Internship Report

Vision-Based Boat Localization Using Stereo Cameras for Remote Situational Awareness

Solène Boudot
ENSTA

Internship Institution
Danmarks Tekniske Universitet

Internship Period
05/05/2025 – 05/09/2025

ENSTA Supervisor
Luc Jaulin

DTU Supervisor
Morgan Louédec, Roberto Galeazzi

Abstract

This internship within the CREA research group at DTU in Copenhagen aimed to develop a boat localization framework using stereo camera vision. Since my colleagues had previously built the measurement setup and collected the data, my work initially focused on designing an algorithm to estimate boat positions and demonstrate the reliability of such a method. The project began with a simple goal: estimating the position of a single boat and comparing it with GPS data using set-based computations performed by my supervisor, which led to the publication of a research paper that I had the honor to co-author. I then improved the measurement setup and, finally, enhanced my own algorithm to make it more robust and sophisticated, notably by enabling it to track multiple boats simultaneously. This internship allowed me to discover and deepen my understanding of several fields such as machine learning, computer vision, Kalman filtering, 3D vision, and experimental design. Ultimately, I developed an operational framework for boat localization, which may serve as a foundation for future research at DTU aiming to further advance stereo-vision-based localization.

Résumé

Ce stage au sein du groupe de recherche CREA du DTU à Copenhague avait d'abord pour but de concevoir un framework de localisation de bateaux à l'aide de vision par caméras stéréo. Mes collègues ayant précédemment construit le dispositif de mesures et fait des collectes de données, mon travail fut d'abord de créer un algorithme pour estimer la position de bateaux, de sorte à montrer la fiabilité d'une telle méthode. Il fallait commencer simplement : réussir à estimer la position d'un seul bateau, puis la comparer avec des données GPS grâce à des calculs ensemblistes (réalisés par mon superviseur), ce qui a fait l'objet d'un papier de recherche que j'ai eu l'honneur de co-écrire. J'ai ensuite pu améliorer le dispositif de mesures, puis, dans un dernier temps, améliorer mon propre algorithme pour le rendre plus robuste, et plus complexe, notamment en le rendant capable de suivre plusieurs bateaux à la fois. Ce stage m'aura permis de découvrir ou de mieux comprendre de nombreux domaines tels que le machine learning et les outils de vision qui y sont associés, le filtrage de Kalman, la vision 3D, la conception... Finalement, j'ai pu mettre au point un framework opérationnel pour localiser des bateaux, qui pourrait constituer la base de travaux futurs au DTU pour approfondir la recherche sur la localisation par vision stéréo.

Contents

1 Introduction

1.1 Presentation of the place of work

The Technical University of Denmark (DTU) is a leading technical university located in Lyngby, near Copenhagen. Founded in 1829, DTU is renowned for its strong engineering, natural sciences, and technology research, and it collaborates broadly with industry and research institutions around the world. Within DTU, the Department of Electrical and Photonics Engineering—also known as DTU Electro—is dedicated to advancing knowledge in light- and electronics-based technologies. It carries out research ranging from photonics, optical sensors, quantum and nanophotonics, power electronics, robotics, to sustainable energy systems, all while pursuing high-level education and close collaboration with industry. The research groups I joined in DTU Electro is the Control, Robotics and Embodied AI (CREA) group. CREA focuses on developing next-generation automation and robotics systems by combining expertise in mechatronics, control theory, neuromorphic computing, artificial intelligence, and system supervision. Their goal is to build intelligent engineered systems that preserve human agency, tackling challenges in domains such as energy, health, security, logistics, and autonomous systems deployment. I was tutored by Roberto Galeazzi, associate professor, and worked more closely with Morgan Louédec, a post-doctorate researcher.

1.2 Problematisation and mission objective

During this internship, I worked on the development of a vision-based localization framework for Maritime Autonomous Surface Ships (MASS). For safety and regulatory reasons, a human pilot must always be able to remotely control a ship in case of emergency. To do so, the pilot needs reliable information about the ship's state and its surroundings in order to assess collision or grounding risks. Since the pilot is not physically onboard, their situational awareness depends entirely on the data sent by the ship's navigation sensors.

At DTU, the research group I joined focuses on improving this situational awareness by combining multiple sensors through fusion methods. The goal is to reconstruct the ship's environment into a semantic map and to estimate the quality of the fused data using uncertainty propagation and matching scores. Previous studies have shown that combining traditional SOLAS navigation sensors with cameras and lidars provides a robust perception of the surroundings, as long as the sensors and transmitted data are not corrupted [1].

However, in real-world conditions, data can be degraded or even falsified, which reduces the pilot's ability to correctly interpret the situation. To make the system more resilient, additional trusted data sources can be used to validate the information received from the ship. In this context, DTU has developed a sensor platform equipped with an AIS receiver, an X-band radar, RGB and infrared cameras, an IMU, and several weather sensors. Another land-based platform with cameras was also deployed in Copenhagen harbour, providing additional ground-truth data that can be trusted even more.

My mission was to work on these land-based cameras and develop a framework capable of localizing boats in various conditions, including degraded data or cyber-attacks. The data collected by DTU during previous measurement campaigns with Copenhagen's Harbour Buses served as the basis for my work.

2 Activities undertaken during the internship

Several activities were carried out during this internship. The first task was providing a framework that could be used for estimating the position of boats on images my coworkers had collected during a first data collection campaign. The objective was to compute the uncertainties of the stereo position estimation method for a research paper. Afterward, I spent about a week and a half improving the enclosure of said stereo system: the electronics used for this project were already in a box, but it couldn't be used outside if it was rainy, which happens a lot in Denmark. I was therefore tasked with making this enclosure and its camera connections water-resistant. Finally, I improved the stereo position estimation code I had already developed: making it more robust and easy to use for further experiments.

In the following sections, an overview of the physical system and its modifications for water resistance will be presented, then the framework will be explained: its first implementation for the research paper, and the improved version for further use.

2.1 Waterproof enclosure for the land platform sensors

2.1.1 How the system works

Stereo localization relies on using two cameras placed apart, each offering a slightly different viewpoint of the target. The boat will therefore not be in the same place on each camera (it will appear more to the left in the right image, and vice versa). Thus, by computing the distance between the boat in each image (which is called the disparity), and by knowing the focal length of the cameras, the distance from the boat to the cameras' vertical plane can be easily computed.
To achieve this, a land platform was deployed in the harbour of Copenhagen (see Figure 2). The land platform consists of three cameras mounted on a rig. There are two RGB cameras at each extremity, separated by 1.98 m, and also one infrared camera in the middle that can be used for further information, especially in case of bad weather.

A Raspberry Pi 4 is used as the motherboard for this system, with all the useful scripts for recording data. The Raspberry Pi and cameras are connected via an Ethernet hub. Lastly, there is also a power supply for all this system.
The land platform will then record several dockings of a Harbour Bus (see Figure 1), which also carries the sensor platform, including a GPS that will be used for checking if the position was correctly estimated. In the end, the main objective will be to verify that the GPS trajectory is not falsified on any boat using the land-based measurements.

