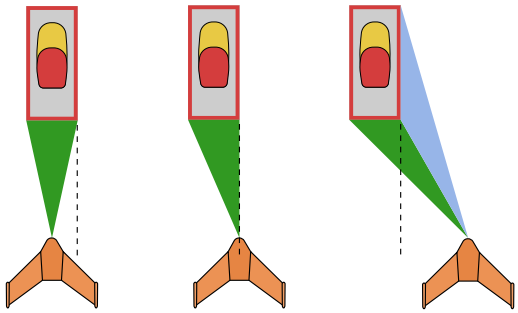


Figure 5.9: The left case shows the drone aligned with the boat's center-line. The middle case shows the drone misaligned by half of the boat's width. The right case shows a greater misalignment which causes the side of the boat to be visible – this increases the bounding box width of the boat in the cameras view.

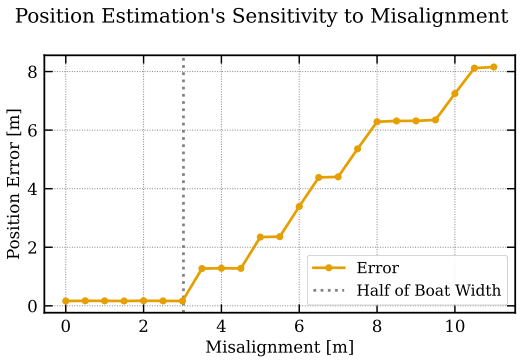


Figure 5.10: Position error's sensitivity to the drone being misaligned behind the boat. The camera is positioned 50 m behind the boat and then offset perpendicularly to the boat's center-line by the misalignment.

5 degrees (to leverage the flat region in Figure 5.10). The reduction of the mean position error when the distance was between 10 and 100 meters and ψ between 0 and 18 degrees was 73.11% (from 6.902 m to 1.856 m). A drawback was that for very short distances (10 meters), the compensation increased the error.

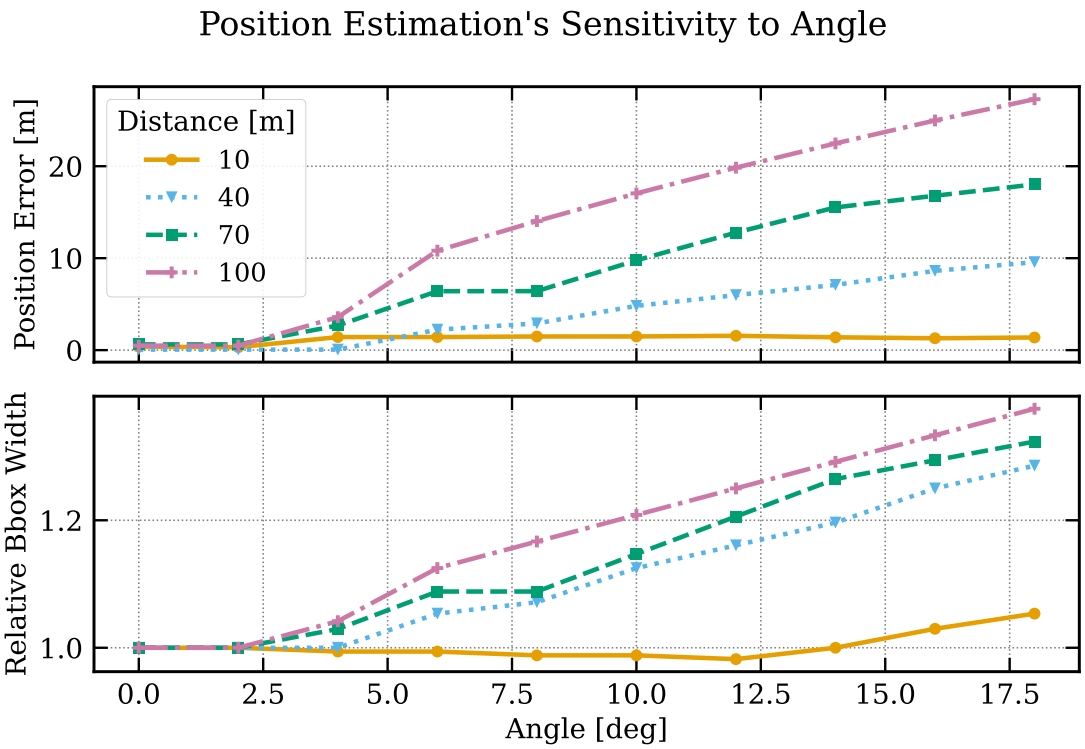
$$w_{\text{proj}} = w_W \cos(\psi) + l_W \sin(\psi) \tag{5.4}$$