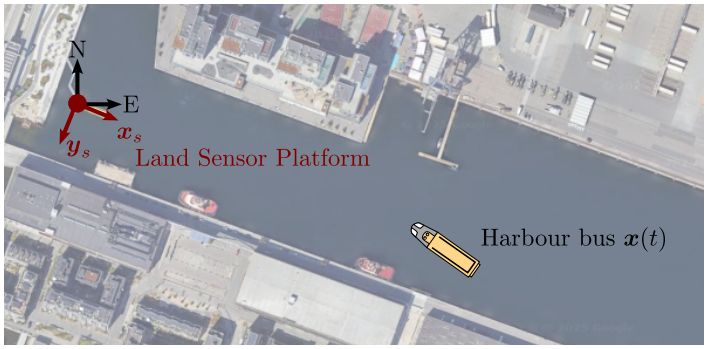


Figure 1: Map of Copenhagen harbour showing sensor platform location

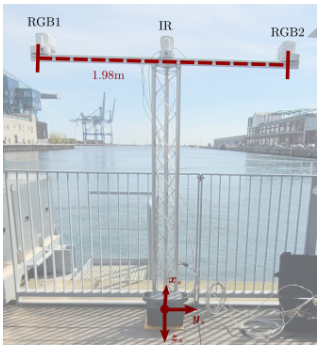


Figure 2: Land Sensor Platform

2.1.2 Making the system water-resistant

Initially, the Raspberry Pi, the SSD, the Ethernet hub and the power gland were put in a box that remained open during measurements, and the cameras also had open holes for the Ethernet cables to pass through (see Figures 3 and 4). However, in case of bad weather, the components would most likely be damaged. Therefore, the box was altered to be kept closed during the measurements, and all connections were made water resistant. Since the goal was only to protect against rain, an IP65 rating was sufficient.

Figure 3: Box before modifications

Figure 4: Camera before modifications

The first task was making sure that the box could close completely. Before, the Ethernet hub and power supply were mounted in such a way that they prevented the box from closing. I therefore designed a 3D-printed mount that could be attached to the box rail and included a secondary rail to hold the original components (see Figures 5 and 6). That way, the orientation of the pieces could easily be shifted without altering the box or the pieces my coworkers crafted before.

Figure 5: Support piece that was added

Figure 6: Ethernet hub mounted on the new support piece, now fitting the box

Then, a support was designed to hold tightly the SSD, which was running freely in the box before. This was also achieved by creating 3D-printed parts that could simply be added to the original design, without altering it (see Figure 7)

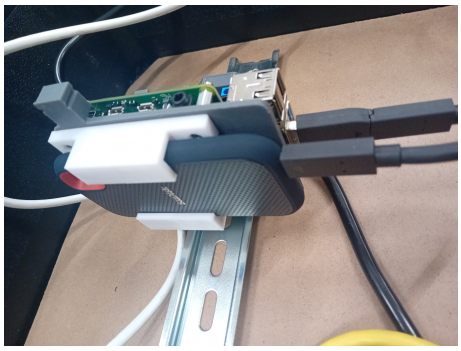


Figure 7: SSD held by the new support

The most important part was making all the Ethernet cables and the power supply cable pass through the box, with a waterproof connection. To achieve this, I drilled holes in the box and installed cable glands (see Figure 8). Relatively large ones had to be used for the RJ45 connectors of the Ethernet cables to pass through without having to cut them, so heat shrink tubing was added to the cables in order to make them larger, so that the glands would hold them firmly enough. However, the power supply connector was too large to pass through. Therefore, I cut it and connected each wire to a rail-mounted terminal block, for a secure and steady connection (see Figure 9).

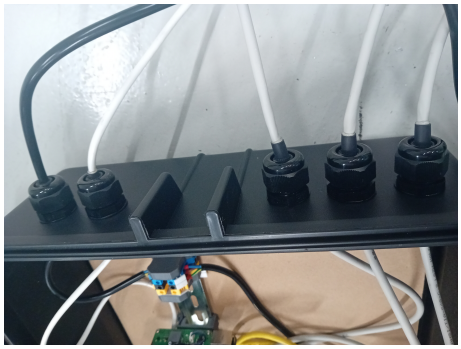


Figure 8: Cable glands installed on the box

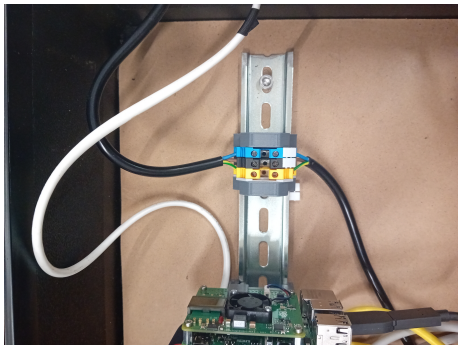


Figure 9: Power supply connection on the terminal block

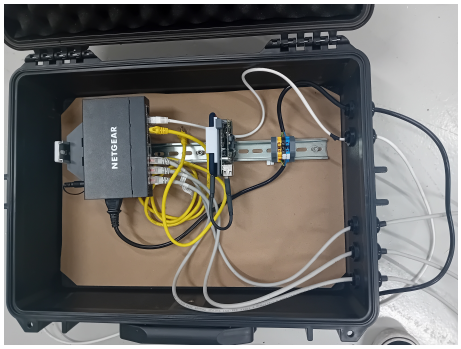


Figure 10: Final design of the box

Once this was done, the last part consisted in sealing the camera cable entries. Since we wanted to be able to plug and unplug the Ethernet cables on the cameras' side for simplicity, it was necessary to have a plug that would let the cables pass through, but could be easily mountable and unmountable to disconnect or connect the wires when needed, all while being water resistant. To perform this, cable glands were a simple solution; however, none of the available cable glands fitted the camera holes. Therefore, I 3D-printed adapters that could connect the cameras on one side and the cable glands on the other, ensuring watertightness with rubber seals (see Figures 11 and 12). Now, the user only needs to unscrew the cable gland to access the Ethernet plug (see Figure 13).

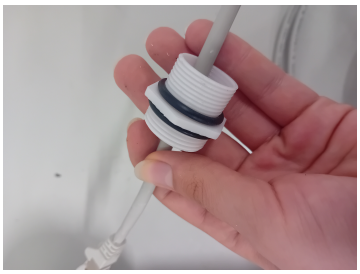


Figure 11: Unmounted adapter

Figure 12: Mounted adapter

Figure 13: Watertight connection of the camera

2.2 First algorithms for boat localization

In order to implement the land platform sensor for cross-validation in future projects, it is first important to demonstrate that this method can yield promising results. This is what we aimed to show in a paper written by my supervisor and me, and to do this we had to create the framework for estimating the position of the boat with the stereo cameras, and compare it with the GPS position of the boat (the ground truth) to detect and quantify the errors using interval analysis methods. I was responsible for developing the framework.

The position of the boat is estimated by measuring the gap (the disparity) between the boat in the left and right RGB images. The main steps of the process are described below.

2.2.1 Data collection

First, the setup was installed in Copenhagen harbour, and recorded images from the three cameras in a ROSbag file. Then, I coded an algorithm that would play the ROSbag and process it simultaneously, in order to save all images in a .png format. Here, the challenge was mostly to name the images: to have something that could be used later on to pair the images, it was necessary to get the precise timestamp of each image and name them accordingly. I therefore used PyTesseract for performing Optical Character Recognition (OCR) on the images (they all had the timestamp on the top left corner, see Figure 14), and automatically rename the 1600 images that obtained during the data collection campaign. A few mislabelled images (around 20) had to be manually corrected.