$$w^*_{\text{proj}} = w_W + 0.4 \sin(\psi) \tag{5.5}$$

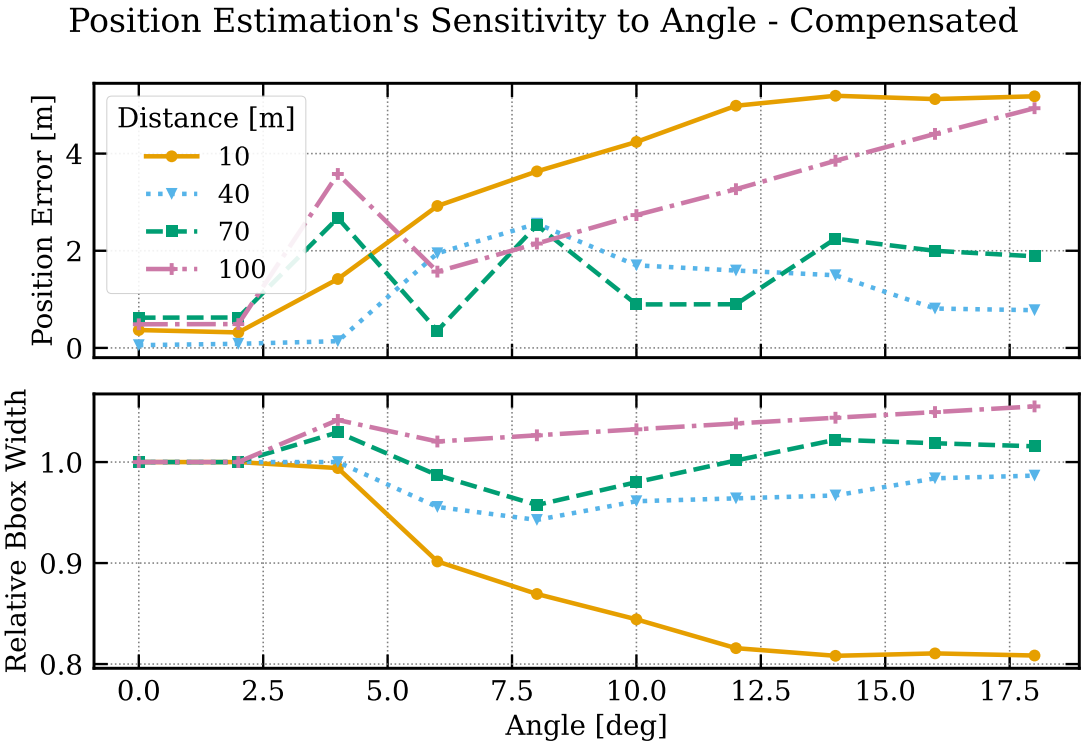
5.3.3 Comparison of the Altitude- and Width-based Method

We compared the two methods for calculating depth and the positional error is presented in Figure 5.12. Notably, the error decreases faster with distance for the width-based method compared to the altitude-based method. We stated in Section 5.3.2 that the width-based method was viable at 60 meters with no perturbations or 55 meters with the observed, 0.88 pixel perturbation. Based on Figure 5.12, the width-based method was also more precise for greater distances when the perturbations were below 1 pixel and the altitude error was larger than 1 meter.

One weakness of the width-based method was its quasi-periodic behavior (recall Section 5.3.2) – the altitude-based method could produce a more stable position estimation, depending on the altitude error. As mentioned in Section 1.1, we were unsuccessful in obtaining the expected altitude error but if it is found to be relatively constant, the estimated position will be more stable compared to the width-based method. Furthermore, if the error's magnitude is known as well, it is possible to compensate for it with a constant bias in the altitude measurement.