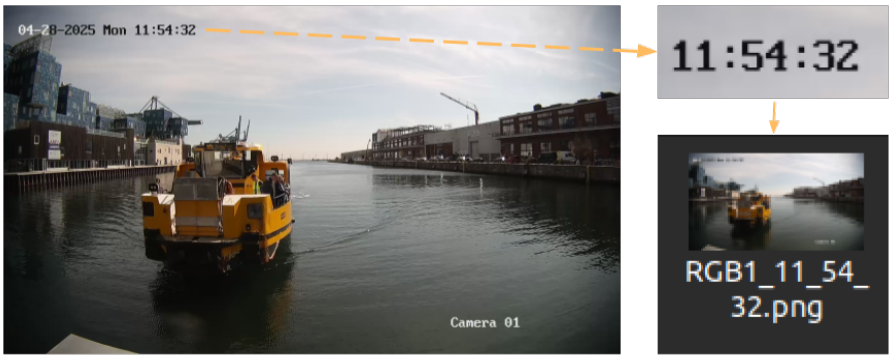


Figure 14: Renaming process of an image with OCR

2.2.2 Calibration

The task that took the most time was the calibration of the cameras. Calibrating the cameras refers to, first, estimating the intrinsic parameters of each camera (focal length, optical center, and distortion coefficients), known as monocular calibration, and then the extrinsic parameters for the whole system (translation and rotation matrices between both cameras), known as stereo calibration [2]. Then, once those parameters are known, rectification maps can be easily computed. Those maps will rectify the images, or in other words, undistort them and align their epipolar lines.

To find the parameters, I decided to go with the standard (and most effective) chessboard method, which consists in capturing multiple images of a chessboard pattern, a geometrically regular and easily recognizable object, and measuring the different parameters on the images thanks to the chessboard corners, knowing its real world size.
The chessboard detections were performed with OpenCV.

Then, I wished to continue the calibration with OpenCV, which is easy to use through several key functions. However, whenever the calibration functions were applied, the results were often incoherent and changed a lot depending on the calibration dataset. I found the origin of one problem: OpenCV takes into account the orientation of the chessboard, and if there is an even∗even or odd∗odd number of rows and columns, it sometimes happens that it reverts the order of the corners it detects... thus skewing the results. But even when the chessboard was changed for an odd*even one, inconsistencies and big reprojection errors (the error that indicates whether a calibration is valid) remained. This was likely because OpenCV's calibration tools are generally not well suited for long-range cameras [3].

This is why I decided to switch to MrCal [4], an open-source calibration framework, to perform the calibration. Mrcal is much more suited for long-range stereo applications [3]. It took quite some time to find how to use it correctly, but this process was made easier thanks to the thorough documentation (which also highlighted the importance of well-written documentation). The corner detection was still done with OpenCV, but the computation of the intrinsic and extrinsic parameters was obtained through MrCal, which finally gave decent results, with low reprojection errors.

Now, the last part was to use those parameters to undistort and rectify the images. This was done with OpenCV's functions too. Some difficulties were initially encountered while rectifying the images, first because I didn't read the documentation correctly, and then when this was solved, because the cameras had moved a bit between the data collection campaign and the day the calibration images were collected. But this could easily be fixed: knowing that in the background, the objects were supposed to be at the same place on both images (zero disparity), the rotation matrix was slightly adjusted manually so the images would align correctly when visualizing. In the end, we could get well calibrated images, and accurate intrinsic parameters, which proved useful for the subsequent stages of the project.

2.2.3 Images pairing

Once the images were calibrated, it was necessary to pair them for stereo processing, i.e. determining which images correspond to each other based on their timestamps. The images were taken approximately every 2 seconds, but sometimes data acquisition gaps occurred, which made it necessary to algorithmically pair the images (it was not possible to simply take the images with the same timestamp, as they were often time-shifted by 1 second). I analyzed the time gaps by plotting the data acquisition timeline (see Figure 15), and then decided to proceed as follows: for each image in the left folder, an image with the same timestamp is associated or, if it doesn't exist, a timestamp within ± 1 second. If that does exist, their names go in the pairs list. This ensured that paired images were captured within a maximum gap of one second, and the big data reception

gaps are not a problem (even though this means some images will be unused).

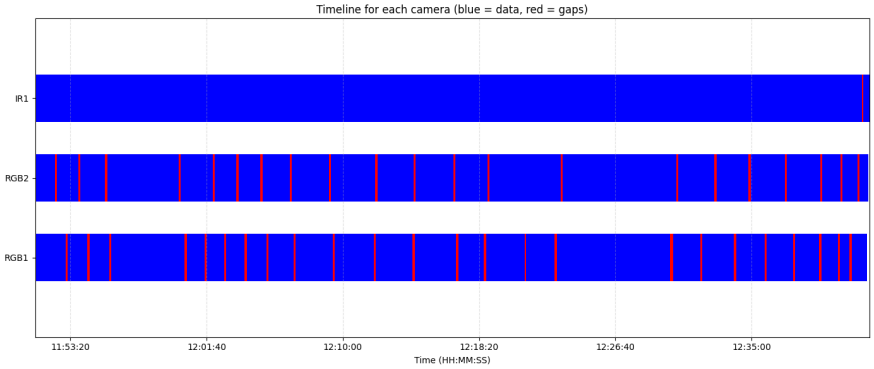


Figure 15: Data reception timeline for each camera

2.2.4 Boat detection

To detect the boat in the images, I first tried to use a color segmentation method, knowing the boat was bright yellow. This worked pretty well when the boat was close, but as soon as it moved away it became way less saturated and harder to segment. Moreover, such a method would have made the algorithm difficult to adapt to other situations (what if now, we need to track a blue boat ?). Therefore, this method was abandoned, and it was decided to try using YOLO instead, a deep-learning model for object detection.

I used YOLOv8 [5] [6] pre-trained on the COCO dataset, in order to find the bounding box of the boat on each picture (see Figure 16). A confidence threshold of 0.35 is used for boat detection (class 8 of the COCO dataset). The threshold was tuned to minimize false detections while maintaining sensitivity at longer ranges. Buildings were often misdetected as boats; they were therefore filtered out by discarding bounding boxes whose centers fell within a predefined building region, which was found through a visual inspection of the images (so only valid for this particular dataset). For simplicity, only the detection with the highest confidence score was considered. The boat is considered as detected on a pair of images if it is detected on at least one image of the pair. In general, the Harbour Bus began being detected when it was approximately in a range of 250m from the land-based sensor platform.

2.2.5 Feature matching

To compute the disparity between the boats, I decided to use a feature matching algorithm in the bounding box returned by YOLO. This consists in finding relevant features in each image, and then pairing together those corresponding to the same point. SIFT (Scale-Invariant Feature Transform) was used for this, precisely because it is scale-invariant, andfeatures need to be extracted on boats that are both close and far [7].

The feature matching was performed using a brute-force matcher (BFMatcher) with L2 distance on the descriptors. Matches were filtered using Lowe's ratio test with a threshold of 0.75, and only correspondences with a vertical disparity smaller than 15 pixels were retained to enforce epipolar consistency [7].

Moreover, to avoid false positives from building structures occasionally present at the edges of the region of interest returned by the YOLO detector, the matches corresponding

to buildings (i.e. whose features' positions belong to the buildings zone) were removed. This approach may remove valid matches when the boat overlaps with buildings, but it ensures that there are way less unwanted matches.

2.2.6 Computation of the disparity

The disparity d_i of each match was computed as the horizontal distance between the matched features. A more thorough filtering of outliers was also used: only disparities between d_{min}=9 px and d_{max}=760 px were retained, which corresponds to an expected object distance between approximately 3 and 250 meters. Disparities with high spread were excluded based on their standard deviation: a threshold of 130 pixels was used when the estimated distance was below 12 meters (i.e., near range; the standard deviation is quite high in this case since the vessel occupies a large portion of the image), and 40 pixels otherwise. Vessel detections with an insufficient number of valid disparity measurements (the threshold is empirically set to 5) were discarded. Finally, since some matches may belong to the background in the corners of the region of interest, an additional filter was applied: if the disparity distribution was noisy (i.e., standard deviation above a threshold of 25 px), the disparities likely originating from the background were removed. After filtering, there was a mean of 51.4 good matches over the images, corresponding to 71.6% inliers.

From the disparity $d_i > 0$ of each match, the estimated disparity $\boldsymbol{d}$ at time t is evaluated as the average disparity, such that

$$\boldsymbol{d} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{d}_i \tag{1}$$

2.2.7 Position estimation

From the position of the center of the detection box in the left image $\boldsymbol{p} = [u, v] \in \mathbb{Z}^2$, the optical center of the camera $\boldsymbol{c} = [C_x, C_y] \in \mathbb{Z}^2$ the average disparity $\boldsymbol{d}$, the baseline $b > 0$, and the focal length $f > 0$, the horizontal position of the ship in the land-based sensor frame is estimated as

$$\hat{\boldsymbol{x}}_{\text{stereo}} = \begin{bmatrix} \frac{f \cdot b}{d} \\ \frac{(u - C_x) \cdot b}{d} + \frac{b}{2} \end{bmatrix} \tag{2}$$

2.2.8 Results

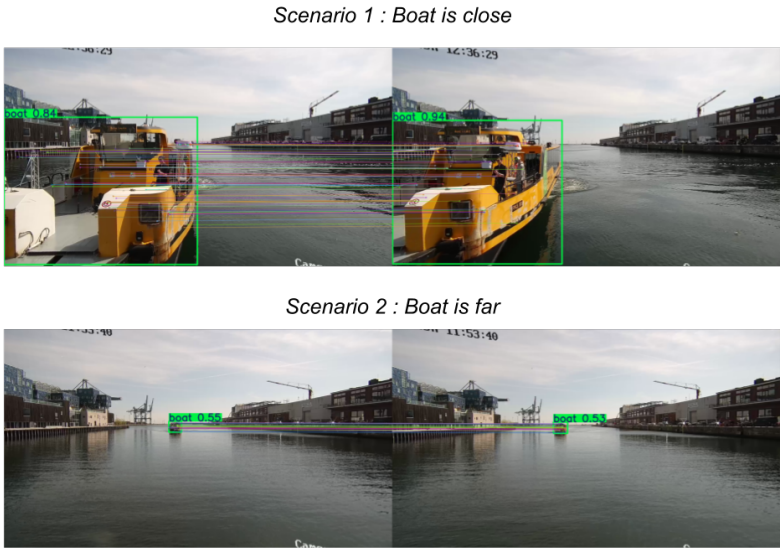
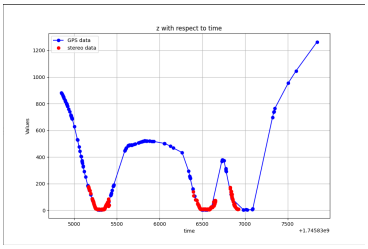
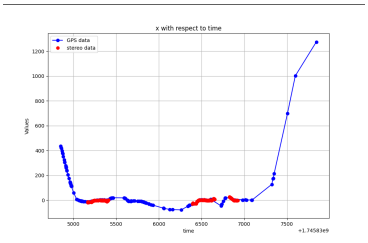


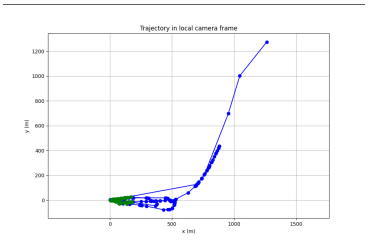
Figure 16: Example results of boat detection and stereo matching using YOLO and SIFT features. Each row shows a stereo image pair (left: left camera, right: right camera). For the first scenario, the estimated distance is 8.27m (mean disparity: 276.0 px, standard deviation: 93.0 px). Some background features were also matched but were filtered out during disparity computation. For the second scenario, the estimated distance is 144.39m (mean disparity: 15.81 px, standard deviation: 0.69 px).



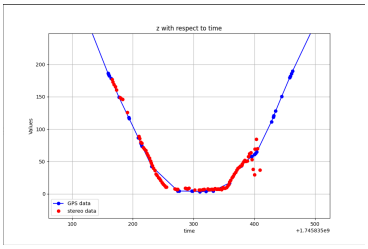
(a) Trajectory on the z-axis with respect to time, all data



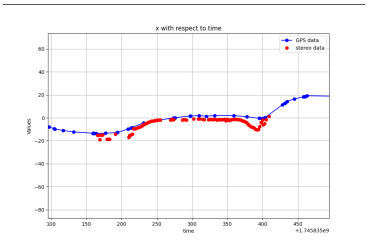
(b) Trajectory on the x-axis with respect to time, all data



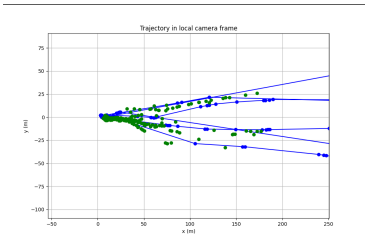
(c) Trajectory (x,z) as seen from above, all data



(d) Trajectory on the z-axis with respect to time, zoomed in



(e) Trajectory on the x-axis with respect to time, zoomed in



(f) Trajectory (x,z) as seen from above, zoomed in

Figure 17: Results for the boat's localization. The ground truth is displayed in blue. Overall, the estimated position follows the GPS trajectory, with occasional noise and data loss.

2.3 Improvement of the boat localization algorithms

Once this method was proven effective, it was necessary to make it more robust. The previous implementation was simplified since we only needed to test the method itself. It relied on dataset-specific features (e.g., building positions), and was limited to a single boat. For real-world use, the algorithm must handle multiple boats and generalize to any environment. Therefore, the next steps aimed to improve robustness and implement multi-object tracking to be able to use the framework in real conditions.

2.3.1 Improvement of boat detection

Motivations

One of the main issues encountered while detecting boats was the presence of buildings above the water surface, which YOLO often misdetected as boats. Indeed, the COCO dataset does not yet have a class for buildings, thus often mistaking them for something else.

To prevent that from happening without knowing their precise position in the images, it was necessary to fine-tune YOLO. This means adding more images to the corresponding dataset and training it again so that it takes those new examples into account.

In this case, one effective way to achieve this was by adding more boat pictures, as well as background pictures (i.e., images containing only buildings above the water surface, with no labels). This way, YOLO learns that there is nothing to detect in such cases. Another way could be to also add a "building" class, not to use those detections, but just so that the model can easily mark the difference between a boat and a building [8].