(a)



(b)

Figure 5.11: (a) Position estimation's sensitivity to a misalignment and the apparent bounding box width relative to when there is no misalignment. (b) Compensation for misalignment by projecting the boat's physical width as: $w^*_{\text{proj}} = w_W + 0.4\sin(\psi)$.

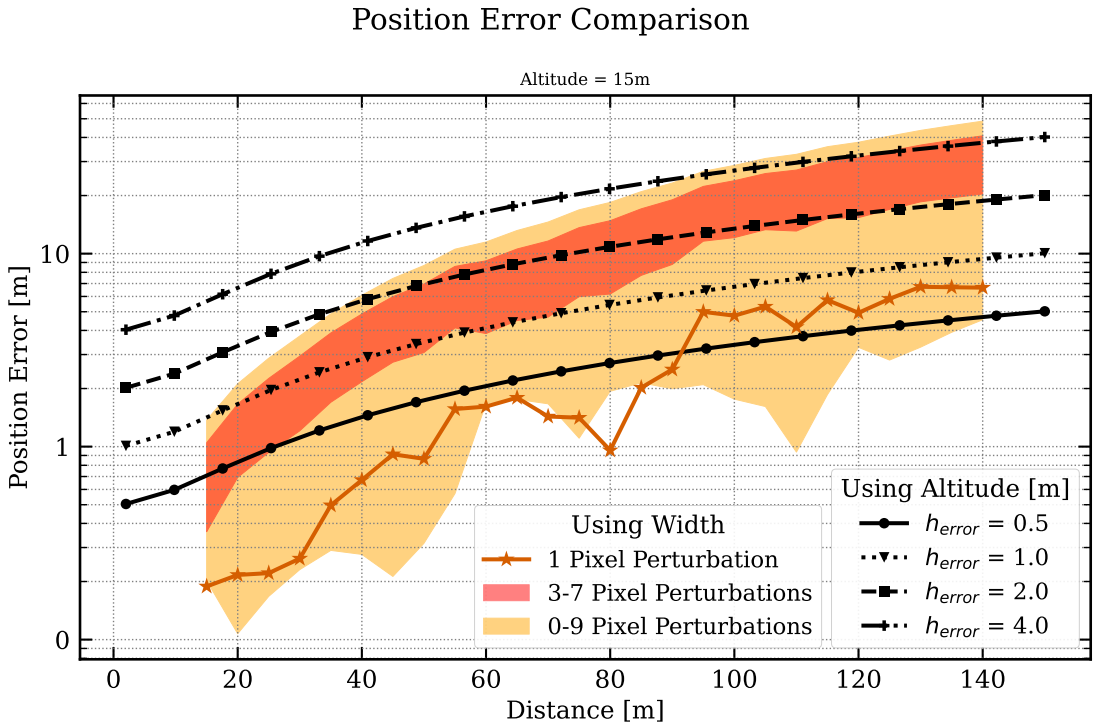


Figure 5.12: Position error comparison between using altitude and width for obtaining depth. To dampen the quasi-periodic errors from the width-based method (recall Section 5.3.2), we used a rolling average with a window size of 3 data points.

6
System Latency

In this chapter, the system described in Chapter 2 is evaluated on its end-to-end latency for multiple image resolutions.

6.1 Latency Analysis

We evaluated the system's end-to-end latency using aerial drone footage of an SSRS boat captured by a DJI drone. The video was resized to three resolutions that correspond with the drone camera's possible streaming resolutions. Tables 6.1 to 6.3 and Figure 6.1 show the performance from this test.

Table 6.1: System latency with resolution 1080p

(1080p)	Detection (ms)	Tracking (ms)	Localization (ms)	System (ms)
Min	172.88	0.57	0.72	174.17
Max	188.53	6.46	7.51	202.49
Mean	**178.72**	**1.84**	**3.10**	**183.65**

Table 6.2: System latency with resolution 720p.

(720p)	Detection (ms)	Tracking (ms)	Localization (ms)	System (ms)
Min	169.68	0.59	2.39	172.66
Max	177.89	3.28	8.75	189.93
Mean	**172.92**	**0.99**	**2.87**	**176.78**

Table 6.3: System latency with resolution 480p.

(480p)	Detection (ms)	Tracking (ms)	Localization (ms)	System (ms)
Min	152.97	0.57	1.21	154.75
Max	182.97	8.00	13.76	204.73
Mean	**159.44**	**1.28**	**6.59**	**167.31**

By analyzing Tables 6.1 to 6.3 and Figure 6.1, we noticed that inference time for the detection module decreases with resolution. However, the tracking and localization modules do not follow the same pattern. Their inconsistency can depend on several factors but the conclusion is that the differences are negligible compared to

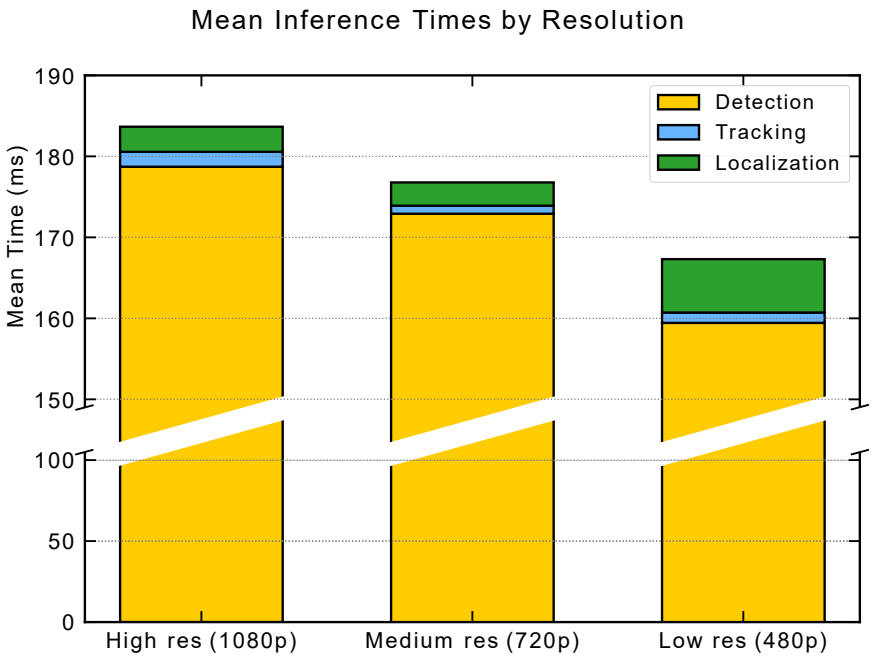


Figure 6.1: System Latency comparison for three resolutions.

the end-to-end latency.

Also worth noting is that latency is not significantly impacted by higher resolutions. We therefore propose using the highest resolution by default to maximize clarity in the video-feed that is being sent back to SSRS and used in the rescue operation.

7
Discussion & Conclusion

This chapter summarizes the system's design, performance and contributions, reflecting on its suitability for Svenska Sjöräddningssällskapet's (SSRS) semi-cooperative drone landing application.

This thesis documents the development and evaluation of a lightweight, modular system for near real-time detection and localization of boats at sea using aerial drone footage. The system is constrained to operate on a single compute thread of an RPi5, aiming to support autonomous drone landings for SSRS.

The implemented system architecture features three containerized core modules: detection, tracking, and localization, which function as microservices and communicate via a custom Redis-based utility class. This approach achieved a mean end-to-end latency of 183 milliseconds.

For detection, a YOLO11n model was selected and further optimized via transfer learning, then deployed with the NCNN framework to maximize inference speed on ARM-based hardware. During the project period, we were not able to collect a dataset large enough to train a robust model for a wide range of weather and lighting scenarios. Expanding the dataset is necessary for improving the reliability of the detection module.

For tracking, a lightweight algorithmic approach was employed instead of an AI-based solution, this to reduce computational cost as well as to avoid issues with unstable ID assignments that was found to emerge with a lower frame rate. The tracker proved reliable in scenarios with a small number of objects and had a negligible impact on the end-to-end latency. Given that the intended application involves sparse object environments, this tracker offers a more time efficient alternative to heavier, AI-based methods.