Process

The different steps that were followed to fine-tune the detection model were [9]:

- **Data collection:** First, find images that contain enough instances of the classes that will be in the model. Also add background images with no instance of any class — this helps reduce the number of false positives. Usually, at least 10,000 instances per class are recommended. For my use case, I took all the boat images from the COCO dataset (the one YOLO is trained on), plus the COCO background images. Additional boat and water pictures were also added. Finally, I created another dataset including a "building" class to test whether adding buildings would reduce false detections, as suggested in [8].

- **Annotation:** Label all the images accordingly, i.e., provide a data file specifying where the instances are and to which class they belong. This can be done with various tools; in this work, Roboflow was chosen. Most of the data was already annotated, but all the buildings visible in COCO images still had to be manually annotated, since COCO does not include a "building" class.

- **Dataset splitting:** Separate the images into three subsets: training (the largest), testing, and validation. The test and validation sets help analyze the model's performance — one during training (to provide feedback and improve learning), and one at the end (to compute the final performance metrics). Roboflow was again used to perform this step.

- **Training:** Once this is done, the training can begin. A base model must first be selected to avoid starting from scratch, and then fine-tuned with carefully chosen parameters, such as the maximum number of iterations. If this number is too small, the model will be undertrained and perform poorly; if too high, it risks overfitting the data (i.e., learning the training set too well and generalizing poorly). Training a model requires significant computational power — at least a GPU should be used. Since I did not have access to DTU's GPUs, I used Google Colab instead, which provides short-term GPU access within a notebook. As a result, my training runs could not exceed a certain number of iterations, which limited their performance.

- **Evaluation:** Once the model was trained, it was important to analyze its performance metrics, such as precision (the proportion of correct detections among all predictions), recall (the proportion of correct detections among all ground-truth instances), and mean average precision (mAP), which is the most commonly used indicator of model performance. If these metrics (ranging from 0 to 1) are too low, the model cannot be considered reliable. For comparison, the mAP_{50-95} of Yolo-v8m is 0.5 (mAP_{50-95} is the mAP computed over IoU thresholds from 0.5 to 0.95), it is the most selective criteria and the most commonly used for estimating a model's performances) [10].

Results

Two models were trained following the process above: one with boats only, and one with boats and buildings. The results, however, were rather disappointing. Many training runs were performed, with different parameters. But first, since the Colab session could only last 4 hours at most, it was not possible to get to a very high number of

iterations. For the model with only boats I reached up to 100 iterations, which is already good but could be improved, but for the boats-and-buildings set, which contained significantly more images, the training reached approximately 50 iterations — insufficient for convergence. The MAP_{50-95} remained low for every model.

Although the new models provided decent results when visualizing the detections on the Harbour Bus data, they did not outperform the original YOLO-v8m baseline. The "boats-and-buildings" model successfully differentiated buildings from boats but introduced many false building detections, leading to a lower mAP ($\approx$0.25 vs. 0.5 for YOLO-v8m). The "boats-only" model slightly improved long-distance detections but also increased false positives. Since detecting a wrong object (e.g., a building) is similar to detecting an irrelevant boat, it was decided to retain the original YOLO model and address false detections later through feature embeddings.

2.3.2 Outliers filtering

As explained above, for the algorithm to work on any given dataset, it was necessary to make a robust filtering on the disparities, that does not rely on parameters of a particular dataset like the positions of the horizon line or the buildings in the camera frame. In this section, we will focus only on what happens after a boat is detected, when the disparities are computed to get the distance. Three levels of filtering can be distinguished: first, a filter on the features (selecting which features actually belong to the boat and not the background). Once the right features and matches have been found, a filter on the disparities (or distances) to remove any outlier that might have passed through the previous filter. And lastly, once a distance measurement has been obtained, perform Kalman filtering to ensure that the measurement is correct (sometimes, because of desynchronisation between the cameras, there are errors), or make up for the data loss whenever the boat can't be detected.

Filtering of the features

When YOLO performs a detection, it returns a bounding box, i.e. a rectangle. When the boat is far away, this rectangle fits it quite well, usually, but when the boat grows closer, it looks less and less like a rectangle, and a more significant portion of the background lies within the bounding box. Because of this, when the feature detection and matching are done, a lot of features that do not belong to the boat, but to the background, are matched. This deflates the mean disparity (since the background is further), and wrongly increases the output distance (see Figure 18).

Before, in order to get rid of this effect, any feature that lay in the building zone was discarded. However, this implied knowing where the buildings were. A detection algorithm could have found their position, but the detection algorithm developed for this purpose did not perform reliably as mentioned above. Therefore, it was instead chosen to add segmentation to the detection. This means refining the detection area by finding a mask which really fits the boat.

To perform this segmentation, Fast Segment Anything Model (FastSAM) was chosen [11].

Once bounding boxes are extracted, FastSAM can be applied inside the bounding boxes to get a mask of the object inside. The segmentation mask was cleaned using

morphological operations (opening followed by closing) to remove noise and fill holes. Only the largest connected component (the boat) was kept, and the mask was slightly dilated to include the object's edges, where many SIFT features lie. All those operations were performed using OpenCV's morphological operations tools.

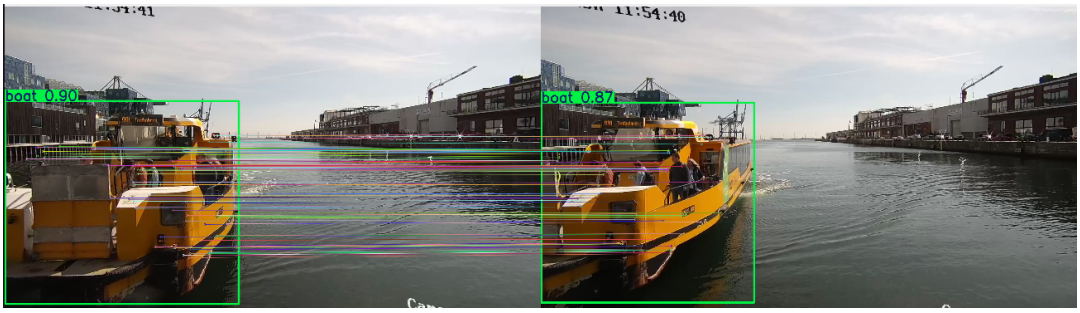


Figure 18: Illustration of the matching of background points, in the top left corner of the detection.

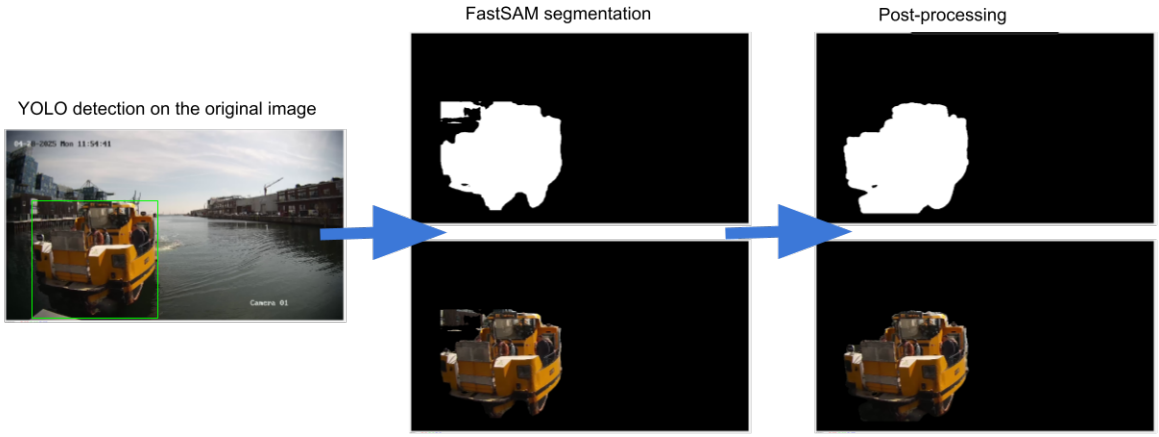


Figure 19: Example of segmentation on one boat. Above are the masks, and below the masks superposed with the original pictures. On the detection frame, a lot of the background is within the bounding box, and in the first segmentation some of it still belongs to the mask, which is cleaned out in the last step.

Filtering of the disparities/distances

Next, once the features are matched, the list of disparities can be computed. In this list, there may be some outliers left (due to bad matching, or some points that still lay in the background...). To remove them and get results as clean as possible before the Kalman filtering, a kind of Hampel filter was used, adapted a bit depending on the distance of the boat (because if the boat is close, a bigger standard deviation is expected than when it is far).

The Hampel filter will simply suppress outliers using the median and an adaptive standard deviation threshold. Only the values in an interval of $3\sigma z$ around the median are kept, with the threshold σz adjusted depending on how far the boat is. The thresholds were set a bit higher than the usual standard deviations of the measurements, so that good data is not removed on other datasets with higher standard deviations, with longer boats, for example. Also, the calculations were conducted using distance (meters) rather than disparities (pixels), which helps for having thresholds that actually make physical sense.

Kalman filter

The final and most important filtering layer was the Kalman filter, used both to smooth noisy measurements and predict missing data caused by data reception gaps.

Here, the most obvious example of the effect of the filter is when the images are desynchronized. Since there sometimes is a 1 second gap between the left and right images, in specific cases (when the boat is turning) it does not have the same angle on both frames and this results in a wrong computed disparity (as illustrated in figure 20).

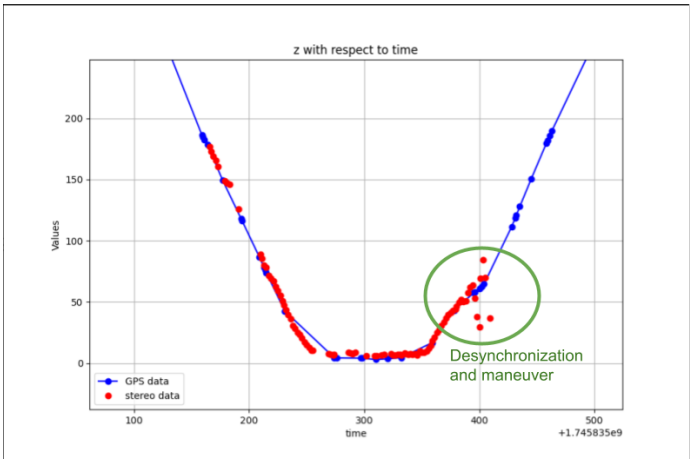


Figure 20: Illustration of the effect of simultaneous data desynchronization and boat maneuver on the z-coordinate

In this case, the boat's state is represented by its position and velocity in the (X, Z) plane:

$$\hat{x}_k = \begin{bmatrix} X_k \\ Z_k \\ V_{x,k} \\ V_{z,k} \end{bmatrix}, \qquad \text{where } X, Z \text{ are the positions and } V_x, V_z \text{ the velocities.}$$

We assume a constant-velocity motion model, which can be expressed as:

$$\hat{x}_{k|k-1} = F_k \, \hat{x}_{k-1|k-1} + w_k,$$

where $w_k \sim \mathcal{N}(0, Q_k)$ is the process noise, and the state transition matrix is:

$$F_k = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

At each time step k, only the positions (X_k, Z_k) are measured. The measurement model is therefore:

$$z_k = H\,\hat{x}_k + v_k, \qquad v_k \sim \mathcal{N}(0, R_k),$$

with

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Prediction step

$$\hat{x}_{k|k-1} = F_k\,\hat{x}_{k-1|k-1},$$

$$P_{k|k-1} = F_k\,P_{k-1|k-1}\,F_k^\top + Q_k.$$

The process noise covariance Q_k depends on the acceleration variance σ_a^2, representing uncertainty in the boat's motion dynamics:

$$Q_k = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix}.$$

Update (correction) step

After receiving a new measurement z_k, the correction is performed as follows:

$$K_k = P_{k|k-1}\,H^\top \left(H\,P_{k|k-1}\,H^\top + R_k\right)^{-1},$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(z_k - H\,\hat{x}_{k|k-1}\right),$$

$$P_{k|k} = (I - K_k H)\,P_{k|k-1}.$$

Mahalanobis gating

To make the Kalman filter more robust to outliers, a *Mahalanobis distance gating* step was added before the update. The innovation (or residual) is defined as

$$y_k = z_k - H\,\hat{x}_{k|k-1},$$

with its covariance

$$S_k = H\,P_{k|k-1}\,H^\top + R_k.$$

The Mahalanobis distance is then computed as:

$$\gamma_k = y_k^\top S_k^{-1} y_k.$$

This scalar quantity measures how far the new observation z_k lies from the expected distribution of innovations, taking into account both the predicted uncertainty and the measurement noise. If γ_k exceeds a predefined chi-squared threshold χ^2_{thresh} (corresponding to the desired confidence interval, e.g. 95%), the measurement is considered inconsistent with the model and is discarded:

$$\text{if } \gamma_k > \chi^2_{\text{thresh}} \quad \Rightarrow \quad \text{update rejected.}$$

In practice, this gating mechanism prevents abrupt corrections caused by the desynchronization mentioned above. To avoid rejecting good measurements during initialization, the gating is only applied when consecutive measurements are available (i.e. when $\Delta t \leq 2$ s).

Physical interpretation

- R_k represents the uncertainty of the measurement, derived from the standard deviation of stereo disparity values, so adaptive in this case. - Q_k defines the confidence in the motion model: a larger σ_a makes the filter more reactive but noisier, while a smaller σ_a makes it more stable but slower to respond. Physically, it represents the standard deviation of the acceleration noise. In this case, it is set to 0.5 m.s^{-2}. - The relative magnitudes of $P_{k|k-1}$ and R_k determine the balance between prediction and measurement: if $P_{k|k-1} \ll R_k$, the filter trusts the prediction more; if $P_{k|k-1} \gg R_k$, it relies more on the new measurement.