A significant portion of this research focused on the localization module, where two methods for estimating the depth to the target boat were implemented and compared: one leveraging the drone's altitude and the other using the known physical width of the boat. The investigation revealed that the accuracy of the altitude-based method was sensitive to errors in the drone's altitude measurements, with these errors becoming more pronounced at shallower glide slope angles. Conversely, the width-based method demonstrated superior performance at closer distances thanks

to being independent of inaccurate altitude data, but was found to be sensitive to inaccuracies in the boat's bounding box width – an issue that became more pronounced at greater distances or when there was a misalignment between the drone and the boat's centerline. Despite an average 0.88 pixel error in the boat's bounding box width by the object detector, the width-based localization approach was found to be more accurate than the altitude-based method (assuming a 1-meter error in altitude data) even at greater distances. Additionally, we proposed a simple yet effective compensation technique to mitigate errors caused by drone-boat misalignment, which was shown to reduce the mean position error by 73.11% for misalignment angles between 0 and 18 degrees.

We acknowledge that since our data comes solely from simulated environments, the results might not be completely applicable to the real world. However, they serve as tools to estimate the expected positional accuracy when one has the relevant measurements of sensory errors.

8
Future Work

Based on our findings, we propose several areas of possible improvement and future research topics. They are listed below.

The drone's estimated altitude error should be measured on real targets to provide insight in how accurate the system can estimate relative distances. If the error is found to be too significant, other methods for altitude acquisition such as outfitting the drone with a radar or LiDAR should be investigated.

Alternatively, the localization objective can be discarded in favor of maintaining an optimal glide slope angle with respect to the boat. By omitting the depth calculation, the methods described in Sections 5.3.1 and 5.3.2 become redundant. However, the distance to the boat would be unknown and can not be used as an input parameter for a landing controller. A hybrid approach that utilizes our proposed methods and enhances the accuracy with the glide slope method might be optimal to address this downside.

Furthermore, AprilTags could be used for localization, which would greatly enhance the positional estimation precision. While SSRS stated that the use of AprilTags on the boat contradicts the "semi-cooperative" nature, we strongly believe that fitting a tag on the boats is an inexpensive solution that can provide a more accurate reference width robust to misalignment and perspective distortion. Additionally, the landing zone can be specified and altered between different types of boats by adjusting where the tag is placed. However, the solution is only applicable at distances where the AprilTag is completely visible for the detection module – complementary methods for position estimations at greater distances may therefore be suitable.

Different types of boats might have vastly different locations that is optimal for landing. It would be beneficial to study more advanced AI or feature extraction techniques to find a safe area on the boat to land on. This would involve avoiding humans and making sure the landing surface is flat to prevent the drone from sliding into the water.

Bibliography

[1] Svenska Sjöräddningssällskapet, *Forskning och utveckling: Drönare*, Accessed: 2025-02-05, 2025. [Online]. Available: https://www.sjoraddning.se/forskning-och-utveckling/dronare.

[2] Transportstyrelsen, *Transportstyrelsens föreskrifter och allmänna råd om navigationsutrustning och navigationssystem på fartyg (TSFS 2010:12)*, Accessed: 2025-02-05, 2010. [Online]. Available: https://www.transportstyrelsen.se/tsfs/TSFS%202010_12.pdf.

[3] H. Olsson, *Redis communication module*, 2025. [Online]. Available: https://github.com/SSRS-Innovation/redis-communication.

[4] L. Kristiansson and H. Olsson, *Detection module*, 2025. [Online]. Available: https://github.com/SSRS-Innovation/object-detector.

[5] S. Zaidi, A. M. Ansari, H. Kim, and N. Barnes, "A survey of modern deep learning based object detection models," *arXiv preprint arXiv:2104.11892*, 2021. [Online]. Available: https://arxiv.org/abs/2104.11892.

[6] V. Labs, *Yolo object detection explained*, https://www.v7labs.com/blog/yolo-object-detection, Accessed: 2025-04-23, 2023.