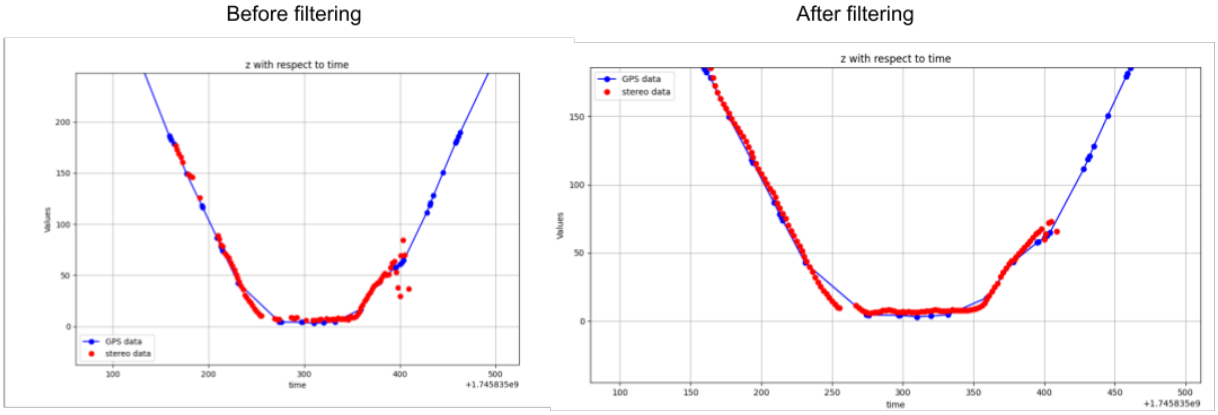


Figure 21: Before and after the filtering, illustrated on a close-up of the z-coordinate with respect to time. The trajectory is less noisy especially in the desynchronization zone, and data has been correctly predicted whenever the boat or features couldn't be detected. The only gaps left are when there are data reception gaps.

2.3.3 Multi-object tracking

If several boats are on a frame, the detection step will return several bounding boxes. But, how do we know how to associate all of the boats over all the frames with respect to time, or in other words, how to compute the trajectory of each individual boat over time ? It seems obvious when we look at it, but is it not the case for a computer... In order to associate the boats over time and compute their trajectory, multi-object tracking must be performed. This means that, thanks to a neural network, an ID number will be

associated to each boat seen throughout the frames. Ideally, one given boat must keep the same ID number on all the frames.

To perform multi-object tracking, StrongSort was used (from the Boxmot repository [12]). This tracker is the most efficient to date [13], which is necessary when working with challenging data (here, with a very low number of frames per second (0.5 fps)).

StrongSort relies on a neural network which combines motion modeling (via a Kalman filter) with visual similarity through feature embeddings. This enables to give an ID to each boat on any frame, and helps keeping stable IDs even under short occlusions — a key advantage for harbour scenes.

Since StrongSort is open source, a few of the tracker's parameters were modified for the identification, because it is designed, like all MOT models, for videos with much higher frame rates. Essentially, it was made more permissive because the boats would move and change aspects more than initially anticipated from one frame to another. So the maximal cosine distance (the "visual" distance between embeddings) and the maximal IoU (intersection over union, which calculates how much two boxes overlap) were increased, and the threshold for valid detections was reduced. This way, the chances of reidentification in case of occlusion were greatly improved.

The figures 22 and 23 illustrate the trajectories of all boats during the whole experiment. One color is one ID, and black indicates that no ID was allocated (this is the case at the beginning of each track since StrongSort needs some detections beforehand to initialize, and for objects that appear briefly, like false detections). The groundtruth is displayed in grey (for the Harbour Bus we followed, many of the tracks below are other boats or other Harbour Buses, this is why there are more).

We can observe that many points indeed follow each other and most tracks are well defined. There are still some instances of ID loss, like in the orange trajectory (better seen on Figure 23), but those are difficult to prevent here with the low FPS, and those cases are still a minority.

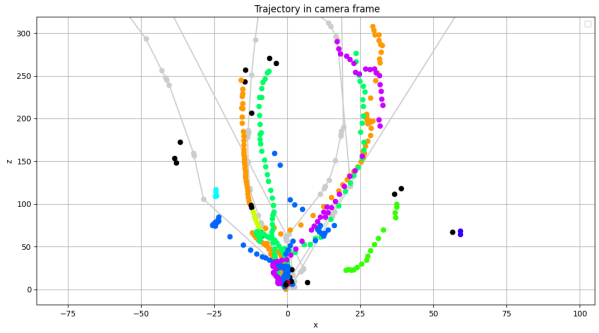


Figure 22: Illustration of ID allocation for the (x,z) trajectory

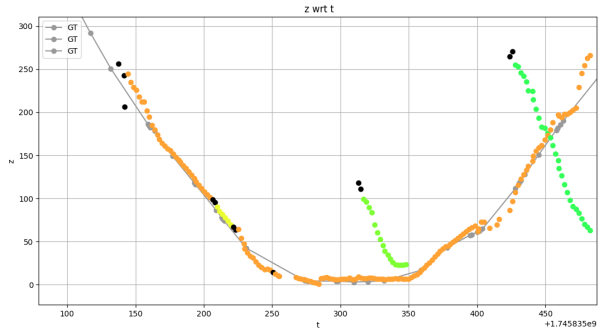


Figure 23: Close-up on the z-axis to highlight ID loss

The next and last step will be to identify the boat of interest among all the boats (here, the Harbour Bus), and select its trajectory. We consider that we know what kind of boat we wish to follow, and its appearance. Therefore, feature embeddings computed with DINO [14] were used. A small dataset (50 images) of the target vessel was built

from various viewpoints. For each detection, embeddings were compared to the reference dataset using cosine similarity. The maximal cosine distance was kept, and if it was superior to a chosen threshold, the boat was considered the same as the one in the dataset. Despite occasional failures (mainly when viewing the boat from the rear), the method reliably recognized the Harbour Bus in most cases, and discarded the other boats or false detections.

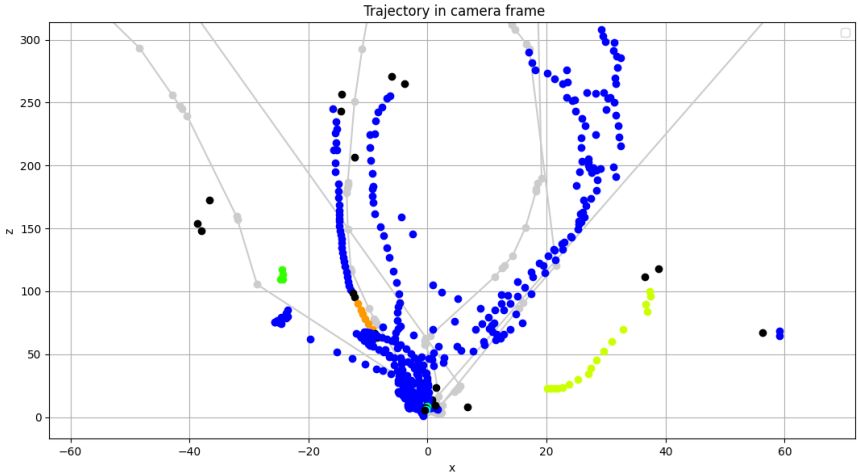
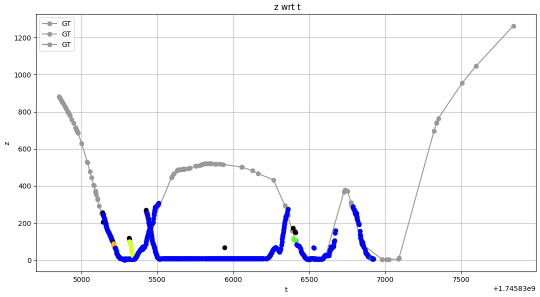
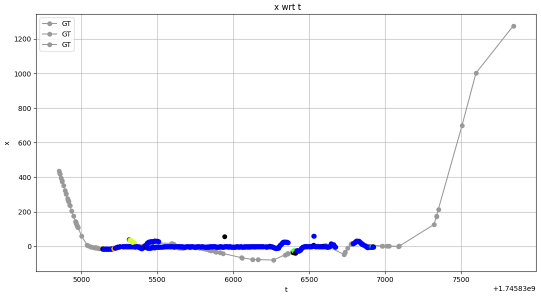