[7] Ultralytics, *Ultralytics yolo quickstart guide*, Accessed: 2025-04-23, 2025. [Online]. Available: https://docs.ultralytics.com/quickstart/.

[8] Y. Zhao, W. Lv, S. Xu, *et al.*, *Detrs beat yolos on real-time object detection*, 2024. arXiv: 2304.08069 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2304.08069.

[9] J. S. Marques, A. Bernardino, G. Cruz, and M. Bento, "An algorithm for the detection of vessels in aerial images," in *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2014, pp. 295–300. DOI: 10.1109/AVSS.2014.6918684.

[10] M. AI, *Pytorch documentation*, Accessed: 2025-05-09, 2025. [Online]. Available: https://pytorch.org/docs/stable/.

[11] O. Community, *Onnx: Open neural network exchange*, Accessed: 2025-05-09, 2025. [Online]. Available: https://onnx.ai/onnx/intro/.

[12] T. A. Lab, *Ncnn documentation*, Accessed: 2025-05-09, 2025. [Online]. Available: https://ncnn.readthedocs.io/en/latest/.

[13] L. Kristiansson and H. Olsson, *Tracking module*, 2025. [Online]. Available: https://github.com/SSRS-Innovation/landing-target-tracker.

[14] B. Sahbani and W. Adiprawita, "Kalman filter and iterative-hungarian algorithm implementation for low complexity point tracking as part of fast multiple object tracking system," in *2016 6th International Conference on System Engi-*

neering and Technology (ICSET), 2016, pp. 109–115. DOI: 10.1109/ICSEngT.2016.7849633.

[15] D. F. Crouse, "On implementing 2d rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016. DOI: 10.1109/TAES.2016.140952.

[16] L. Kristiansson and H. Olsson, *Localization module*, 2025. [Online]. Available: https://github.com/SSRS-Innovation/coordinate-conversion.

[17] J. Redding, T. McLain, R. Beard, and C. Taylor, "Vision-based target localization from a fixed-wing miniature air vehicle," in *2006 American Control Conference*, 2006, 6 pp.-. DOI: 10.1109/ACC.2006.1657153.

[18] G. E. Contributors, *Godot engine*. [Online]. Available: https://godotengine.org.

[19] Docker Inc., *What is docker?* Accessed: 2025-04-23, 2024. [Online]. Available: https://docs.docker.com/get-started/docker-overview/.

[20] S. Sanfilippo, *Redis: In-memory data structure store*, Accessed: 2025-04-23, 2024. [Online]. Available: https://redis.io.

[21] R. Padilla, S. L. Netto, and E. A. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, IEEE, 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130. [Online]. Available: https://www.researchgate.net/publication/343194514.

[22] P. Sturm, "Pinhole camera model," in *Computer Vision: A Reference Guide*, K. Ikeuchi, Ed. Cham: Springer International Publishing, 2021, pp. 983–986, ISBN: 978-3-030-63416-2. DOI: 10.1007/978-3-030-63416-2_472. [Online]. Available: https://doi.org/10.1007/978-3-030-63416-2_472.

[23] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: https://github.com/opencv/opencv.

[24] OpenCV, *Opencv camera calibration*, Accessed: 2025-01-30, 2025. [Online]. Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

A

Important Concepts

This appendix presents brief explanations of key technical concepts used in the report. It is intended to provide readers with helpful context for understanding parts of the system and that is not explained in full within the main chapters.

A.1 Docker

Docker is a platform for developing, shipping, and running applications using containerization. A container is a lightweight, standalone, and executable software package that includes everything needed to run an application - such as code, runtime, and libraries. This allows for high portability and ensures consistent behavior across different platforms and machines [19].

In this thesis, Docker is used to separate functionality to allow for a more manageable and extendable architecture. Each module is packaged with its dependencies, providing an ease-of-use deployment and avoids compatibility issues.

A.2 Redis

Redis (REmote DIctionary Server) is an in-memory data structure store that functions as a high-performance, key-value database, cache, and message broker. A notable feature is its ability to support inter-process communication through publish/subscribe messaging and streaming APIs. These features allow Redis to act as a central communication server. Messages are emitted on a specified channel and received by all clients that are actively listening on said channel. Streams are log-like data structures stored on the server that can be retrieved by a client when desired [20].

In this thesis, Redis messages and streams are used to send data between Docker containers as a JSON string.

B

Score Metrics

Modern object detectors typically output a list of predictions in the form of bounding boxes, class labels and confidence scores. While the predictions are easily visualized for qualitative evaluation, the models must show robustness and are therefore quantitatively evaluated. This is not as descriptive in visual form and therefore quantitative score metrics such Average Precision (AP) and mean Average Precision (mAP) can be utilized. These metrics have become the most widely used for benchmarking, especially in challenges and competitions such as PASCAL VOC and COCO [21]. The survey further describes that they rely on evaluation components such as precision, recall and Intersection over Union (IoU) to quantify how well the model predicts according to the ground truth.

In order to calculate precision and recall, the result from a predicted frame is analyzed and bounding box data is measured in the following form:

- True Positive (TP): A correct detection of a ground truth bounding box.
- False Positive (FP): An incorrect detection of a predicted bounding box that is not ground truth.
- False Negative (FN): An undetected ground truth bounding box.

To evaluate what each predicted bounding box is classified as, a common method is to use the IoU (Figure B.1) which measures the overlap between a predicted bounding box and a ground truth bounding box:

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}, \tag{B.1}$$

where B_p denotes the predicted bounding box and B_{gt} denotes the ground truth bounding box. The IoU represents the ratio of shared area to the total combined area of the two close-by boxes. If the IoU score is above a certain threshold $t \in (0, 1)$, the prediction is considered as TP, otherwise it is considered a FP. Any remaining, unmatched ground truth object is considered as FN.

Further, to represent the boxes in a quantitative form, precision and recall is calculated as following:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{B.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{B.3}$$

where precision represents how accurately the predicted boxes match true objects, and recall measures how well the model detects all ground truth objects.

The precision and recall scores are iteratively calculated for a confidence interval ranging from 0 to 1. The scores are then plotted in a precision $\times$ recall curve (PR curve) to be further analyzed. The resulting curve shows the trade-off between precision and recall for a varying confidence threshold. When the threshold is close to 0, the model would have a high recall since it will detect more objects (FN is low), thus finding all ground truths. The precision will consequently be very low since the FP score will be high. Conversely, for a confidence threshold very close to 1, the precision will be high and the recall low, see figure B.2.

The average precision (AP) is then calculated as the area under the graph (AUC), and provides a single-value summary of the model's performance across all thresholds. Lastly, mean average precision (mAP), is the mean AP across all predicted classes. It is calculated as

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i, \tag{B.4}$$

with AP_i representing the AP for class i and N the total number of classes. The mAP and AP score for single class detection problems – such as for this thesis – would yield the same score.

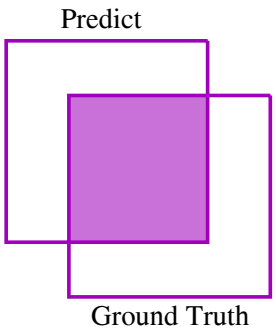


Figure B.1: Illustration of IoU. The ratio of overlapping area is equivalent to the intersection over union (IoU).

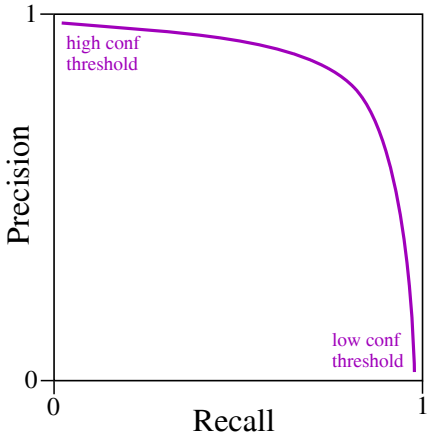


Figure B.2: Example of a PR curve. Red curve represents the PR curve consisting of precision-recall score pairs for a varying confidence threshold. A near perfect object detection model would yield a curve with an area under the curve (AUC) being close to 1.