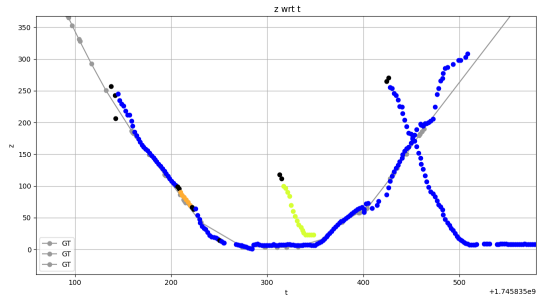
Figure 24: (x,z) trajectories of all boats, with Harbour Buses in blue and GPS ground truth in grey



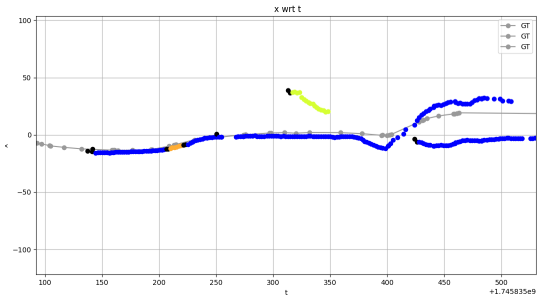
(a) Trajectory on the z-axis with respect to time, all data



(b) Trajectory on the x-axis with respect to time, all data



(c) Trajectory on the z-axis with respect to time, zoomed in



(d) Trajectory on the x-axis with respect to time, zoomed in

Figure 25: Trajectories of all boats, with Harbour Buses in blue. The ground truth is displayed in grey. There is another Harbour Bus (a real one, with tourists), in the middle of the frames, hence the blue trajectory that doesn't follow our Harbour Bus ground truth. Overall, the Harbour Bus is well recognized, and other boats are not misdetected as Harbour Buses.

2.3.4 Discussion

The results are overall rather satisfactory. The boats have been successfully detected, their position correctly estimated, several boats were followed at once and the boat of interest, the Harbour Bus, was detected as such most of the time.

But there still is room for improvement. Some ID losses could not be prevented as seen in Figure 22, and the Harbour Bus recognition is not completely satisfactory yet. The boat is not always recognized when it is turned backwards or when it is not seen long enough, and there are sometimes misdetections: there is one occurence of a building being misdetected as a Harbour Bus (see the blue points on the far right of Figure 24).

It is essential to have a pretty extensive database for each boat, the Harbour Bus dataset was maybe not large enough and the recognition could be easily improved by adding more reference images.

3 Conclusion

During this internship, I contributed to the development of a land-based stereo vision system for vessel localization, as part of the CREA group's research on autonomous

maritime perception. The goal was to design and validate a platform capable of tracking boats in the Copenhagen harbour using visual sensors only.

The first step was extracting, timestamping, and organizing for further processing the images acquired during the data collection campaign conducted in the harbour.

Then, the next step was to develop the first algorithm for boat localization. This included calibrating the stereo cameras, pairing the images based on their timestamps, and detecting the target vessel. I implemented a complete pipeline combining YOLO-based object detection, SIFT feature matching, disparity computation, and distance estimation.

After, the work moved toward improving the robustness and adaptability of the method. Several experiments were conducted to enhance object detection through YOLO fine-tuning, even if this wasn't added in the end. A three-layer filtering approach was then developed—combining FastSAM segmentation, Hampel statistical filtering, and Kalman temporal filtering—to handle outliers and missing data. Finally, the integration of StrongSort enabled multi-object tracking, and DINO-based embeddings were used for visual re-identification of the Harbour Bus.

Some time was also spent improving the physical setup. This involved designing and improving the waterproof box that hosts the cameras and computing hardware, ensuring stability and weather resistance for long-term outdoor deployment.

Overall, the project resulted in a functional and modular system for real-time vessel detection and localization. The combination of hardware design, algorithmic development, and data analysis provided a complete overview of a vision-based perception pipeline, from sensor setup to performance evaluation.

Future work could focus on improving real-time performance — for example, by replacing SIFT with faster feature detectors such as ORB — and integrating the infrared camera to enhance detection under low-visibility conditions. Additionally, expanding the embedding database or fine-tuning a dedicated recognition network could further improve the system's ability to distinguish between different types of boats.

This internship provided valuable research experience at the intersection of computer vision and autonomous maritime systems, and contributed to ongoing efforts within the CREA group to enhance the reliability of visual-based navigation.

References

[1] Helgesen, K. Vasstein, E. Brekke, and A. Stahl, "Heterogeneous multi-sensor tracking for an autonomous surface vehicle in a littoral environment," *Ocean Engineering*, vol. 252, p. 111168, May 2022.

[2] M. K. M. N.-A.-R. A. Islam, Md. Asikuzzaman and M. Pickering, "Stereo vision-based 3d positioning and tracking." `https://www.researchgate.net/publication/343146086_Stereo_Vision-Based_3D_Positioning_and_Tracking`, 2020.

[3] D. Kogan, "mrcal." `https://mrcal.secretsauce.net`, 2021. Documentation du projet.

[4] D. Kogan, "mrcal: camera calibration and uncertainty propagation toolbox." https://github.com/dkogan/mrcal, 2021. GitHub repository.

[5] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. Official Ultralytics documentation, AGPL-3.0 licence.

[6] R. Varghese and S. M., "Yolov8: A novel object detection algorithm with enhanced performance and robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, pp. 1–6, 2024.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[8] S. V. Grini, "Object detection in maritime environments." https://edmundfo.folk.ntnu.no/msc2019-2020/grini_simen_msc_reduced.pdf, 2019. Master Thesis.

[9] A. Yadav, "Fine-tuning yolov8: a practical guide." https://medium.com/@amit25173/fine-tuning-yolov8-a-practical-guide-61343dada5c1, December 2024.

[10] G. Jocher, "Model comparisons: Choose the best object detection model for your project." https://docs.ultralytics.com/compare/yolov8-vs-yolo11/#ultralytics-yolov8, 2024.

[11] Y. A. Y. D.-T. Y. M. L. M. T. J. W. X. Zhao, W. Ding, "Fast segment anything." https://www.researchgate.net/publication/371758266_Fast_Segment_Anything, 2023.

[12] M. Brostrom, "boxmot." https://github.com/mikel-brostrom/boxmot, 2022. GitHub repository.

[13] Y. S. Y. Z.-F. S. T. G. H. M. Y. Du, Z. Zhao, "Strongsort: Make deepsort great again," *arXiv*, 2022.

[14] I. M. H. J.-J. M. P. B. M. Caron, H. Touvron and A. Joulin, "Emerging properties in self-supervised vision transformers," *arXiv preprint*, vol. abs/2104.14294, 2021.