C

Coordinate Projection with the Pinhole Camera Model

Given a screen-space coordinate, it can be converted to the camera's reference frame using the pinhole camera model [22]. It requires an intrinsic transformation matrix and the object depth d. The intrinsic matrix accounts for the camera lens's focal length and distortion and has the form:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{C.1}$$

where f_x and f_y are the focal lengths in pixels and c_x and c_y are the origin shift. Usually the origin in an image is in the top-left corner and thus needs to be shifted to the center by half of the image width and height. The computer vision library OpenCV [23] provides tools for estimating this matrix based on a series of photos of a checkerboard pattern captured by the camera being calibrated. The instructions for the calibration procedure can be found in the OpenCV manual [24]. Since the image is 2-dimensional, the resulting vector is in image plane coordinates ($Z_c = 1$) and has an unknown scale referred to as the depth (d). This is the distance between the camera and the object along the camera's Z-axis.

An extrinsic matrix is a combined rotation and translation matrix that can be used to convert from camera-space to world-space:

$$T_{cw} = \begin{bmatrix} R & -Rt \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{C.2}$$

where R is a 3×3 rotation matrix representing the camera's rotation and t is a 3×1 translation matrix representing its position.

To convert a coordinate in image-space (u, v) to world-space (X_w, Y_w, Z_w), use the matrices as follows:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = T_{cw} \cdot d \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{C.3}$$

Note that homogeneous coordinates are used.

D

Additional Figures

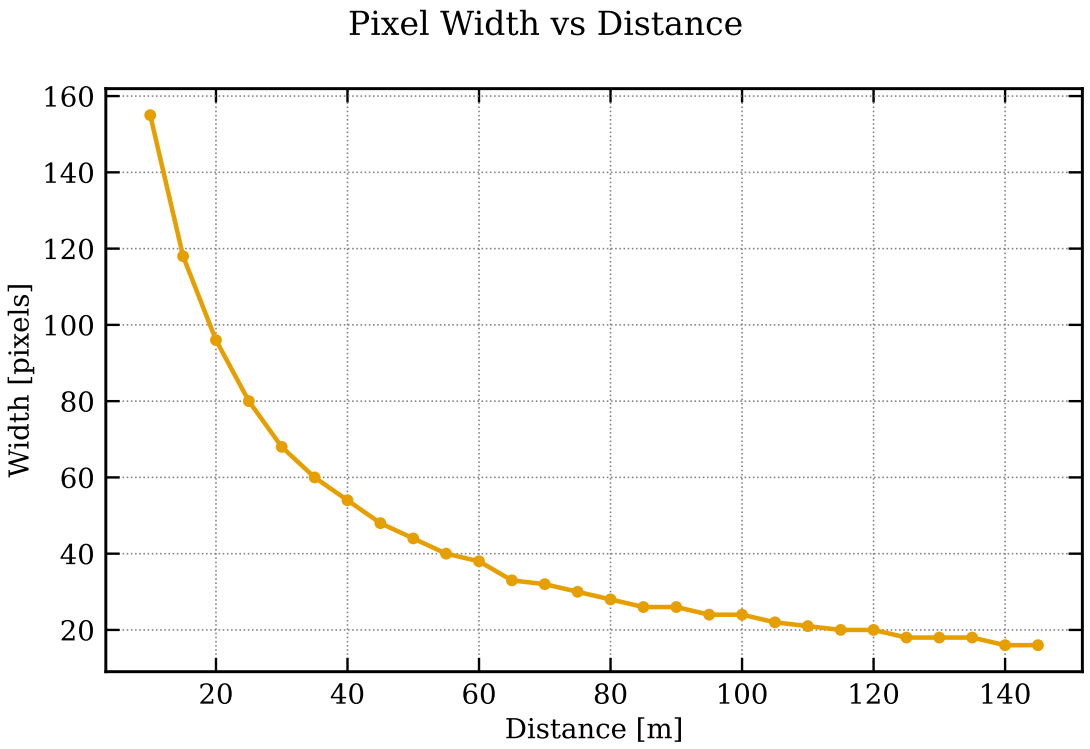


Figure D.1: The apparent width of the boat's bounding box vs the camera's distance from the boat.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY