DTU

DTU Electro
Department of Electrical and Photonics Engineering

Localization of a GNSS denied UAV relative to a Moving Local Reference Frame in an Offshore Environment

Master Thesis
February, 2024

By
Ernestas Simutis

Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Electrical and Photonics Engineering, Elektrovej, Building 326, 2800 Kgs. Lyngby Denmark
https://electro.dtu.dk/

Approval

This thesis has been prepared over six months at the Section for Automation & Control, Department of Electrical and Photonics Engineering, at the Technical University of Denmark, DTU, in partial fulfillment for the degree Master of Science in Engineering, MSc Eng.

Ernestas Simutis - s212571

Signature

2024-02-02

Date

Abstract

Thesis addresses the challenge of autonomously localizing an Unmanned Aerial Vehicle (UAV) in a GNSS (Global Navigation Satellite System) denied offshore environment, specifically focusing on tracking of a moving surface vessel. The uniqueness of this task lies in the offshore setting, characterized by a lack of static features or landmarks for traditional localization methods. The primary objective is to develop a novel localization system enabling the UAV to autonomously position itself relative to a vessel despite the complexities of a non-inertial reference frame, caused by the vessel's own volition and external sea influences. A significant hurdle is the absence of a motion model for the vessel, which is treated as an unknown entity, with no shared information between the UAV and the vessel. Project aims to overcome these challenges by introducing an approach for UAV localization relative to the moving vessel, thereby enabling autonomous navigation with real-time localization feedback in offshore environment where GNSS is unreliable or unavailable.

Acknowledgments

Thank you Roberto and Jeppe for invaluable guidance, support, and the opportunity to bounce ideas throughout the project.

Thank you Caro for your incredible support, patience, and encouragement in all the dreadful (and sometimes cheerful) moments throughout these past months.

Notation

a	scalar
$\boldsymbol{a}$	vector
${}^{B}\boldsymbol{a}$	vector expressed in frame B
${}^{A}_{B}\boldsymbol{t}$	translation vector between frame B and frame A origins
$\boldsymbol{a} \cdot \boldsymbol{b}$	dot product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a} \times \boldsymbol{b}$	cross product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{A}$	matrix
$\boldsymbol{A}^{T}$	matrix transpose
$\boldsymbol{A}^{\dagger}$	Moore-Penrose pseudoinverse of matrix $\boldsymbol{A}$
$\boldsymbol{I}_{N}$	$N \times N$ identity matrix
$\boldsymbol{0}_{N \times N}$	$N \times N$ zero matrix
${}^{A}_{B}\boldsymbol{R}$	rotation matrix orientation of frame B relative to frame A
$\boldsymbol{q}$	quaternion
$\boldsymbol{q}_{a} \otimes \boldsymbol{q}_{b}$	quaternion product
${}^{A}\boldsymbol{q}$	quaternion expressed in frame A
${}^{A}_{B}\boldsymbol{q}$	quaternion orientation of frame B relative to frame A
$[\boldsymbol{a}]_{\times}$	skew symmetric matrix of vector $\boldsymbol{a}$

Contents

Chapter 1

Introduction

The chapter first outlines a detailed problem formulation together with motivation for developing a UAV state estimation framework under given conditions. Later proceeds to cover literature addressing related challenges, the setup for testing and assessing performance, assumptions made and the notable contributions of the thesis. Finally, the chapter concludes with an overview of the report structure.

1.1 Problem Statement & Motivation

This project deals with the problem of localizing a GNSS denied Unmanned Aerial Vehicle (UAV) in an offshore environment, where the aim of the UAV is to locate a moving surface vessel in a local relative frame between the two agents. Due to the homogeneity of an offshore environment, the UAV is unable to localize relative to static features/landmark or external reference positioning system like GPS. Therefore, a method for localizing the UAV relative to the surface vessel has to be developed to eliminate the reliance on the references mentioned above. This poses challenges in that the surface vessel is moving both according to its own volition and due to external influences from the sea, which means that its local frame is non-inertial. Developed method should be able to extract information from a video feed and other on-board sensors that can be used to determine the relative position, velocity, etc. between the UAV and the surface vessel.

The most important assumption going forward is that there is a single vessel in FOV (Field of View) of camera at all times. In practice, the condition can be achieved by introducing a visual servoing controller after detecting the vessel in the the FOV of the camera or applying a controller on the state estimate itself. Despite this assumption, a significant amount of effort is invested into lifting this constraint and making the system more robust in case of visual track loss or failure. The case where multiple vessels are present in the FOV is not considered in this thesis and would pose additional challenges, such as vessel identification or expansion of state space to include multiple vessels in the estimation problem.

If a UAV is equipped with a relative localization capabilities in GNSS-denied environment and using this information as feedback the developed system could be applied for various practical applications like:

- **Compensation for GPS Vulnerabilities.** GPS signals can be subject to interference or spoofing. By localizing relative to a surface vessel, the UAV can maintain operational capabilities even in the event of loss or corruption of the GPS signal.

This capability is crucial to maintain the continuity of critical missions, particularly in scenarios where GPS reliability is a concern.

- **Search and Rescue Operations.** UAVs equipped with relative localization capability can autonomously track and follow rescue vessels, providing aerial perspectives that are vital for locating individuals in distress at sea. This system would allow UAVs to operate effectively in vast, featureless maritime environments where traditional navigation systems are less effective.
- **Environmental Monitoring.** The system can be employed for environmental monitoring missions, such as monitoring marine wildlife. In these cases, the UAV ability to autonomously follow a moving vessel enables the continuous observation of dynamic environmental phenomena.
- **Maritime Surveillance and Security.** In scenarios where territorial waters need monitoring or where there is a need to track unauthorized or illegal activities, such as smuggling or illegal fishing, UAVs can offer persistent real-time surveillance capabilities. Unlike stationary surveillance systems, UAVs can dynamically adjust their position to maintain optimal observation of a target vessel.
- **Relative Localization to Arbitrary Targets.** The system could be easily generalized to an arbitrary object tracking in a GPS denied environment. For example, in a warehouse environment where the GPS signal is not available, the system could be used to track a moving object, such as a forklift or a person.

In summary, the ability to operate in GNSS-denied environments ensures resilience and versatility in various real-world scenarios where traditional navigation systems may fail or be inadequate.

1.2 Platform & Setup

The system is tested on a "Holybro X500" drone (seen in Figure 1.1) in real-world field tests flying over water. The drone is equipped with a Pixhawk 6C flight controller, companion computer LattePanda 3 and the following sensors: IMU (Inertial Measurement Unit), compass, altimeter (laser based range sensor), barometer, GPS receiver (used only for ground truth measurements and evaluation) and RGB-D camera.

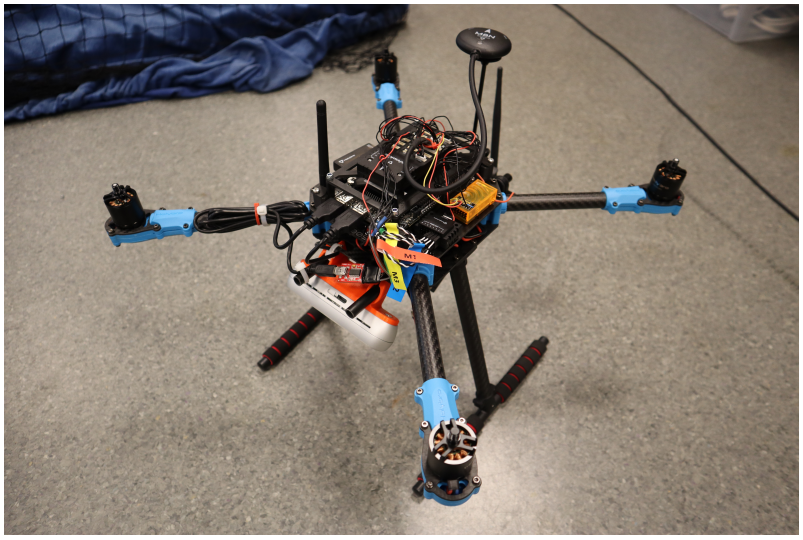


Figure 1.1: The drone platform used in experiments.

The computer has an Intel® Celeron® N5105 CPU and 8GB of RAM. This creates a challenge due to limited processing resources, especially since the algorithm must operate in real time. A key concern is the vehicle's battery capacity, which is affected by the weight of the payload. This constraint restricts the choice of computational platforms that can be mounted on the UAV. For instance, a graphics processing unit equipped computer could aid in accelerating vision processing tasks, but would reduce flight time due to extra weight and power consumption.

1.3 Related Work

This section provides a comprehensive review of relevant research areas, including state estimation, visual object detection and tracking, visual inertial odometry, and general aspects of computer vision relevant to the thesis.

Attitude and Heading Reference Systems. Attitude and Heading Reference Systems (AHRS) can be divided into two main types: Bayesian probabilistic approaches like the Kalman Filter [1] and complementary filters such as the Mahony Filter [2] or the Madgwick Filter [3] relying on optimization approaches to minimize the estimate error. The Kalman Filter is more complex due to linearization of the system and estimation of noise parameters for the process model and each and every sensor. On the other hand, filters like the Mahony and Madgwick are simpler, more efficient, and easier to implement, with little need for tuning besides static biases.

The Madgwick filter, in particular, uses gradient descent to correct orientation estimates based on gravity and the Earth's magnetic field. These are reliable references for estimating the orientation that can be continuously used to nudge the orientation in the right direction by optimization techniques. Madgwick's filter has been shown in [3] to perform on par or better than Kalman-based filters in accuracy for attitude estimation. One major short-coming is sensitivity to high accelerations because the gravity vector gets distorted and might results in erroneous observed direction of gravity. Furthermore, distortion of the local magnetic field can also lead to errors in orientation estimation, but this is a minor issue when flying above the ground. Finally, the Madgwick filter is extremely lightweight and computationally efficient, making it ideal for embedded systems and low-power devices.

Visual Object Detection & Tracking. Deep learning based approaches perform an excellent job of identifying objects in pictures and outperform traditional computer vision methods by a large margin. Some prominent architectures for the job are End-to-End Object Detection with Transformers (DETR) [4] and You Only Look Once YOLO [5]. The former is interesting because of the simplistic architecture that removes reliance on hand-crafted components such as NMS (Non-Maximum Suppression) and anchor boxes while still achieving state-of-the-art results. The latter method is a more established convolutional neural network-based approach combined with the previously mentioned hand-crafted components to post-process the network outputs. The performance of both methods is very similar but the latter is more widely used in the industry due to its maturity and availability of open source implementations.

YOLOv7 [6] is one of the latest extensions of the original YOLO networks. At the time of writing the algorithm surpasses most of the other object detection algorithms in terms of both accuracy and runtime. Their contribution called trainable bag-of-freebie dives in the components of architecture trying to optimize the training and inference performance with efficient architecture. The resulting algorithm is capable of achieving more than 100 FPS (Frames Per Second) on a single commodity GPU with 56.8% AP (Average Precision)

on COCO dataset [7]. Despite their effectiveness, neural network-based methods can be impractically slow on resource-limited devices like UAVs, where using a heavy, power-intensive GPU is not a viable option. For real-time object tracking, where processing is required for every frame at rates of 30Hz or higher, the speed of the algorithm becomes crucial.

Conventional tracking algorithms such as KCF (Kernelized Correlation Filter) [8] or MOSSE (Minimum Output Sum of Squared Error) [9] were developed back in the days before deep learning completely took over and can be considered obsolete. However, the tradeoff between short-term accuracy and latency of these algorithms is still appealing to certain applications. For instance, MOSSE authors claim that their algorithm is able to run at more than 600 FPS on single commodity CPU core while keeping the track of an object over extended period of time (up to couple of minutes of test sequences). The inner working of these early tracking algorithms was to search the image for similar features detected in the last frame to maintain a track on an object. Although the tracking is initially accurate, it becomes unreliable over time due to changes in appearance, occlusions, or other interferences. Additionally, the algorithms must be initialized manually or by other heuristics to begin the tracking process.

Relative Positioning. Many methods present techniques of landing a UAV relative to a known visual marker like in [10, 11]. The state estimate is based on visually detecting a pre-defined structure/marker and localizing a robot/platform against it. Another possible approach is utilizing the known geometry of environment to retrieve 3 dimensional representation of a pixel. Paper [12] shows how geometric constraints can be added to a problem and jointly optimized to estimate the target position. However, compared to these papers, this work assumes no presence of a marker or a known landmark, thus vision system has to rely on different aspects of sensory input to localize a foreign entity.

The problem of 3D Visual Object Tracking from a UAV is widely addressed in many previous works. For example, [13] deals with the problem of tracking multiple persons in the camera view in an urban environment. The setting of surrounding allows the authors to apply standard visual localization techniques to obtain drones state because of the visual richness of environment. The visual tracking is done with deep learning-based technique and the object can be back projected to 3D space with known camera parameters, drone state obtained in previous step and estimate of the ground plane. The authors show efficacy of the method on a real dataset and performance on par with the ground truth obtained from GPS. Furthermore, they observe some failure modes with false ground plane estimation, abrupt camera motions, and blur.

The study outlined in [14] describes a method for tracking monocular targets (specifically a car in an urban environment) with the presence of a GPS signal for the global positioning and inertial measurement unit. The image detection task is performed through the YOLO neural network while the tracking task is performed utilizing the KCF [8] visual tracking method based on the correlation filters discussed previously. Given the known altitude and attitude of the UAV, the authors derive a simple expression for calculating the relative position between the UAV and a target vehicle. The estimated attitude and position of the UAV is obtained by an IMU and GPS fusion but details of the estimator are not provided. The work presented in the paper relies on the availability of the GNSS signal and compute the location of the target in a global reference frame. Thesis work builds on the ideas developed in the paper to removes the assumption of GNSS signal availability and localize the target in a local relative reference frame. Furthermore, the paper includes a control method to guide the UAV in following the target and keeping it in the FOV of the

camera. The experiments were performed with a DJI drone and a NVIDIA Jetson TX1 computer. The results show that the UAV is able to track a car in an urban environment with a maximum error of 5 meters. Another major difference between the thesis and the paper is that the thesis is focused on tracking a moving vessel in a structure-less offshore environment.

Optical Flow. Camera data provides a rich source of information about the surrounding environment and changes across frames. Optical flow is a key concept in computer vision primarily concerned with estimating pixel displacements between consecutive images and enables numerous application like visual odometry, image stabilization, etc. Key idea behind most of these algorithms is to identify and match local image features across frames, with the flow defined as the displacement of these features. A prominent example is the "Lucas-Kanade" feature tracker [15]. Feature based methods rely on the presence of distinct features in both images and tend to struggle in environments lacking rich texture and distinct structure, as they require discernible features to track. Given that offshore environments typically lack all of the above, the aforementioned methods are not well suited for addressing the challenges outlined in the thesis.

In contrast with feature-based approaches, direct methods operate across the entire image, focusing on minimizing metrics like pixel intensities and coherence over small regions to obtain a dense flow. These methods are computationally more demanding, but offer robustness against texture-less areas. Although, they can be sensitive to illumination changes and large displacements, the dense flow provides a larger set of data points across the image plane, thus in combination with robust fitting techniques can provide a reliable estimate of the global optical flow. This is particularly beneficial in offshore environments, where the lack of distinct features makes feature-based approaches less effective or renders them unusable. As it will be shown later on, dense optical flow can be utilized to estimate the velocity of the UAV even when the vessel is out of sight during the operation. The literature offers many dense optical flow algorithms, for instance RLOF (Robust Local Optical Flow) [16, 17, 18, 19], Fast Optical Flow using DIS (Dense Inverse Search) [20] and many more.

Due to computational burden of dense correspondences, one has to make a tradeoff between accuracy and runtime of the algorithm. The paper "Fast Optical Flow using DIS" [20] presents a method for computing dense optical flow, which is significantly faster than state-of-the-art methods while maintaining comparable accuracy. The paper benchmarks DIS with other optical flow methods like RLOF and other variations of dense optical flow algorithms showcasing its high frame-rate performance (authors claim rates in range of $10-600$Hz), making it well-suited for real-time applications. The method's limitations are primarily in handling motion discontinuities and large displacements. This paper includes detailed experiments and analysis on benchmark datasets like Sintel and KITTI, demonstrating the efficiency and effectiveness of DIS compared to other optical flow methods.

Velocity Estimation from Optical Flow. Optical flow with auxiliary sensor data can be converted into 3D space, enabling the estimation of velocity. The paper [21] focuses on estimating the 6-DoF (Degrees of Freedom) velocity of a camera using feature points in RGB-D images. The algorithm establishes a relationship between camera velocity and feature point velocity using an image Jacobian matrix. The camera's spatial velocity, composed of linear and angular components, is calculated from the velocities of feature points in the image plane, determined using depth values and feature point coordinates. A post-filtering step with a Kalman filter refines the velocity estimation, enhancing accuracy by addressing potential inaccuracies from camera movement and image noise. The

shortcoming of the approach lies in the assumption that the camera can provide reliable depth measurements, which is not always the case, especially while operating outdoors or points are far from the camera.

The paper [22] presents a fully autonomous setup for micro aerial vehicles that leverages on-board hardware for velocity estimation and control. The core of this work is utilization of the continuous homography constraint for recovering ego-velocity from optical flow, using a monocular camera and IMU for gravity vector and angular velocity estimation. The authors developed two variants of the classical continuous 4-point algorithm and performed extensive experimental evaluations against known GT (ground truth). The results demonstrate the method's effectiveness in accurately determining the ego-velocity of a flying UAV under realistic conditions, using limited on-board computational resources. The method is able to estimate the velocity with a mean error of 0.042m/s and a standard deviation of 0.031 m/s. The flight was conducted in a lab, at relatively low height and with no flow outlier rejection protocol therefore further investigation is required to assess efficacy of the method in outdoor environment and different altitudes. Additionally, the paper details how the velocity estimation was successfully utilized for closed-loop control of the UAV's velocity in real-time. The difference approach in the last section utilizes the information from IMU to estimate gravity vector and assumes that all points lie on the ground plane. Still the estimated velocity is a factor of depth to the plane which must be obtained by other means.

The image Jacobian (used in the above works) can be found in paper [23, 24]. The application discussed primarily involves the visual servoing of a robot with a camera. However, as demonstrated in [21, 22], this principle can also be used to estimate velocity of a moving camera. This work will use a similar approach to obtain the velocity of the UAV from the optical flow but operating in a different (offshore) environment and depth obtained from altitude estimation.

The work in [25] presents a novel indoor navigation technique for Micro Aerial Vehicles (MAVs) using optical flow and inertial data. It introduces a depth map generated from an onboard omnidirectional camera's optical flow for real-time obstacle detection and avoidance. The paper's core is a refined absolute-scale velocity estimation method, integrating visual and inertial measurements into a least-squares problem solving for depths and velocity of a vehicle jointly. This technique is demonstrated through simulations, showing effective MAV navigation in indoor, GPS-denied environments without the need of pre-existing depth information of pixel features like in the previous two cases.

Sensor fusion. By far the most notable work in the field of sensor fusion is the seminal paper on the Kalman Filter [1]. The work presents a method for estimating the state of a linear dynamical system from a series of noisy sensor measurements. Since the time of publication, many people covered and extended the original work, for instance [26], covers the filter in more modern notation and discusses the extension to a non-linear case. To this day the estimator remains highly relevant for most information fusion applications and is widely used in the industry.

More recent works like [27, 28] address the inherent complexity of selecting the right noise parameters for the Kalman filter. Key takeaway is that hand-tuned parameters are not optimal and these can be adjusted after collecting experimental data and observing the ground truth state (by complementary sensors or other means that are not available during operation) and framing an optimization problem to find optimal noise characteristics. Specifically, paper "Optimization or Architecture: How to Hack Kalman filtering" [28] shows automatic MSE (Mean Squared Error) optimization over process and sensor noise

covariances from data can greatly improve estimation accuracy. Extensive experiments show that the method is able to outperform hand-tuned filter and even non-linear (neural) techniques. Moreover, this has immense practical implications since the architecture or filter is left unchanged and noise parameters can be seamlessly updated in already deployed systems.

1.4 Contributions

The main contributions of this thesis are as follows:

1. Integration of AHRS system with positioning system to obtained full 6 DoF state estimate of the UAV.
2. Object detection fine-tuning & adaptation to the offshore environment to detect vessels at sea.
3. Real-time tracking algorithm design to aid detection system to track a vessel under large latency characteristics of the detector. The algorithm is able to recover from track loss, reacquire the target and maintain minimal lag between data acquisition and tracking output.
4. System for recovering 3D relative target position from a monocular camera and altitude estimate.
5. Utilization of dense optical flow to obtain an absolute spatial velocity measurement under the operating conditions.
6. Bayesian sensor fusion mechanism to integrate available information into a unified agent's pose estimate.
7. Proposed general framework for relative localization to a moving target in offshore environment using only on-board sensing. During an extensive literature review, no equivalent work was found.
8. Deployment-ready open source implementation of the system fulfilling real-time and accuracy requirements. The source code is freely available to download[1].
9. Design and execution of experimental campaign to collect a dataset of real-world flights with ground truth measurements.
10. Comprehensive evaluation of system efficacy and efficiency on the aforementioned dataset, coupled with an empirical assessment of the fulfillment of the real-time requirements.

1.5 Overview

Thesis report is structured in the following manner:

- Methodology (Chapter 2): This chapter delves into the theoretical framework, detailing the methods employed in the thesis and an in-depth look at the implementation aspects of these methods.
- Experiments (Chapter 3): Here, the results of the experiments conducted are presented. This section systematically presents the results derived from the methodologies applied in practice.

[1] https://github.com/simutisernestas/UAVTR

- Discussion & Future Work (Chapter 4): The thesis concludes with big picture overview of the outcomes, implication of the findings and proposes directions for future research in this area.

Chapter 2

Methodology

This chapter outlines the techniques applied in this project and the practical aspects of deploying them in a real-world UAV (Unmanned Aerial Vehicle) system. It begins with an overview of the system, delves into the specifics of each component, and concludes with a description of the code structure implemented in the system.

2.1 Framework & Methods Overview

The framework is composed of three main components: a vision system, an AHRS system, and a localization system. In combination these components allow for localization of the UAV relative to the surface vessel i.e. obtaining a 6 Degree of Freedom (DoF) pose estimate of a UAV. The reference frames of pose vector are shown in Figure 2.1.

The attitude (θ, γ, ψ) is tracked in the Earth referenced inertial frame (ENU) while the relative position $\Delta \boldsymbol{p}$ is expressed in a moving non-inertial frame, both of them originating at the center of the drone $\boldsymbol{p}_d$. The need for separation of the two comes from the fact that orientation estimate relies on Earth referenced quantities like gravity and earth's magnetic field while position is referenced to the moving boat $\boldsymbol{p}_v$.

AHRS system is responsible for estimating the attitude of the UAV utilizing IMU and compass measurements providing an estimate of agent's orientation in a fixed ENU frame axes. The specific algorithm used here is the Madgwick Filter [3], which will be discussed in more detail later in this section. The effectiveness of the filter is crucial because all other subsystems depend on an accurate attitude estimate.

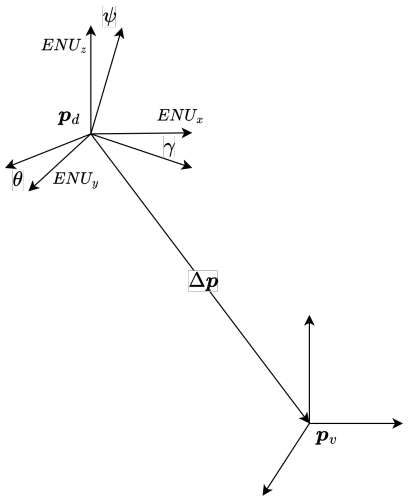


Figure 2.1: Reference frames.

The vision system is responsible for interpreting the video feed to determine the bounding box of the surface vessel in the image plane and tracking it across frames. The former is handled by a neural network YOLOv7 [6] and the latter by a MOSSE [20] correlation filter reinitialized whenever a detection is obtained from the object detector. Combination of these allows for real-time tracking at about 30Hz even though vessel detection on a regular consumer CPU can take up to a second to compute. Besides that, vision system is also used to

measure camera spatial velocity computed by utilizing DIS dense optical flow algorithm [20].

Lastly, the localization system merges vision and AHRS data with inputs from other sensors like barometers and altimeters to figure out the UAV's position and velocity relative to the surface vessel. It uses a Kalman filter to fuse noisy sensor data into a reliable estimate of the system's state. The full system setup is shown in Figure 2.2. Note that many details are omitted for clarity and will be explored in the following sections of this chapter.

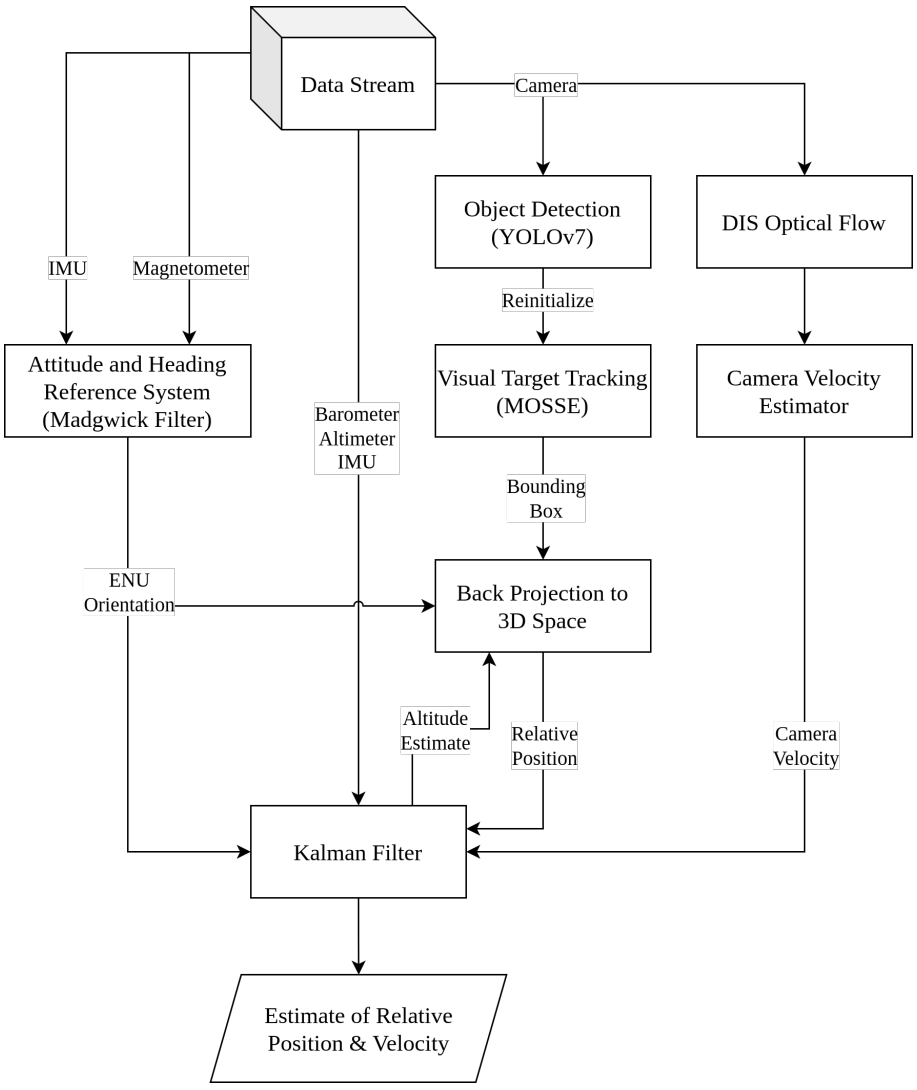


Figure 2.2: System diagram.

The complete state estimation pipeline is implemented in *C++* and operates in real-time on the companion computer on-board the UAV. This enables feedback control laws to be applied for closed-loop system regulation and ad-hoc navigation or tracking applications. All components are interfaced through ROS2 (Robot Operating System) middleware, allowing for easy inter-process communication between parts of the system.

2.2 Attitude and Heading Reference System

The AHRS system is based on the original work on the Madgwick filter [3]. The filter stands out as being extremely simple to tune, having a single adjustable parameter which can be determined by the characteristics of the system. It is also computationally efficient and can be run even on a low-power microcontroller.

2.2.1 Quaternions

Before describing the inner working of the Madgwick filter, mathematical foundations of quaternion operations are outlined as it is the underlying representation used in the filter. Afterwards, the filter itself is described in detail.

The quaternion product as a composition of orientations of different frames is described in Equation (2.1) where the resulting quaternion represents the orientation of frame C relative to frame A.

$$\overset{A}{C}\hat{\boldsymbol{q}} = \overset{B}{C}\hat{\boldsymbol{q}} \otimes \overset{A}{B}\hat{\boldsymbol{q}} \tag{2.1}$$

The product operations on the quaternions are achieved by the Hamiltonian product shown in Equation (2.2). The operation is not commutative i.e. $q_a \otimes q_b \neq q_b \otimes q_a$.

$$\begin{aligned} \boldsymbol{q_a} \otimes \boldsymbol{q_b} &= \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix} \otimes \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix} \\ &= \begin{bmatrix} a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3 \\ a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2 \\ a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1 \end{bmatrix}^T \end{aligned} \tag{2.2}$$

Quaternion inverse (or conjugate) is defined as per Equation (2.3). This can be used to get the inverse rotation of an orientation like in Equation (2.4).

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^* = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \end{bmatrix}^T \tag{2.3}$$

$$\overset{B}{A}\boldsymbol{q}^{-1} = \overset{B}{A}\boldsymbol{q}^* = \overset{A}{B}\boldsymbol{q} \tag{2.4}$$

The vector expressed in frame A can be rotated to frame B by quaternion multiplication Equation (2.5) (note that the vector is prepended with zero for compatibility with the quaternion product).

$$^B\boldsymbol{v} = \overset{A}{B}\boldsymbol{q} \otimes {}^A\boldsymbol{v} \otimes \overset{A}{B}\boldsymbol{q}^* \tag{2.5}$$

Lastly, one can easily convert quaternion orientation to a rotation matrix as per Equation (2.6) and use this for vector rotation or orientation representation.

$$\overset{A}{B}\boldsymbol{R} = \begin{bmatrix} 2q_1^2 - 1 + 2q_2^2 & 2\left(q_2q_3 + q_1q_4\right) & 2\left(q_2q_4 - q_1q_3\right) \\ 2\left(q_2q_3 - q_1q_4\right) & 2q_1^2 - 1 + 2q_3^2 & 2\left(q_3q_4 + q_1q_2\right) \\ 2\left(q_2q_4 + q_1q_3\right) & 2\left(q_3q_4 - q_1q_2\right) & 2q_1^2 - 1 + 2q_4^2 \end{bmatrix} \tag{2.6}$$

2.2.2 Madgwick Filter

The filter uses gyroscope measurements to estimate orientation of the sensor relative to Earth reference frame by integrating them over time. However, gyroscope measurements are subject to bias and noise, introducing drift in the estimation. To counteract this, accelerometer and magnetometer measurements are used to correct the estimate with gradient descent optimization.

Gyroscope rates can be arranged into a vector and rate of change of orientation of sensor relative to Earth references frame can be calculated as shown below in Equation (2.7) Notation of S left subscript or superscript represents a sensor reference frame while E left subscript or superscript Earth reference frame, for instance ENU.

$$\begin{aligned} {}^{S}\boldsymbol{\omega} &= \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \\ {}^{S}_{E}\dot{\boldsymbol{q}} &= \frac{1}{2} {}^{S}_{E}\hat{\boldsymbol{q}} \otimes {}^{S}\boldsymbol{\omega} \end{aligned} \tag{2.7}$$

This allows for integration of angular rates over time seen in Equation (2.8). The subscript ω denotes the change in orientation over time due to the angular rates measured by IMU while hat indicated the estimated/uncertain value. Due to its susceptibility to drift, as previously noted, this, by itself, cannot reliably estimate the attitude.

$$\begin{aligned} {}^{S}_{E}\dot{\boldsymbol{q}}_{\omega,t} &= \frac{1}{2} {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} \otimes {}^{S}\boldsymbol{\omega}_t \\ {}^{S}_{E}\boldsymbol{q}_{\omega,t} &= {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} + {}^{S}_{E}\dot{\boldsymbol{q}}_{\omega,t}\Delta t \end{aligned} \tag{2.8}$$

Additional information from IMU or other sensors can be used to drive the estimation error to zero. Madgwick filter introduces the correction based on the minimization of following objective function depicted in Equation (2.9). Essentially, the function takes a predefined (ideally static) Earth referenced vector ${}^{E}\hat{\boldsymbol{d}}$, rotates it by an estimated sensor orientation ${}^{S}_{E}\hat{\boldsymbol{q}}$ and subtracts the measured field vector ${}^{S}\hat{\boldsymbol{s}}$. As error approaches zero, the estimated orientation approaches the true orientation.

$$\begin{aligned} &\min_{{}^{S}_{E}\hat{\boldsymbol{q}}} \boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) \\ &\boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) = {}^{S}_{E}\hat{\boldsymbol{q}}^{*} \otimes {}^{E}\hat{\boldsymbol{d}} \otimes {}^{S}_{E}\hat{\boldsymbol{q}} - {}^{S}\hat{\boldsymbol{s}} \\ &{}^{S}_{E}\hat{\boldsymbol{q}} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix} \\ &{}^{E}\hat{\boldsymbol{d}} = \begin{bmatrix} 0 & d_x & d_y & d_z \end{bmatrix} \\ &{}^{S}\hat{\boldsymbol{s}} = \begin{bmatrix} 0 & s_x & s_y & s_z \end{bmatrix} \end{aligned} \tag{2.9}$$

The function is minimized by the gradient descent algorithm, which is a general iterative optimization technique. At each iteration, the update is calculated by multiplying a step size μ and the normalized gradient of the function as shown in Equation (2.10). The gradient is obtained by taking the derivative of the function with respect to the quaternion (Jacobian) and multiplying its transpose by the function itself illustrated in Equation (2.11).

$${}^{S}_{E}\boldsymbol{q}_{k+1} = {}^{S}_{E}\hat{\boldsymbol{q}}_k - \mu \frac{\nabla \boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}_k, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right)}{\left\| \nabla \boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}_k, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) \right\|} \tag{2.10}$$

$$\nabla \boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}_k, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) = \boldsymbol{J}^T\left({}^{S}_{E}\hat{\boldsymbol{q}}_k, {}^{E}\hat{\boldsymbol{d}}\right) \boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}_k, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) \tag{2.11}$$

The expanded objective function and Jacobian in general case can be computed as per Equations (2.12) and (2.13) respectively.

$$\begin{aligned} &\boldsymbol{f}\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{d}}, {}^{S}\hat{\boldsymbol{s}}\right) = \\ &= \begin{bmatrix} d_x q_w^2 + d_x q_x^2 - d_x q_y^2 - d_x q_z^2 + 2d_y q_w q_z + 2d_y q_x q_y - 2d_z q_w q_y + 2d_z q_x q_z - s_x \\ -2d_x q_w q_z + 2d_x q_x q_y + d_y q_w^2 - d_y q_x^2 + d_y q_y^2 - d_y q_z^2 + 2d_z q_w q_x + 2d_z q_y q_z - s_y \\ 2d_x q_w q_y + 2d_x q_x q_z - 2d_y q_w q_x + 2d_y q_y q_z + d_z q_w^2 - d_z q_x^2 - d_z q_y^2 + d_z q_z^2 - s_z \end{bmatrix} \end{aligned} \tag{2.12}$$

$$\boldsymbol{J}^T\left({}^S_E\hat{\boldsymbol{q}}, {}^E\hat{\boldsymbol{d}}\right) =$$

$$= \begin{bmatrix} 2d_xq_w + 2d_yq_z - 2d_zq_y & -2d_xq_z + 2d_yq_w + 2d_zq_x & 2d_xq_y - 2d_yq_x + 2d_zq_w \\ 2d_xq_x + 2d_yq_y + 2d_zq_z & 2d_xq_y - 2d_yq_x + 2d_zq_w & 2d_xq_z - 2d_yq_w - 2d_zq_x \\ -2d_xq_y + 2d_yq_x - 2d_zq_w & 2d_xq_x + 2d_yq_y + 2d_zq_z & 2d_xq_w + 2d_yq_z - 2d_zq_y \\ -2d_xq_z + 2d_yq_w + 2d_zq_x & -2d_xq_w - 2d_yq_z + 2d_zq_y & 2d_xq_x + 2d_yq_y + 2d_zq_z \end{bmatrix} \tag{2.13}$$

For instance, given a normalized gravity ${}^E\hat{\boldsymbol{g}}$ and accelerometer measurement ${}^S\hat{\boldsymbol{a}}$ Equation (2.14) vectors, the objective function and Jacobian boil down to Equations (2.15) and (2.16) respectively.

$$\begin{aligned} {}^E\hat{\boldsymbol{g}} &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^S\hat{\boldsymbol{a}} &= \begin{bmatrix} 0 & a_x & a_y & a_z \end{bmatrix} \end{aligned} \tag{2.14}$$

$$\boldsymbol{f}_g\left({}^S_E\hat{\boldsymbol{q}}, {}^E\hat{\boldsymbol{g}}, {}^S\hat{\boldsymbol{a}}\right) = \begin{bmatrix} -a_x - 2q_wq_y + 2q_xq_z \\ -a_y + 2q_wq_x + 2q_yq_z \\ -a_z + q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{2.15}$$

$$\boldsymbol{J}_g^T\left({}^S_E\hat{\boldsymbol{q}}, {}^E\hat{\boldsymbol{g}}\right) = \begin{bmatrix} -2q_y & 2q_x & 2q_w \\ 2q_z & 2q_w & -2q_x \\ -2q_w & 2q_z & -2q_y \\ 2q_x & 2q_y & 2q_z \end{bmatrix} \tag{2.16}$$

Combination of the gyroscope integration shown in Equation (2.8) and the gradient decent correction shown in Equation (2.10) forms the basis of Madgwick filter. Algorithm 1 depicts the procedure how attitude can be estimated from a sequence of given normalized angular rates and acceleration measurements. The initial state can be set to an arbitrary value and single tuning parameter μ is used to control the rate of convergence of the filter. This version does not consider constant scaling or biases, thus offline calibration is implied before running the filter.

Algorithm 1 Madgwick Filter with Gravity Reference

Require: Angular rates $\boldsymbol{\Omega}$, Accelerometer measurements $\boldsymbol{A}$, Sampling frequency f, Parameter μ, Initial Orientation $\boldsymbol{q}_0$

$\boldsymbol{g} \leftarrow [0, 0, 0, 1]^T$
$\Delta t \leftarrow 1/f$
$\boldsymbol{q} \leftarrow \boldsymbol{q}_0$
for $i = 1$ to length($\boldsymbol{\Omega}$) **do**
  $\boldsymbol{a}_q \leftarrow [0, \boldsymbol{A}[i]]^T$
  $\dot{\boldsymbol{q}} \leftarrow \frac{1}{2}\boldsymbol{q} \otimes [0, \boldsymbol{\Omega}[i]]^T$
  $\boldsymbol{f} \leftarrow \text{ObjectiveFunction}(\boldsymbol{q}, \boldsymbol{g}, \boldsymbol{a}_q)$
  $\boldsymbol{J} \leftarrow \text{Jacobian}(\boldsymbol{q}, \boldsymbol{g})$
  $\nabla\boldsymbol{f} \leftarrow \frac{\boldsymbol{J}^T\boldsymbol{f}}{\|\boldsymbol{J}^T\boldsymbol{f}\|}$
  $\dot{\boldsymbol{q}} \leftarrow \dot{\boldsymbol{q}} - \mu \cdot \nabla\boldsymbol{f}$
  $\boldsymbol{q} \leftarrow \boldsymbol{q} + \dot{\boldsymbol{q}}\Delta t$
  $\boldsymbol{q} \leftarrow \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|}$
end for

Gravity reference only provides information about roll and pitch of the system, and relying solely on gyroscope to estimate the heading reference is prone to drift as discussed earlier. However, the Madgwick filter can work with any vector field, for instance, most common

addition is a magnetometer which can provide a reliable heading reference assuming no presence of local magnetic field distortions like metals or magnets present in an environment. The assumption while flying outdoors is almost never an issue, although one must be careful to place the sensor away from UAV motors and other electrical components.

The magnetometer correction step can be derived by substituting ${}^{E}\hat{\boldsymbol{d}}$ with ${}^{E}\hat{\boldsymbol{b}}$ and ${}^{S}\hat{\boldsymbol{s}}$ with ${}^{S}\hat{\boldsymbol{m}}$ from Equation (2.17) in Equation (2.9). Sensor orientation aligned with earth's magnetic field can be assumed to have only two non-zero components (Y component is negligible), q_w and q_z as shown in Equation (2.17).

$$\begin{aligned} {}^{E}\hat{\boldsymbol{b}} &= \begin{bmatrix} 0 & b_x & 0 & b_z \end{bmatrix} \\ {}^{S}\hat{\boldsymbol{m}} &= \begin{bmatrix} 0 & m_x & m_y & m_z \end{bmatrix} \end{aligned} \tag{2.17}$$

The objective function and Jacobian for the magnetometer case can be computed as outlined in Equations (2.18) and (2.19) respectively.

$$\boldsymbol{f}_m\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}\right) = \begin{bmatrix} b_x q_w^2 + b_x q_x^2 - b_x q_y^2 - b_x q_z^2 - 2b_z q_w q_y + 2b_z q_x q_z - m_x \\ -2b_x q_w q_z + 2b_x q_x q_y + 2b_z q_w q_x + 2b_z q_y q_z - m_y \\ 2b_x q_w q_y + 2b_x q_x q_z + b_z q_w^2 - b_z q_x^2 - b_z q_y^2 + b_z q_z^2 - m_z \end{bmatrix} \tag{2.18}$$

$$\boldsymbol{J}_m^T\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}\right) = \begin{bmatrix} 2b_x q_w - 2b_z q_y & -2b_x q_z + 2b_z q_x & 2b_x q_y + 2b_z q_w \\ 2b_x q_x + 2b_z q_z & 2b_x q_y + 2b_z q_w & 2b_x q_z - 2b_z q_x \\ -2b_x q_y - 2b_z q_w & 2b_x q_x + 2b_z q_z & 2b_x q_w - 2b_z q_y \\ -2b_x q_z + 2b_z q_x & -2b_x q_w + 2b_z q_y & 2b_x q_x + 2b_z q_z \end{bmatrix} \tag{2.19}$$

These can be combined with the gravity reference in a joint function gradient expression shown in Equation (2.20) and optimized together by gradient decent, as described previously.

$$\begin{aligned} \boldsymbol{f}_{g,m}\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{S}\hat{\boldsymbol{a}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}\right) &= \begin{bmatrix} \boldsymbol{f}_g\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{S}\hat{\boldsymbol{a}}\right) \\ \boldsymbol{f}_m\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}\right) \end{bmatrix} \\ \boldsymbol{J}_{g,m}\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}\right) &= \begin{bmatrix} \boldsymbol{J}_g^T\left({}^{S}_{E}\hat{\boldsymbol{q}}\right) \\ \boldsymbol{J}_m^T\left({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}\right) \end{bmatrix} \\ \nabla \boldsymbol{f} &= \begin{bmatrix} \boldsymbol{J}_g^T\left({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}\right) \boldsymbol{f}_g\left({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}, {}^{S}\hat{\boldsymbol{a}}_t\right) \\ \boldsymbol{J}_{g,b}^T\left({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}, {}^{E}\hat{\boldsymbol{b}}\right) \boldsymbol{f}_{g,b}\left({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}, {}^{S}\hat{\boldsymbol{a}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}\right) \end{bmatrix} \end{aligned} \tag{2.20}$$

The magnetic reference can be a static local Earth magnetic field vector measured at a location before the operation. The other approach that the authors of [3] provide is to use a dynamic reference derived by considering magnetic distortion compensation. In essence, the magnetic field measurement, ${}^{S}\hat{\boldsymbol{m}}_t$, is aligned with the Earth by current estimate of sensor orientation and then ${}^{E}\hat{\boldsymbol{b}}_t$ is constructed to have zero inclination shown in Equation (2.21). This ensures that magnetic distortion has impact only on bearing angle estimation and at the same time eliminates the need for a pre-defined magnetic reference vector.

$$\begin{aligned} {}^{E}\hat{\boldsymbol{h}}_t = \begin{bmatrix} 0 & h_x & h_y & h_z \end{bmatrix} &= {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} \otimes {}^{S}\hat{\boldsymbol{m}}_t \otimes {}^{S}_{E}\hat{\boldsymbol{q}}^{*}_{est,t-1} \\ {}^{E}\hat{\boldsymbol{b}}_t &= \begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix} \end{aligned} \tag{2.21}$$

Resulting implementation is almost identical to the one in Algorithm 1 with the exception of magnetic reference instead of gravity and the objective function and Jacobian computed jointly per both sensors thus not included for brevity.

Finally, updates in the filter can occur asynchronously, guided by the specific sensor rates. Integration of angular rates follows the IMU's sampling frequency, whereas corrections are aligned with each sensor's individual rate. Despite this variability, the fundamental operations of the filter remain unchanged.

2.3 Visual Target Tracking

Target tracking in an image plane is decomposed into two sub-problems of object identification/detection and tracking over multiple frames. Although it could be approached entirely by end-to-end deep learning approach the problem is the runtime characteristics of those systems. The application requires real-time performance and thus a traditional computer vision tracking algorithm MOSSE [9] is combined with object detection provided by YOLOv7 [6] deep neural network. The section first introduces the main concept behind object detector and fine-tuning process to fit the application requirements. Then, the tracking algorithm incorporation is described in detail to achieve real-time performance under sparse detector measurements.

2.3.1 Neural Networks & Deep Learning

To begin with, let us introduce the basic concepts of neural networks and deep learning [29] because it is an essential part of the detector used here. The core idea of a neural network is to approximate a function/mapping Equation (2.22) between input $\boldsymbol{x}$ and output $\boldsymbol{y}$ with an arbitrary number of trainable parameters $\boldsymbol{\theta}$. The end goal is to find an optimal set of parameters $\boldsymbol{\theta}$ that minimizes the loss function $\mathcal{L}$ Equation (2.23) so that the network can approximate the mapping as closely as possible. In recent days, the function to be approximated can take form of mind-boggling tasks like predicting a most likely word to come in a sentence, essentially modeling human language in some cases as closely as a real person would.

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{2.22}$$

$$\boldsymbol{\theta} = \min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{2.23}$$

Choosing the internal structure of the function f internals is still an open research question but some architectures like Transformers [30] dominate the field by being able to efficiently map input from output independent of the nature of the data itself. In general, the function f is a combination of linear mappings and non-linear activation functions in between as shown in Equation (2.24). The linear mapping is a matrix multiplication of input $\boldsymbol{x}$ and a weight matrix $\boldsymbol{W}$ (from $\boldsymbol{\theta}$ Equation (2.25)) followed by an activation function σ which is a non-linear function like sigmoid or ReLU. The output of the activation function is then fed into the next layer as input and the process repeats until the final layer is reached. Because of this recursive structure the function can be arbitrary complex thus it is not a surprise that neural networks are able to approximate practically any function given enough data points to train on.

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}_L \sigma(\boldsymbol{W}_{L-1} \sigma(\boldsymbol{W}_{L-2} \dots \sigma(\boldsymbol{W}_1 \boldsymbol{x}))) \tag{2.24}$$

$$\boldsymbol{\theta} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, \dots, \boldsymbol{W}_L\} \tag{2.25}$$

The question is how to find the optimal set of parameters $\boldsymbol{\theta}$ that minimizes the loss function $\mathcal{L}$ Equation (2.23). Due to the complexity of the function f it may have many locally optimal set of parameters thus a huge part of the research goes into the efficient optimization of these networks. The most common approach is to use gradient descent which is an iterative optimization algorithm that uses the gradient of the loss function

$\nabla\mathcal{L}$ to update the parameters $\boldsymbol{\theta}$ in the direction of the steepest descent. The gradient is computed by taking the derivative of the loss function with respect to the parameters $\boldsymbol{\theta}$ Equation (2.26) and multiplying it by a step size μ which is a hyper-parameter of the algorithm. The step size controls the rate of convergence of the algorithm and is usually set by trial and error.

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu\nabla\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})) \tag{2.26}$$

The simplest example of this process can be illustrated by learning a linear function $f(x) = ax + b$ given a set of points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. The loss function is defined as a mean squared error between the predicted value $\hat{y}$ and the ground truth y Equation (2.27). The gradient of the loss function with respect to the parameters $\boldsymbol{\theta} = \{a, b\}$ is computed by taking the derivative of the loss function Equation (2.28) and multiplying it by a step size μ. After a number of iteration the algorithm convergences to the function defined by the set of points and this is nothing new but a simple linear regression.

$$\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.27}$$

$$\begin{aligned}\nabla\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})) &= \left[\frac{\partial\mathcal{L}}{\partial a} \quad \frac{\partial\mathcal{L}}{\partial b}\right] \\ &= \left[\frac{1}{n}\sum_{i=1}^{n} 2x_i(y_i - \hat{y}_i) \quad \frac{1}{n}\sum_{i=1}^{n} 2(y_i - \hat{y}_i)\right] \\ \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \mu\nabla\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta}))\end{aligned} \tag{2.28}$$

In reality a neural network is much more complex and taking analytical derivative of a function with respect to all its weights is virtually impossible. However, theory of calculus provides us with a chain rule Equation (2.29) that allows us to compute the derivative of a function $f(g(x))$ with respect to x by multiplying the derivative of the function f with respect to g and the derivative of the function g with respect to x. This reduces the problem to computing only local derivatives of the parameters with respect to their immediate inputs and the operation can be repeated recursively until the first input layer is reached in the chain of functions.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \tag{2.29}$$

For instance if we take a layered loss function $L(f(g(x)))$ and compute the derivative with respect to x we get Equation (2.30). Although here there is a single parameter x, this process scales up to any number of layers and parameters. The expression gives a clue on how each network parameter impacts the final output of the network and forms the core of back-propagation algorithm which is a general technique for computing gradients of functions expressed as computational graphs or chain of functions. By knowing gradients of each parameter in the network one can update them slightly in the direction of steepest descent and thus to minimize the loss function as per Equation (2.28).

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \tag{2.30}$$

Consider a simple neural network with two layers using mean squared error (MSE) as the loss function and hyperbolic tangent ($tanh$) as the activation function for the first layer. First layer is composed of input x, weight w_1, bias b_1, linear transformation z_1 and activation a_1 as per Equations (2.31) and (2.32). Second layer is composed of weight w_2, bias b_2 and linear transformation z_2 as per Equation (2.33). The loss function is

computed as per Equation (2.34). In neural network literature it is called a forward pass where operation are flowing from input to output in a connected network.

$$z_1 = w_1 x + b_1 \tag{2.31}$$
$$a_1 = \tanh(z_1) \tag{2.32}$$
$$z_2 = w_2 a_1 + b_2 \tag{2.33}$$
$$L = \frac{1}{2}(z_2 - Y)^2 \tag{2.34}$$

On the other hand the optimization variables are computed in a backward pass where gradient are computed for each and every weight that contributed to the output loss in the forward pass. Starting with gradient of loss w.r.t. z_2 Equation (2.35), this value is carried on to Equation (2.36) and Equation (2.37) to compute gradients of loss w.r.t. w_2 and b_2. And Equations (2.38) to (2.41) with their respective variables all the way up first layer weights w_1 and b_1.

$$\frac{\partial L}{\partial z_2} = 2(z_2 - Y) \tag{2.35}$$
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = 2(z_2 - Y) \cdot a_1 \tag{2.36}$$
$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = 2(z_2 - Y) \cdot 1 \tag{2.37}$$
$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} = 2(z_2 - Y) \cdot w_2 \tag{2.38}$$
$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} = 2(z_2 - Y) \cdot w_2 \cdot (1 - a_1^2) \tag{2.39}$$
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = (2(z_2 - Y) \cdot w_2 \cdot (1 - a_1^2)) \cdot x \tag{2.40}$$
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = 2(z_2 - Y) \cdot w^T \cdot (1 - a_1^2) \tag{2.41}$$

As it is clear by going step-by-step through back-propagation that every time the gradient is flowing backwards from the previous step towards the input. The only new addition is the right most chain rule component where we can compute a single derivative w.r.t. to the next element in computational graph. Thus the left most component can be cached for efficient calculation of gradient. The procedure forms basis for optimization where each weight can be updated by a small step size toward the direction minimizing the loss function as per Equation (2.26).

The conventional gradient decent does not work in practice due to the size of present-day datasets. A common technique to overcome this issue is applying a stochastic version of the algorithm called stochastic gradient decent (SGD) by instead of computing parameter gradients over full dataset it is only done for a smaller random subset of the data thus the stochastic part. This is repeated for multiple iterations until the loss function converges to a minimum. The optimization of neural networks is a wide research area and many other techniques exist to improve the convergence and its numerical behavior however is not the focus of this work.

Linear feed-forward layers Equation (2.24) are important but computationally very expensive because of the dense connections between the layers. Another important architecture is a convolutional neural network specifically design to deal with dense information like

images. The convolution operation Equation (2.42) can be used instead of a dense connections to reduce the number of parameters in the network. It is defined by a sliding window K of a fixed size $(m \times n)$ called kernel and is composed of trainable parameters that are applied to the input image I by definition of convolution Equation (2.42). The output of the operation is a resulting feature map S for a single kernel. Usually a network is build from many of these kernels stacked together to form a high dimensional feature maps and these are passed to another layer of kernels.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.42}$$

2.3.2 Object Detection

YOLO (You Only Look Once) presented in [5] is an object detection system that applies a single neural network for the tasks of classification and bounding box regression. First, a convolutional neural network is used for feature extraction from an image and dense feed-forward connection at the end for the final output. The input image is divided into a grid of cells (i.e. anchors) which after passing by the network outputs a confidence score (probability) for how likely it is to contain an object and the associated bounding boxes for each cell. The output is a single tensor consisting of B bounding boxes, C class probabilities for each cell of size S. Thus total size of the tensor is $S \times S \times (B * 5 * C)$, 5 due to the fact that bounding box and confidence score are represented by 5 numbers.

$$\begin{aligned}
\mathcal{L} = \lambda_{\text{coord}} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
+\lambda_{\text{coord}} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{2.43}$$

The network is optimized jointly with a loss function $\mathcal{L}$ Equation (2.43), S^2 represents the number of grid cells the image is divided into, and B denotes the number of bounding boxes each cell predicts. $\mathbf{1}_{ij}^{\text{obj}}$ and $\mathbf{1}_{ij}^{\text{noobj}}$ are indicator functions that equals 1 if an object is present or not present, respectively, in the i-th cell for the j-th bounding box. x_i, y_i, w_i, h_i are the ground truth center coordinates, width, and height of the bounding box, while $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$ are their respective predicted values. C_i and $\hat{C}_i$ represent the actual and predicted confidence scores that a box contains an object. $p_i(c)$ and $\hat{p}_i(c)$ denote the true and predicted probabilities of the object class in cell i belonging to class c. Finally, λ_{coord} and λ_{noobj} are weights to balance the importance of different components of the loss function, with λ_{coord} typically being greater to emphasize the accuracy of bounding box predictions.

Outputs are merged into a final prediction by aforementioned hand design techniques like NMS (Non-maximum suppression) [5]. NMS is a process in computer vision used for

refining object detection. It begins by sorting detected bounding boxes by their confidence scores. The box with the highest score is kept, and all other boxes that overlap with it beyond a certain threshold (measured as Intersection over Union - IoU) are suppressed. This process is repeated for each subsequent box in the sorted list, ensuring that for each detected object, only the bounding box with the highest confidence score is retained. This technique effectively reduces redundancy in detection results.

There have been numerous improvement on this original work on YOLO since its inception. This work utilized architecture YOLOv7 [6] introducing significant advancements over the original paper, focusing on accuracy and efficiency in real-time object detection. It implements "trainable bag-of-freebies," enhancing accuracy without increasing inference costs. New strategies in module replacement and dynamic label assignment tackle challenges in object detection. YOLOv7 employs extend and compound scaling methods for better parameter and computation utilization, resulting in a 40% reduction in parameters and 50% cut in computation compared to its predecessors. These improvements allow YOLOv7 to achieve higher detection accuracy and faster inference speeds, marking a substantial leap from the original YOLO model.

Although these are significant advancements in computational burden of the method the main ideas outlined in above discussion still largely apply. Even though network structure and gradient propagation paths are slightly different the training procedure and underlying optimization of the objective function remains unchanged. Moreover, the network is still quite heavy for CPU based systems, not reaching more than 1 FPS on a commodity laptop CPU. Therefore the rest of the section will focus on the contribution of adapting the network to the specific application at hand rather than dissecting the incremental improvements in the architecture. Specifically, the fine-tuning of the network to the specific task of vessel tracking and increasing the measurement density over time to meet application requirements.

Figure 2.3: Maritime dataset samples.

The weights provided in the original YOLOv7 [6] implementation repository are great for general purpose object detection tasks on diverse scenes. The weights are trained on COCO dataset [7] having 80 distinct categories of object including boats i.e. the target object in vessel tracking task. After assessing the performance of the pre-trained network it was clear that detection quality is not satisfiable and fine-tuning is necessary for application at hand. Retraining process is described in the following paragraphs to adapts the model to the specific task. This is a general framework that can be applied to any object detection tasks if a well labeled dataset is present.

Fine-tuning of the model was done a publicly available maritime vessel/boat detection dataset [31]. Figure 2.3 shows a few samples taken from the dataset. Additionally, a subset of the data from real-world experiments [32] was used for validation of re-trained model. Dataset consist of more than 20 thousand labeled images of one or more ships at

sea with labeled bounding boxes. Fine-tuning greatly improved results over the baseline of provided default weights generalizing to unseen cases in the experimental data discussed in the next chapter. From the samples provided above one can see that taken various different angles and mostly not aligned well with what a UAV would capture by a camera. Despite this fact the model was able to generalize well to the experimental data.

The official implementation of YOLOv7 [6] facilitates transfer learning on baseline models with minimal modifications, allowing for the adaptation of a pre-trained model to specific application needs through additional training on a custom dataset. The primary challenge in this process is the computational requirements, hence the DTU HPC cluster was utilized for the task. Training was conducted on a single NVIDIA A100 GPU for 100 epochs, with a batch size of 32 and a learning rate of 0.001. The process took approximately 1 hour to complete. The AP of the model on the validation set exceeded 0.8, which is significantly higher than the precision reported by the authors. It is important to note that the task was simplified compared to the original COCO detection challenge, as the model was required to detect only a single type of object within a highly homogeneous environment (surrounded by water), which likely contributed to the increased AP.

2.3.3 Visual Tracking

Detection model alone cannot provide real-time feedback thus combination with MOSSE adaptive correlation filter is used to mitigate the latency. The practical consideration of using them together are not trivial because of the large latency of the neural network and will be addressed in the implementation section. Here the focus is on the tracking algorithm itself.

MOSSE correlation filter [9] is a lightweight (can achieve hundreds of FPS performance on commodity CPU) visual tracking system based on correlation search in subsequent images. To achieve real-time execution the algorithm works on image data transformed to Fourier domain. This is enabled by the fact that convolution operation after FFT (Fast Fourier Transform) reduces to element wise multiplication. Starting with an input ROI (Region of Interest) image f and an arbitrary filter h and their Fourier domain counterparts $F = \mathcal{F}(f)$ and $H = \mathcal{F}(h)$ the correlation can be computed as per Equation (2.44) with $\odot$ representing element wise multiplication and H^* - complex conjugate of H. To go back to spatial domain the inverse Fourier transform can be applied as $\mathcal{F}^{-1}(Z) = z$. The capital notation here is implying quantities in Fourier domain while the lower case is for spatial domain.

$$G = \mathcal{F}(g) = F \odot H^* \tag{2.44}$$

DFT (Discrete Fourier Transform) for 2D data matrix f (like grey scale image) is defined as per Equation (2.45) [33]. The equation present a way to compute a single pixel value of the output matrix with a window of size $M \times N$ sliding over full image.

$$\mathcal{F}(u, v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) e^{-2\pi i \left(\frac{um}{M} + \frac{vn}{N}\right)} \tag{2.45}$$

While initializing filter is set to find and optimal filter H that minimizes the sum of squared error between the desired correlation G_i (usually a Gaussian response g_i with a peak at the center of bounding box as in Figure 2.4) and the actual response $F_i \odot H$ as per Equation (2.46). The index i is indicating a sample from a set of training images created before hand from the ground truth ROI by applying random warping (affine transformations) to original image to account for object changing its appearance over the tracking time window. The optimization problem has a closed form solution

as per Equation (2.47) plus authors augment the denominator with regularization term $(F_i \odot F_i^* + \epsilon)$ to improve stability of element wise division.

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{2.46}$$

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{2.47}$$

Figure 2.4: The left is an image captured from camera and on the right is the corresponding Gaussian response g_i for a bounding box of a vessel.

From here filter H^* is updated on each incoming frame j as per Equations (2.48) to (2.50) with a learning rate $\eta = 0.125$ to adapt for changing appearance and down-weighting the contribution from previous frames. The new peak of the response map in spatial domain is used to update the bounding box position and process repeats. The algorithm is summarized in Algorithm 2.

$$H_j^* = \frac{A_j}{B_j} \tag{2.48}$$

$$A_j = \eta(G_j \odot F_j^*) + (1 - \eta)A_{j-1} \tag{2.49}$$

$$B_j = \eta(F_j \odot F_j^*) + (1 - \eta)B_{j-1} \tag{2.50}$$

The MOSSE tracker is re-initialized with a bounding box whenever a new detection is available from YOLO model. The tracker fills in the gaps between these sparse measurement and works well in short time window in-between neural network detection. This concludes visual tracking system description and next section will investigate how the image plane bounding box in combination with other sensory inputs can be utilized to obtain target position.

Algorithm 2 MOSSE tracking algorithm

Input: f_1 - initial bounding box image patch
Output: b_j - bounding box center for frame j
$F \leftarrow generateWarped(f_1)$ - set of warped training images in Fourier domain
$G \leftarrow generateResponses(F)$ - set Gaussian responses in Fourier domain
$H_1^* \leftarrow (\sum_i G_i \odot F_i^*)/(\sum_i F_i \odot F_i^* + \epsilon)$
for $j = 2, 3, \dots$ **do**
    f_j - next bounding box image patch
    $F_j \leftarrow \mathcal{F}(f_j)$
    $G_j \leftarrow H_{j-1}^* \odot F_j$
    $g_j \leftarrow \mathcal{F}^{-1}(G_j)$
    $b_j \leftarrow argmax(g_j)$ - next bounding box center
    $A_j \leftarrow \eta(G_j \odot F_j^*) + (1-\eta)A_{j-1}$
    $B_j \leftarrow \eta(F_j \odot F_j^*) + (1-\eta)B_{j-1}$
    $H_j^* = \frac{A_j}{B_j}$
end for

2.4 3D Relative Target Positioning

Positioning relative to a surface is vessel is based on information from visual tracking subsystem. The video-feed data is a 2D stream of pixels therefore it is necessary to introduce fundamental theory of computer vision to be able to extract 3D information.

2.4.1 Camera Model

In order to do back-projection operation from planar camera image to 3D space it is important to understand camera intrinsic parameter encapsulated in camera matrix $\boldsymbol{K}$ Equation (2.51) [34, 35]. The math is following pinhole camera model meaning image is formed by projecting light rays through a single "pinhole" onto a plane i.e. image. The parameters in the matrix are focal length f - distance from plane to pinhole, (δ_x, δ_y) are displacement due to the fact that images are stored in computer from top-left corner of the plane and non-square pixels, and non-rectangular pixels are accounted by (α, β) respectively.

$$\boldsymbol{K} = \begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.51}$$

Additionally, camera is usually exposed to lens distortion effect that can be modelled radial distortion model presented in Equation (2.52) [35] where the mapping between distorted pixels $\boldsymbol{p}_d$ and the its equivalent $\boldsymbol{p}$ after undistortion is established. The distortion is caused by the fact that light rays are not passing through an actual pinhole but rather through a lens. Phenomena can be described by a polynomial function (Plumb Bob model) of the distance from the center of the image plane with coefficients $(k_1, k_2, \dots)$. The distortion is illustrated in Figure 2.5.

$$\begin{aligned} \boldsymbol{p}_\mathrm{d} &= \boldsymbol{p} \cdot (1 + \Delta r\,(\|\boldsymbol{p}\|_2)) \\ \Delta r(r) &= k_1 r^2 + k_2 r^4 + \dots \end{aligned} \tag{2.52}$$

All of the above described parameters can be obtained by calibration techniques abundant in the literature [34, 35]. The process is out of scope and not considered here because all the intrinsic and distortion camera parameters are provided by the manufacturer. Note that all the following image operation are done on undistorted images i.e. already accounted

for distortion. This can be easily done by having coefficients (from calibration or provided by manufacturer) using the mapping presented in Equation (2.52).

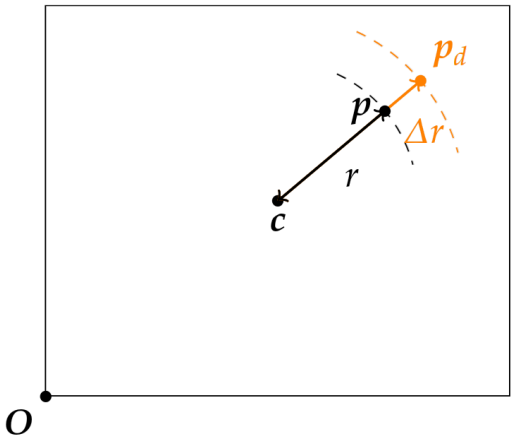


Figure 2.5: Lens distortion. Taken from [36].

2.4.2 Pixel Inverse Projection

One can obtain the 3D position of the target from 2D bounding box information and height measurement provided by other internal sensors and assuming that drone is flying at a significant height such as everything below is on the ground plane.

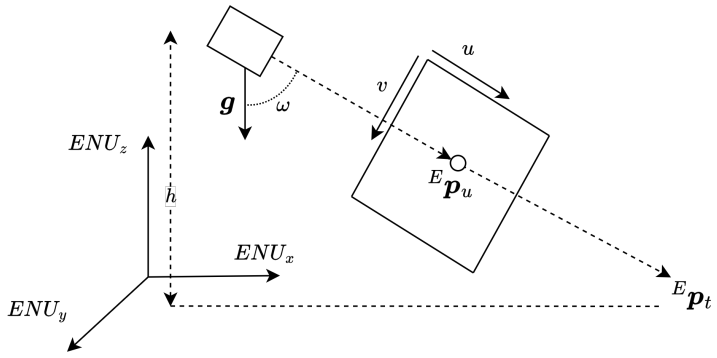


Figure 2.6: Inverse projection of a pixel to the target position.

Given a camera intrinsics matrix $\boldsymbol{K}$, camera orientation in ENU frame ${}^E_C\boldsymbol{R}$, homogenous pixel coordinate in image plane $\boldsymbol{p} = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$ and height h we can arrive at the target position in ENU frame ${}^E\boldsymbol{p}_t$ Equation (2.56) with assumption of it being on a ground plane [14]. First pixel is converted to camera frame point Equation (2.53) that is transformed to Earth frame Equation (2.54), normalized and then scaled by the depth information from Equation (2.55) to obtain the target position Equation (2.56). The calculations in Equations (2.53) to (2.56) are well illustrated visually in Figure 2.6. Note that in reality the camera and ENU frame origins are closely situated on the drone platform, thereby the

height is negative in ENU frame.

$$^{C}\boldsymbol{p} = \boldsymbol{K}^{-1}\boldsymbol{p} \tag{2.53}$$

$$^{E}\boldsymbol{p}_u = \frac{^{E}_{C}\boldsymbol{R}\,^{C}\boldsymbol{p}}{\|\,^{C}\boldsymbol{p}\|} \tag{2.54}$$

$$d = \frac{h}{^{E}\boldsymbol{p}_u^T \cdot \boldsymbol{g}} \tag{2.55}$$

$$^{E}\boldsymbol{p}_t = d \cdot {}^{E}\boldsymbol{p}_u \tag{2.56}$$

The depth in Equation (2.55) follows from the fact that dot product of ground plane normal $\boldsymbol{g} = \begin{bmatrix}0 & 0 & -1\end{bmatrix}^T$ and the unit vector pointing towards the target $^{E}\boldsymbol{p}_u$ is equals to $\cos(\omega)$, where ω is the angle between those two vectors. Note that, by the dot product definition two unit orthogonal vector dot product equals to zero i.e. $\boldsymbol{a} \cdot \boldsymbol{b} = \cos(\frac{\pi}{2}) = 0$. Thus there is a singularity at certain pixel location where ground normal and the vector pointing towards the target are orthogonal in Equation (2.55). Therefore pixels approaching this limit should be treated carefully.

2.5 Optical Flow Based Spatial Camera Velocity

The section initially delves into the implementation and application of the dense optical flow algorithm as utilized in the thesis. This algorithm is pivotal in analyzing and understanding the movement patterns in consecutive frames of video data. Following this, the section transitions into a detailed explanation of how the output from this algorithm, primarily in the form of flow vectors, is methodically converted into measures of camera spatial velocity. This conversion is crucial for accurately guiding the estimation of relative position, which involves determining the spatial relationship and movement dynamics between the camera and its surrounding environment. The thorough discussion in this section aims to provide a clear understanding of both the technical aspects of the optical flow algorithm and its practical application in spatial analysis for relative position estimation.

2.5.1 Dense Optical Flow

When selecting an optical flow algorithm for a given application, two critical considerations are needed. Firstly, the algorithm must meet real-time processing needs, ensuring efficient operation with minimal latency between frames, possibly at the expense of some accuracy for speed. Secondly, it needs to effectively handle environments with minimal texture, as encountered in offshore settings. In such environments, algorithms relying solely on feature extraction often struggle due to a lack of distinct features to track across frames. However, this issue can be mitigated by employing methods that establish dense correspondences and perform global optimization across the image plane. The algorithm for application at hand was selected to be DIS [20] because of its efficiency and dense correspondence search nature.

DIS optical flow algorithm consists of three main stages: inverse search for patch correspondences, creation of a dense displacement field through patch aggregation across multiple scales, and variational refinement of obtained flow vectors. Following paragraphs delve into the details of each stage.

The algorithm starts off by finding matching patches between two consecutive frames. For a single template patch T taken from image I_t, of size $\theta \times \theta$ located at a pixel $\boldsymbol{p} = \begin{bmatrix}x & y\end{bmatrix}^T$. An optimization problem is framed to find a best matching area in the other frame I_{t+1} such that loss function Equation (2.57) is minimized to find the optimal pixel displacement

vector $\boldsymbol{u} = \begin{bmatrix} u & v \end{bmatrix}^T$.

$$\boldsymbol{u} = \underset{\boldsymbol{u}}{\operatorname{argmin}} \sum_{x} \left[I_{t+1} \left(\boldsymbol{x} + \boldsymbol{u} \right) - T(\boldsymbol{x}) \right]^2 \tag{2.57}$$

The optimization involves two alternating steps where the objective is switched to $\Delta\boldsymbol{u}$ as per Equation (2.58) to optimize around the current estimate and then back to $\boldsymbol{u} = \boldsymbol{u} + \Delta\boldsymbol{u}$. The process is repeated until convergence of the the first loss function Equation (2.57) is achieved. The algorithm used for stepping toward a solution is the gradient decent. Furthermore, the process requires interpolation of the target window $I_{t+1}(\boldsymbol{x}+\boldsymbol{u})$ to achieve sub-pixel accuracy.

$$\Delta\boldsymbol{u} = \underset{\Delta\boldsymbol{u}}{\operatorname{argmin}} \sum_{x} \left[I_{t+1} \left(\boldsymbol{x} + \boldsymbol{u} + \Delta\boldsymbol{u} \right) - T(\boldsymbol{x}) \right]^2 \tag{2.58}$$

Additionally, the optimization objective can be inverted to an equivalent representation where gradient of an image I_t can be reused for efficient computation as per Equation (2.59) [37].

$$\Delta\boldsymbol{u} = \underset{\Delta\boldsymbol{u}}{\operatorname{argmin}} \sum_{x} \left[T(\boldsymbol{x} - \Delta\boldsymbol{u}) - I_{t+1}(\boldsymbol{x} + \boldsymbol{u}) \right]^2 \tag{2.59}$$

The second step is to create a dense flow field by aggregating information from different scales of the image going from coarse to fine levels. Different scales here represent the size of the patches that are taken into consideration. The algorithm goes through five steps for each scale:

1. **Grid creation**. Uniform grid of patches is created with a number of patches per dimension.
2. **Initialization**. The displacements from the previous scale are used as initial guess for the current scale. The first one being set to zero.
3. **Inverse search**. For each patch in the grid the inverse search is performed to find the optimal displacement vector as discussed above.
4. **Densification**. The obtained sparse flow field is densified by taking a weighted average of the displacements from different scales (N_s) overlapping at the pixel location as per Equation (2.60). Where λ_i is binary indicator if patch overlaps with the pixel, d_i is a function of intensity difference between the template and translated patch and Z is normalization factor equal to $Z = \sum_i \lambda_{i,\boldsymbol{x}} / \max\left(1, \|d_i(\boldsymbol{x})\|_2\right)$.
5. **Variational refinement**. The flow is refined further based on energy minimization described below.

$$\boldsymbol{U}_s(\boldsymbol{x}) = \frac{1}{Z} \sum_{i}^{N_s} \frac{\lambda_{i,\boldsymbol{x}}}{\max\left(1, \|d_i(\boldsymbol{x})\|_2\right)} \cdot \boldsymbol{u}_i \tag{2.60}$$

The last step is to refine flow field at each scale by minimizing the energy function Equation (2.61) [38] over an image domain Ω. The function is a sum of three terms each wrapped in a robust penalizer $\Phi = \sqrt{a^2 + \epsilon^2}$ with $\epsilon = 0.001$ and weighted by σ, γ, α respectively.

$$E(\boldsymbol{U}) = \int_{\Omega} \sigma\Psi\left(E_I\right) + \gamma\Psi\left(E_G\right) + \alpha\Psi\left(E_S\right) d\boldsymbol{x} \tag{2.61}$$

First term enforces the difference of intensities in original and translated location to be minimal. Specifically, pixel brightness constancy states that $I(x, y, t) = I(x+u, y+v, t+1)$ or $(\nabla_3 I)\boldsymbol{u} = 0$ where $\nabla_3 I$ is the image spatio-temporal gradient i.e. $\nabla_3 = (\delta x, \delta y, \delta t)$ and then $E_I = \boldsymbol{u}^T \beta(\nabla_3 I)(\nabla_3^T I)\boldsymbol{u}$ with β being normalization factor (norm of the gradient).

The second term adheres to gradient constancy assumption which is less sensitive than the first term to small variations in pixel brightness. The assumption states that $\nabla I(x, y, t) = \nabla I(x+u, y+v, t+1)$, with $\nabla = (\delta x, \delta y)$, $\boldsymbol{J}_{xy} = \beta_x \left(\nabla_3 I_{dx}\right)\left(\nabla_3^T I_{dx}\right) + \beta_y \left(\nabla_3 I_{dy}\right)\left(\nabla_3^T I_{dy}\right)$ and $E_G = \boldsymbol{u}^T \boldsymbol{J}_{xy} \boldsymbol{u}$ with β_x, β_y being normalization factors.

The smoothness term in the energy function is represented as the sum of the squared magnitudes of the gradients of the displacements, specifically $E_S = ||\nabla u||^2 + ||\nabla v||^2$. This term serves to minimize the overall variation in the flow field.

2.5.2 Camera Spatial Velocity

Dense optical flow provides rich information about camera movement from frame to frame. However, it is not directly usable for relative position estimation because the flow vectors are expressed in pixel coordinates which need to be treated separately in order to recover 3D spatial information about camera's trajectory. The following paragraphs describe how the flow vectors are converted to camera spatial velocity.

Jacobian matrix Equation (2.62) is called interaction matrix relating the optical flow vector velocities to camera spatial velocity $({}^C\boldsymbol{v}, {}^C\boldsymbol{\omega})$ i.e. twist ${}^C\boldsymbol{\xi}$ is derived in [23, 24] and can be compactly represented as $\dot{\boldsymbol{p}} = \boldsymbol{J}\,{}^C\boldsymbol{\xi}$. Where f is focal length of the camera (obtained from camera intrinsics calibration and is provided by the manufacturer), u and v are pixel coordinates in an image and Z is pixel depth in camera's frame of reference. All of the above excluding Z are known once the optical flow is obtained.

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} -\hat{f}/Z & 0 & u/Z & uv/\hat{f} & -\left(\hat{f} + u^2/\hat{f}\right) & v \\ 0 & -\hat{f}/Z & v/Z & \hat{f} + v^2/\hat{f} & -uv/\hat{f} & -u \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{2.62}$$

Note that time dependence could be removed and the same relationship would hold for displacements of pixels and camera linear and angular displacements. Nevertheless, solving for velocity is more advantageous later on in fusion because the accelerometer measurements are very noisy making integration of acceleration to velocity and position unstable. Therefore, velocity measurements help in grounding velocity estimation and in turn making position integration more robust.

The depths can be obtained by the same operations as shown in Equation (2.56) but the points must be expressed in camera frame and z component is equal to Z in Equation (2.62). This can be achieved by rotating the point back to camera frame and taking its third element as in Equation (2.63). The expression assumes that all points lie on the ground plane thus this is only valid when flying at significant height which is most often the case. The introduced error magnitude will be investigated later on in the section.

$$\begin{aligned} {}^C\boldsymbol{P} &= {}^E_C\boldsymbol{R}\,{}^E\boldsymbol{P} \\ {}^C\boldsymbol{P} &= \begin{bmatrix} X & Y & Z \end{bmatrix}^T \end{aligned} \tag{2.63}$$

Although the goal in [23, 24] is visual servoing of a robot the problem could be inverted to estimate the camera velocity from optical flow measurements. Since the $\boldsymbol{J}$ is a non-square

matrix, it is not invertible with a single sample of a pixel. However, if three points are stacked in a single vector, Jacobian becomes square and invertible. Note that this is not always the case, i.e. the matrix might be still ill-conditioned depending on chosen pixel coordinates.

If more than 3 flow vectors are used, these can be utilized for more robust fitting with least squares. For instance, one can use pseudo-inverse to solve this over-determined problem for n points as ${}^{C}\boldsymbol{\xi} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{p}}$. Where $\boldsymbol{J}^{\dagger} \in \mathbb{R}^{6\times 2n}$ is pseudo-inverse of $\boldsymbol{J}$ and $\dot{\boldsymbol{p}} \in \mathbb{R}^{2n}$ is the stacked vector of optical flow velocities. The pseudo inverse can be computed from the SVD decomposition of $\boldsymbol{J}$ as per Equation (2.64) where $\boldsymbol{\Sigma}^{+}$ is the inverse of all nonzero singular values and $\boldsymbol{U}$ and $\boldsymbol{V}$ are the left and right singular vector matrices.

$$\begin{aligned} \boldsymbol{J} &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{T} \\ \boldsymbol{J}^{\dagger} &= \boldsymbol{V}\boldsymbol{\Sigma}^{+}\boldsymbol{U}^{T} \end{aligned} \tag{2.64}$$

The important assumption here is that the environment is static or object in the scene move at significantly lower speed than UAV itself. This can be easily violated in the application at hand if vessel is moving at a substantial speed or surface water is not calm enough. The latter will be investigated more in the experimental part, however the former can be handled by another assumption that a moving vessel is occupying a a reasonably small area of an image.

RANSAC (Random Sample Consensus) algorithm can be applied to do robust fitting on a number of sample pixel velocities to find a "dominant" model over image domain s.t. outliers and vectors associated with dynamic object in the scene are eliminated from the least squares formulation. Number of iterations needed for RANSAC to converge with probability of p, ϵ percentage of outliers and s minimal number of points to instantiate a model can be found by Equation (2.65) [39]. The complete algorithm to convert dense optical flow to camera spatial velocity is summarized in Algorithm 3.

$$N = \frac{\log(1-p)}{\log\left(1-(1-\epsilon)^{s}\right)} \tag{2.65}$$

Up until now the twist vector ${}^{C}\boldsymbol{\xi}$ was considered in camera reference frame; therefore, the last step is to recover world frame velocity of the UAV. This can be easily achieved by using adjoint transformation matrix as per Equation (2.66) [24] to convert the twist vector to the drone body frame followed by rotation of the velocity component to the ENU frame as ${}^{E}\boldsymbol{v} = {}^{B}_{E}\boldsymbol{R}\,{}^{B}\boldsymbol{v}$. The adjoint matrix has only static transform component defined by the geometry of the sensor layout on the UAV frame ${}^{C}_{B}\boldsymbol{R}$ being the rotational part between the camera and the body frame and ${}^{C}_{B}\boldsymbol{t}$ being the arm length between the origins of the frames. Notation $[\boldsymbol{x}]_{\times}$ represent skew-symmetric matrix of a vector $\boldsymbol{x}$.

$${}^{B}\boldsymbol{\xi} = \begin{pmatrix} {}^{C}_{B}\boldsymbol{R} & [{}^{C}_{B}\boldsymbol{t}]_{\times}\,{}^{C}_{B}\boldsymbol{R} \\ \boldsymbol{0}_{3\times 3} & {}^{C}_{B}\boldsymbol{R} \end{pmatrix} {}^{C}\boldsymbol{\xi} \tag{2.66}$$

The fusion of velocity measurement with other sensory input will be discussed in the next section. Since the filter is operating in bayesian framework a noise model is required to be able to propagate uncertainty through the system. However, the problem of obtaining accurate measure of noise is multifaced and nontrivial because of the steps involved in the algorithm to obtain the velocity. There are multiple factors at play here, like the

Algorithm 3 Dense Flow to Camera Velocity

Require: Images $\boldsymbol{I}_t, \boldsymbol{I}_{t-1}$, Camera Extrinsics ${}^{C}_{E}\boldsymbol{R}_t$ & Intrinsics $\boldsymbol{K}$, Altitude h_t
Ensure: Camera Spatial Velocity ${}^{C}\boldsymbol{\xi}$

$\boldsymbol{f}_{all} \leftarrow$ ComputeDenseOpticalFlow$(\boldsymbol{I}_t, \boldsymbol{I}_{t-1})$
$(\boldsymbol{p}_{uv}, \boldsymbol{f}) \leftarrow$ DownSample$(\boldsymbol{I}_t, \boldsymbol{f}_{all})$
$\boldsymbol{d} \leftarrow$ ComputeDepths$(\boldsymbol{p}_{uv}, {}^{C}_{E}\boldsymbol{R}_t, h_t)$
$\boldsymbol{J} \leftarrow$ ComputeJacobian$(\boldsymbol{p}_{uv}, \boldsymbol{f}, \boldsymbol{d})$
$l \leftarrow 0$ ▷ Maximum inliers number
for $i = 1, 2, \ldots, k$ **do** ▷ k - Number of RANSAC iterations
  $(\boldsymbol{f}_s, \boldsymbol{J}_s) \leftarrow$ TakeNSamples$(\boldsymbol{f}, \boldsymbol{J})$ ▷ Minimal N to solve least-squares
  $\hat{\boldsymbol{\xi}} \leftarrow \boldsymbol{J}_s^{\dagger}(\boldsymbol{f}_s/\Delta t)$
  $j \leftarrow 0$ ▷ Number of inliers
  for $\boldsymbol{f}_i \in \boldsymbol{f}$ & $\boldsymbol{J}_i \in \boldsymbol{J}$ **do**
    $r \leftarrow$ ComputeResidual$(\boldsymbol{f}_i, \boldsymbol{J}_i, \hat{\boldsymbol{\xi}})$
    if $r <$ threshold **then**
      $j \leftarrow j + 1$
    end if
  end for
  if $j > l$ **then**
    $l \leftarrow j$
    ${}^{C}\boldsymbol{\xi} \leftarrow \hat{\boldsymbol{\xi}}$
  end if
end for
return ${}^{C}\boldsymbol{\xi}$

assumption of points lying on the ground plane, noise, and outliers present in the flow calculation influencing the least-squares solution quality.

A simple geometric case was investigated by generating random points in front of the camera with uniformly distributed depths (±3m) around the 20m bound while the other two coordinates are chosen to fit the points in an image plane and also drawn from the uniform distribution. Next, camera is moved by an arbitrary translation and rotation and points in both cameras are projected into images and flow vectors can be calculated difference between the projected pixel locations. To mimic real-world, Gaussian noise was applied in the projection function to simulate imperfection of camera sensor, in turn, distorting the flow vectors between the points. The error magnitudes are visualized in a box plot for each of the six components of camera twist vector ${}^{C}\boldsymbol{\xi}$ in Figure 2.7.

Furthermore, to get a sense of planar surface assumption impact another simulated case was investigated. This time the lower of bound of points' distance from the camera was varied from 5m to 45m with uniform distribution (±3m) around these values and resulting MSE over full ${}^{C}\boldsymbol{\xi}$ is plotted in Figure 2.8. Obviously as the camera is getting further away from the points assumption is more and more true thus error approaches the baseline with full information of depth. This hints at the optimal height that UAV should fly to try avoiding violating the assumptions of flat ground plane.

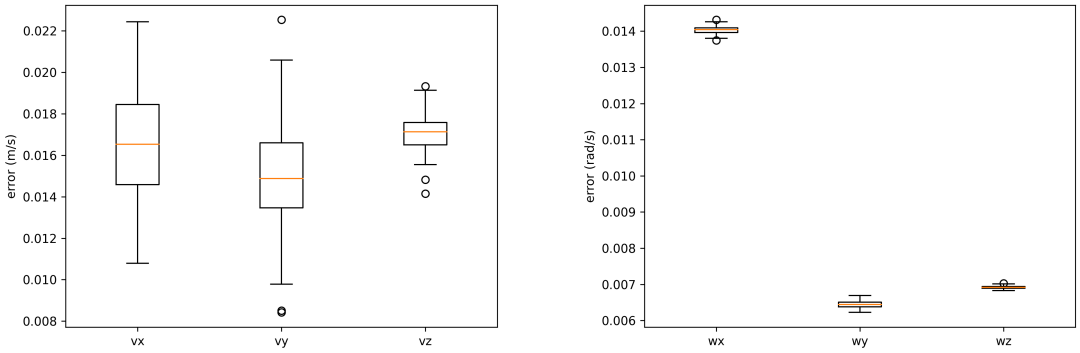


Figure 2.7: Simulation error magnitudes of ${}^{C}\boldsymbol{v}$ (left) and ${}^{C}\boldsymbol{\omega}$ (right).

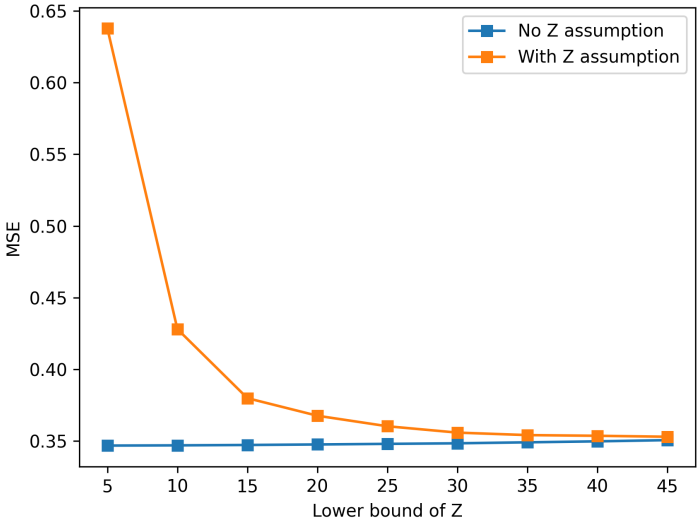


Figure 2.8: Simulation error magnitudes of ${}^{C}\boldsymbol{\xi}$ with varying lower bound of points' distance from the camera.

2.6 Sensor Fusion

The system incorporates the aforementioned visual target positioning in 3D and optical flow based velocity estimation with internal sensor values of IMU, barometer and altimeter in a Bayesian filter for localizing the UAV relative to the vessel relying only on on-board sensing and computation. All of the measurements are inherently noisy, thus the need of probabilistic framework to fuse uncertain information. The specific filter in use is the Kalman Filter [1] therefore the following paragraphs will cover the main concepts behind the filter and practical considerations when implementing a sensor fusion system.

2.6.1 Kalman Filter

In a linear discrete case, the state can be advanced with a state transition matrix $\boldsymbol{F}$ and an input matrix $\boldsymbol{G}$ as per Equation (2.67) [26, 40]. Due to model imperfection the second part of update is dealing with the state uncertainty by updating state estimate covariance matrix with pre and post multiplication with $\boldsymbol{F}$ and adding additional discretized process noise covariance $\boldsymbol{Q}$. This is a tuning parameter and is determined on case-by-case basis, where optimization techniques can be employed to improve performance of filter.

$$\begin{aligned} \hat{\boldsymbol{x}}_{n+1,n} &= \boldsymbol{F}\hat{\boldsymbol{x}}_{n,n} + \boldsymbol{G}\boldsymbol{u}_n \\ \boldsymbol{P}_{n+1,n} &= \boldsymbol{F}\boldsymbol{P}_{n,n}\boldsymbol{F}^T + \boldsymbol{Q} \end{aligned} \tag{2.67}$$

The second part of the filter deals with integrating noisy sensor measurements to update the state accordingly Equation (2.68) [26]. Where $\boldsymbol{H}$ - state to measurement mapping, $\boldsymbol{R}$ - measurement covariance (obtained from sensor datasheet or experimentally) encoding sensor noise parameters, $\boldsymbol{z}_n$ - actual measurement and $\boldsymbol{K}$ being Kalman gain.

$$\begin{aligned} \hat{\boldsymbol{x}}_{n,n} &= \hat{\boldsymbol{x}}_{n,n-1} + \boldsymbol{K}_n\left(\boldsymbol{z}_n - \boldsymbol{H}\hat{\boldsymbol{x}}_{n,n-1}\right) \\ \boldsymbol{P}_{n,n} &= \left(\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}\right)\boldsymbol{P}_{n,n-1}\left(\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}\right)^T + \boldsymbol{K}_n\boldsymbol{R}_n\boldsymbol{K}_n^T \\ \boldsymbol{K}_n &= \boldsymbol{P}_{n,n-1}\boldsymbol{H}^T\left(\boldsymbol{H}\boldsymbol{P}_{n,n-1}\boldsymbol{H}^T + \boldsymbol{R}_n\right)^{-1} \end{aligned} \tag{2.68}$$

If measurement covariance is small compared to process noise covariance i.e. sensor has minimal noise and bias characteristics, update step will trust the measurement way more and $\boldsymbol{K}$ gain will be large. On the contrary, the filter will rely on the model rather than trusting the noisy sensor. This grants two tuning knobs for the user to adjust for the specific use-case operating conditions.

In the specific case, the state vector to be estimated is modeled in Equation (2.69) where ${}^E\boldsymbol{p}_\Delta \in \mathbb{R}^3$ is the relative position between the drone and the boat, ${}^E\boldsymbol{v}_d \in \mathbb{R}^3$ is the absolute velocity of the drone, ${}^E\boldsymbol{v}_b \in \mathbb{R}^2$ is the absolute velocity of the boat planar (surface water), ${}^E\boldsymbol{a}_d \in \mathbb{R}^3$ is the acceleration of the drone and ${}^E\boldsymbol{b}_a \in \mathbb{R}^3$ is acceleration bias of the drone. Note that even though relative position here is notated in ENU frame, the origin of this vector is centered on the drone body frame, and therefore the relative part.

$$\hat{\boldsymbol{x}} = \begin{bmatrix} {}^E\boldsymbol{p}_\Delta & {}^E\boldsymbol{v}_d & {}^E\boldsymbol{v}_b & {}^E\boldsymbol{a}_d & {}^E\boldsymbol{b}_a \end{bmatrix}^T \in \mathbb{R}^{14} \tag{2.69}$$

The model of the drone dynamics and inputs to the system (in the collected experiment data) are not available thus the kinematic constant acceleration LTI (Linear Time Invariant) model Equation (2.70) is employed with ${}^E\dot{\boldsymbol{p}}_\Delta = {}^E\boldsymbol{v}_b - {}^E\boldsymbol{v}_d$ and ${}^E\dot{\boldsymbol{v}}_d = {}^E\boldsymbol{a}_d$ and full state transition matrix $\boldsymbol{A} \in \mathbb{R}^{14\times14}$. The rest of the time derivatives of the state vector are assumed to be zero. This is reasonable assumption for jerk and bias variation however tracking an agile, abruptly maneuvering boat would cause a violation obviously i.e.

rendering ${}^{E}\dot{\boldsymbol{v}}_b = \mathbf{0}$ wrong. Note that ${}^{E}\boldsymbol{v}_b$ is modeled as unknown first-order disturbance because no explicit information about vessel movement is present. In continuous state space notation this can be expressed as per Equation (2.70) where $\boldsymbol{\omega}(t) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_w)$ is white process noise.

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{\omega}(t) \\ &= \begin{bmatrix} \mathbf{0}_{3\times3} & -\boldsymbol{I}_{3\times3} & \begin{bmatrix} \boldsymbol{I}_{2\times2} & 0 \\ 0 & 0 \end{bmatrix} & \dots & \mathbf{0}_{3\times5} \\ \mathbf{0}_{3\times8} & \dots & \dots & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \dots & \dots & \mathbf{0}_{8\times14} & \dots & \dots \end{bmatrix} \boldsymbol{x}(t) + \boldsymbol{\omega}(t)\end{aligned} \tag{2.70}$$

Filter is operating in a discrete domain therefore discrete state transition and process noise matrices are obtained as per Equations (2.71) and (2.72) [40]. The Δt represent discrete time intervals between IMU measurements. Although these may not be equally spaced in time, the variation is negligible and can be ignored in the modeling part. Furthermore, the actual process noise solution here is not that important because the process noise goes through a tuning step later.

$$\boldsymbol{F} = e^{\boldsymbol{A}\Delta t} = \sum_{k=0}^{\infty} \frac{(\boldsymbol{A}\Delta t)^k}{k!} = \boldsymbol{I}_{14\times14} + \boldsymbol{A}\Delta t + \frac{(\boldsymbol{A}\Delta t)^2}{2!} + \dots \tag{2.71}$$

$$\boldsymbol{Q} = \int_0^{\Delta t} F(\tau)\boldsymbol{\Sigma}_w F^T(\tau)\, d\tau \tag{2.72}$$

The measurement vector is $\boldsymbol{z} = \begin{bmatrix} {}^{E}\boldsymbol{p}_{vis} & {}^{E}\boldsymbol{v}_{of} & {}^{E}\boldsymbol{a}_{imu} & h_{bar} & h_{alt} \end{bmatrix}^T \in \mathbb{R}^{10}$, where ${}^{E}\boldsymbol{p}_{vis} \in \mathbb{R}^2$ represents a relative position measurement performed with a camera in the 2D ground plane (still in ENU frame axes, however, the origin of the frame is centered on the body frame of the drone), ${}^{E}\boldsymbol{v}_{of}$ is velocity measurement from optical flow in ENU frame, ${}^{E}\boldsymbol{a}_{imu}$ is accelerometer reading from the IMU rotated to the ENU frame and both h_{bar} and h_{alt} provide altitude information by barometer and altimeter, respectively. Since these quantities are aligned with fixed frame axes, no rotation matrix is required in the measurement function, rendering it linear.

The measurement model $\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}_k$ with $\boldsymbol{v}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{R})$ is divided into 4 sub-models due to the different sampling rates of each sensor. Most states are simply routed through identity to their respective measured values and only acceleration has two components modelling the added bias i.e. ${}^{E}\boldsymbol{a}_{imu} = {}^{E}\boldsymbol{a}_d + \boldsymbol{b}_a$. The measurement matrices are presented in Equations (2.73) to (2.76) where each matrix corresponds to the measurement models of position $\boldsymbol{H}_{\boldsymbol{p}}$, velocity $\boldsymbol{H}_{\boldsymbol{v}}$, acceleration $\boldsymbol{H}_{\boldsymbol{a}}$ and barometer & altimeter $\boldsymbol{H}_h$ measurement models. Each model carries their associated measurement noise covariance matrix accordingly.

$$\boldsymbol{H}_{\boldsymbol{p}} = \begin{bmatrix} \boldsymbol{I}_{2\times2} & \mathbf{0}_{2\times12} \end{bmatrix} \in \mathbb{R}^{2\times14} \tag{2.73}$$

$$\boldsymbol{H}_h = \begin{bmatrix} \mathbf{0}_{1\times2} & \boldsymbol{I}_{1\times1} & \mathbf{0}_{1\times11} \end{bmatrix} \in \mathbb{R}^{1\times14} \tag{2.74}$$

$$\boldsymbol{H}_{\boldsymbol{v}} = \begin{bmatrix} \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times8} \end{bmatrix} \in \mathbb{R}^{3\times14} \tag{2.75}$$

$$\boldsymbol{H}_{\boldsymbol{a}} = \begin{bmatrix} \mathbf{0}_{3\times8} & \boldsymbol{I}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix} \in \mathbb{R}^{3\times14} \tag{2.76}$$

The observability matrix is calculated with the full stacked measurement model matrix $\boldsymbol{H} \in \mathbb{R}^{10\times14}$ and the continuous time state update matrix $\boldsymbol{A}$ as per Equation (2.77). The

matrix rank is equal to the state vector dimension $rank(\boldsymbol{O}) = 14$, thus the system is fully observable.

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{H} \\ \boldsymbol{HA} \\ \boldsymbol{HA}^2 \\ \vdots \\ \boldsymbol{HA}^{n-1} \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} \boldsymbol{H_p} \\ \boldsymbol{H_v} \\ \boldsymbol{H_a} \\ \boldsymbol{H}_h \end{bmatrix} \tag{2.77}$$

In order for this to work, all sensory input must be aligned with the ENU frame before fusion. The transformation are readily available by combining AHRS system orientation estimate discussed earlier and static transformation between sensor frame (pre-defined before an operation). For an arbitrary sensor rigidly attached to the drone body frame, the affine transformation from the sensor frame S to the body frame B is defined as per Equation (2.78) where ${}^{S}_{B}\boldsymbol{R}$ is sensor orientation with respect to body and ${}^{S}_{B}\boldsymbol{t}$ is displacement between the origins of two frames. This combined with AHRS orientation estimate ${}^{B}_{E}\boldsymbol{R}$ in Equation (2.79) can be used to transform any sensor measurement to ENU frame.

$${}^{S}_{B}\boldsymbol{T} = \begin{bmatrix} {}^{S}_{B}\boldsymbol{R}_{3\times3} & {}^{S}_{B}\boldsymbol{t}_{3\times1} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} \tag{2.78}$$

$${}^{S}_{E}\boldsymbol{T} = {}^{B}_{E}\boldsymbol{T}\,{}^{S}_{B}\boldsymbol{T} = \begin{bmatrix} {}^{B}_{E}\boldsymbol{R}_{3\times3} & \boldsymbol{0}_{3\times1} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} {}^{S}_{B}\boldsymbol{R}_{3\times3} & {}^{S}_{B}\boldsymbol{t}_{3\times1} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} \tag{2.79}$$

For example, range reading of altimeter attached to the drone can be converted to ENU frame vector ${}^{E}\boldsymbol{h}$ by Equation (2.80) where r is the range reading in altimeter sensor frame (z axis pointing downwards) and ${}^{A}_{E}\boldsymbol{T}$ is the transformation from altimeter frame to ENU frame. This can be applied to all other sensors, excluding barometer (because the reading is already measuring the absolute height), to get the measurements in unified ENU frame before fusion.

$${}^{E}\boldsymbol{h} = {}^{A}_{E}\boldsymbol{T} \cdot \begin{bmatrix} 0 & 0 & r & 1 \end{bmatrix}^{T} \tag{2.80}$$

2.6.2 Covariance tuning

The performance of the filter highly depends on the correctness of both the sensor and the process noise covariances matrices. Typically, manufacturers of sensors provide the noise parameters, which are then commonly input into the corresponding indices of the sensor covariance matrix. The process noise is a tuning parameter that can be obtained from data or trial and error. In this work both matrices were tuned using a modified version of optimization technique layed out in [28]. Several factors led to this decision: lack of information about specific sensor noise characteristics (for example, not provided in datasheet or are not quantified like in velocity measurements case), empirically observed accelerometer noise increase during flight (caused by vibration) and difficult & sub-optimal quality of hand-tuning process.

Optimization is performed against the GPS referenced position and velocity relative to the vessel. Acceleration ground truth data is obtained by simple numerical differentiation $\boldsymbol{a} = \frac{\Delta \boldsymbol{v}}{\Delta t}$. Note that all of these quantities have their own noise properties and are by no means prefect, however, they provide an accurate enough proxy to obtain a set of parameters that supersede the hand tuned ones. Then, the training dataset for a single trajectory of N time steps is constructed as a set of measurements $\{\boldsymbol{z}_k\}_{k=1}^{K}$ for each time step k and a set of reference state vectors $\{\boldsymbol{x}_k\}_{k=1}^{K}$.

The objective function is defined as the MSE between these two sets for each timestamp k as per Equation (2.81) [28] such that $\hat{\boldsymbol{Q}} \in \boldsymbol{S}_{++}^{14}$ and $\hat{\boldsymbol{R}} \in \boldsymbol{S}_{++}^{10}$ where $\boldsymbol{S}_{++}^{n}$ is a set of

all symmetric positive definite (SPD) matrices of size $n \times n$, $\hat{\boldsymbol{x}}_k$ is current estimate of the filter and $\boldsymbol{x}_k$ is the reference state vector.

$$\underset{\hat{\boldsymbol{Q}},\hat{\boldsymbol{R}}}{\operatorname{argmin}} \sum_{k=1}^{K} || \, \hat{\boldsymbol{x}}_k(\{\boldsymbol{z}_k\}_{k=1}^{K}, \hat{\boldsymbol{Q}}, \hat{\boldsymbol{R}}) - \boldsymbol{x}_k \, ||^2 \tag{2.81}$$

The constraint is achieved using Cholesky decomposition, that is, any SPD matrix can be expressed as lower & upper triangular matrices with positive diagonals as $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$. Triangular matrix is parameterized by differentiable parameters $\boldsymbol{L}(\boldsymbol{\theta})$ where $\boldsymbol{\theta} \in \mathbb{R}^{n(n+1)/2}$ with $n(n+1)/2$ being the number of non-zero elements in lower triangular matrix i.e. number of diagonal elements plus everything below them. To achieve a positive diagonal, these elements are defined as exponential of the corresponding parameters θ_i as per Equation (2.82). Both covariance matrices can be reconstructed by this parameterization as $\hat{\boldsymbol{Q}} = \boldsymbol{L}(\boldsymbol{\theta}_Q)\boldsymbol{L}(\boldsymbol{\theta}_Q)^T$ and $\hat{\boldsymbol{R}} = \boldsymbol{L}(\boldsymbol{\theta}_R)\boldsymbol{r}(\boldsymbol{\theta}_R)^T$.

$$\boldsymbol{L}(\boldsymbol{\theta}) = \begin{bmatrix} e^{\theta_1} & 0 & 0 & \dots & 0 \\ \theta_2 & e^{\theta_3} & 0 & \dots & 0 \\ \theta_4 & \theta_5 & e^{\theta_6} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{n-1} & \theta_{n-2} & \theta_{n-3} & \dots & e^{\theta_n} \end{bmatrix} \tag{2.82}$$

Due to the lack of theoretical support in [28] for not fixing the off-diagonal elements in the noise covariance matrix of the sensor, the parameterization of $\boldsymbol{R}$ was modified to simply equal the diagonal of $\boldsymbol{L}(\boldsymbol{\theta})$ according to Equation (2.83).

$$\hat{\boldsymbol{R}}(\boldsymbol{\theta}_R) = \begin{bmatrix} e^{\theta_{R_1}} & 0 & 0 & \dots & 0 \\ 0 & e^{\theta_{R_2}} & 0 & \dots & 0 \\ 0 & 0 & e^{\theta_{R_3}} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & e^{\theta_{R_n}} \end{bmatrix} \tag{2.83}$$

Optimization is performed with the PyTorch [41] automatic differentiation engine exploiting the underlying principle that computers use a set of simple arithmetic operations under the hood. Gradients of the objective function with respect to parameters $\boldsymbol{\theta}$ can be easily obtained by applying chain rule repeatedly on those simple operations. This is equivalent process as one would train a neural network described in previous sections. Optimization steps can be performed with any standard algorithm like Adam, SGD, etc. The modified training procedure is briefly summarized in Algorithm 4.

Note that training is done only on a single trajectory and later tested on all the sequences of experimental data to assess if obtained covariance matrices can generalize to unseen data, i.e. are not overfitted.

Algorithm 4 Optimized Kalman Filter (OKF). Modified version from [28].

Require: training data $\{(\boldsymbol{x}_k, \boldsymbol{z}_k)\}_{k=1}^{K}$;
$d_x \leftarrow \text{size}(\boldsymbol{x}_1), \quad d_z \leftarrow \text{size}(\boldsymbol{z}_1)$
Initialize $\boldsymbol{\theta}_Q \in \mathbb{R}^{d_x(d_x+1)/2}, \quad \boldsymbol{\theta}_R \in \mathbb{R}^{d_z}$
while training not finished do **do**
  $\hat{\boldsymbol{Q}} \leftarrow \boldsymbol{L}(\boldsymbol{\theta}_Q)\boldsymbol{L}(\boldsymbol{\theta}_Q)^T, \quad \hat{\boldsymbol{R}} \leftarrow \boldsymbol{L}(\boldsymbol{\theta}_R)\boldsymbol{L}(\boldsymbol{\theta}_R)^T$
  $\mathcal{C} \leftarrow 0$
  for k in $\boldsymbol{K}$ **do**
    Initialize $\hat{\boldsymbol{x}} \in \mathbb{R}^{d_x}$
    $\hat{\boldsymbol{x}} \leftarrow \text{KF\_predict}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{Q}})$
    $\hat{\boldsymbol{x}} \leftarrow \text{KF\_update}(\hat{\boldsymbol{x}}, \boldsymbol{z}_k, \hat{\boldsymbol{R}})$
    $\mathcal{C} \leftarrow \mathcal{C} + \text{MSE}(\hat{\boldsymbol{x}}, \boldsymbol{x}_{k,t})$
  end for
  $\boldsymbol{\theta}_Q, \boldsymbol{\theta}_R \leftarrow \text{optimization\_step}(\mathcal{C}, (\boldsymbol{\theta}_Q, \boldsymbol{\theta}_R))$
end while
return $\hat{\boldsymbol{Q}}, \hat{\boldsymbol{R}}$

2.7 Implementation Details

Work is implemented[1] in *C++* language to meet real-time runtime requirements of application enabling deployment of the discussed methods on a UAV platform and use of localization as feedback. The target platform is Linux operating system (Ubuntu distribution). Important to note that it is not suitable for hard-realtime computing application and does not give any guarantees over deterministic task scheduling. However, for all intents and purposes of research quality code these aspects can be neglected.

Code is design in a modular way allowing for easy extensions and add-ons. All sensor data are routed through ROS2 (Robot Operating System - middleware layer for intra-process communication [42]) to their respective nodes to be processed. The message passing in ROS2 works on publish/subscribe model where every node in the network can subscribe to a topic and receive relevant data. This allows for seamless integration when deploying or testing on recorded/simulation data. Even though experimental part later on is carried out on a replayed data from a *rosbag*, while in a real-world use case sensor data would be published by ROS2 drivers, the system is agnostic to that and would work the same way.

First and foremost, the lightweight Kalman filter pipeline is running in parallel independently of all other parts of the system. This pathway is not obstructed/interrupted by anything else to maintain real-time capabilities. Thus once a measurement arrives it is readily integrated into the filter. This is a central part of the system where the information arriving from post-processed or raw sensor data is fused together to obtain the final state estimate. Linear algebra operation here and other nodes are handled by an open-source library *Eigen3* [43].

IMU is the driving sensor for state estimation, dictating the frequency of the estimates because the measurements arrive at the highest rate compared to the rest of the sensors. Inertial measurements (together with magnetometer data) pass by AHRS to update attitude estimate in Madgwick Filter and ENU referenced acceleration measurement is integrated into the Kalman Filter with prediction & update steps. Additionally, an attitude estimate is published to be available for any part of the system depending on drone's orientation. Open source Madgwick filter implementation [44] - *Fusion* is used for AHRS. It is highly-optimized *C* code design to run on embedded low power processors thus has

extremely minimal overhead. The latency of the IMU measurement integration pathway will be investigated in the experimental part of the thesis.

Object detection part is coupled with visual tracking in a single process because of their interdependence. The node subscribes to an image data coming in through a ROS topic and is publishing a bounding-box if a vessel is detected in the current frame. Node is utilizing *OpenCV* - computer vision library for image processing [45] and *ONNX Runtime* - inference engine for neural networks [46].

Major challenge here is dealing with latency of the object detector. As mentioned previously, even though YOLOv7 is among the SOTA real-time detectors in terms of accuracy and speed, it is still quite heavy for CPU processing and takes up to a second to process a single frame. Consider the case on a timeline in Figure 2.9. Once a new image is captured at time T_{im_0}, the detector output is delayed by ΔT and arrives at T_{det} by the time the result passes to the filter (communication delay), and the state estimation is already at time T_{now} (driven by the IMU and other sensors). Furthermore, due to this latency the images in between cannot be processed and must be dropped. Otherwise the gap between detection and real-time state estimation would grow unbounded.

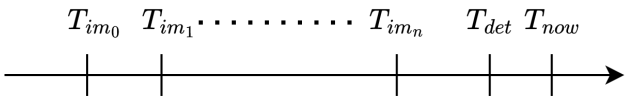


Figure 2.9: Timeline of detection.

MOSSE tracker described in previous sections could be used here to track the vessel in real-time. However, the problem still remains because MOSSE has to be initialized with a bounding box in order to start tracking. If it is initialized with detector output we are back to the same scenario as in Figure 2.9, plus having additional (although low) overhead introduced by increased processing demands.

To circumvent this, a simple technique was implemented to exploit MOSSE fast execution time (sub-image acquisition period) and "catch-up" to the current time frame from the latest available detection. There two threads running in parallel, one is responsible for detection on the latest image and the other for tracking the vessel in real-time. Once a new detection arrives a new tracker is initialized with that bounding box and carries on tracking of the bounding box over a buffer (asynchronously filled up with new frames) of past images between I_{t-n} up until the newest image I_t. The "caught-up" tracker is passed to the tracker thread and swapped in place with the old one. From this point onwards the tracker is running in real-time and providing the location of the target in the image plane with minimal latency. If a new detection is obtained the process repeats and new tracker swapped again. This is essential due to the fact that the tracker is not perfect, can drift over time, or loose track completely. The procedure is depicted in Figure 2.10.

The bounding box together with other sensory information like camera spatial velocity, altimeter, barometer, accelerometer measurements are routed to estimator node. Here data either directly or with some additional processing (like converting a bounding box to a three-dimensional relative target position) are fused into the Kalman filter. The node is implemented in a multithreaded fashion to allow parallel processing of the incoming data. This gives robustness to the system by not blocking estimation pipeline if a sensor task is delayed or data outage is experienced.

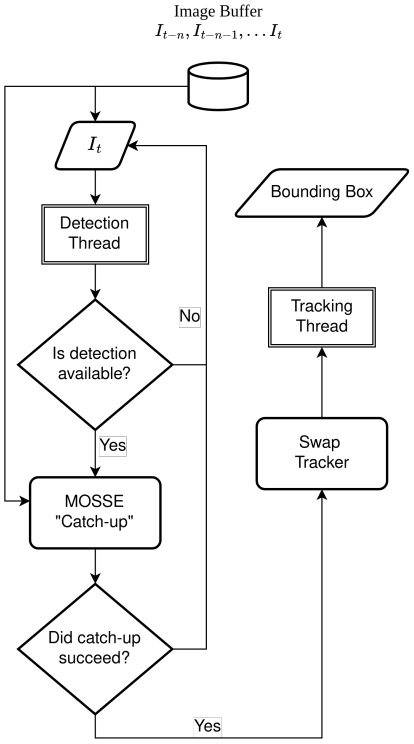


Figure 2.10: Real-time tracking process.

Chapter 3

Experiments

During development, the system was tested in simulation, offering a quick way to validate the progress. However, the simulated environment does not model the actual scenarios encountered during a real operation. In later stages of the project, a real-world data set was collected and used to further validate the system. Due to availability of field experiment dataset the focus of this chapter will be on results obtained from these tests.

3.1 Dataset Collection

This section details an experimental setup and dataset collection in an offshore coastal environment, tailored to address specific aspects of the problem formulation. The campaign, leveraging both a UAV (Unmanned Aerial Vehicle) and a surface vessel, aims to capture a wide array of data under varying conditions, thereby enabling to evaluate the performance of the proposed system in a realistic setting.

3.1.1 Setup

Central to the experimental design were two Pixhawk 6C flight controllers, one for both the UAV and the surface vessel. These controllers were tasked with capturing essential data such as IMU, GPS, compass, and barometer readings and internal PX4 state estimation (aided by GNSS) used for manual position control by the controller. Additionally, the drone had an Intel Realsense T455 camera to record a video stream during flight and an altimeter to measure the height above the surface both connected to the on-board companion computers.

Two GPS receivers and an auxiliary state estimation output allow the ground truth to be obtained for benchmarking the proposed state estimation pipeline and assessing its performance & accuracy. Although ground truth is not perfect, these data provide a good reference for comparing the two.

For data recording, a Raspberry Pi 4 was deployed on the surface vessel and a LattePanda 3 was mounted on the UAV. Both computers operated on Ubuntu 22 and featured ROS2 for communication with the flight controllers and sensor drivers on a shared network. Therefore, sensory inputs and ground-truth measurements are readily available for recording as ROS topics. The two on-board companion computers had to be synchronized in time and were tasked to record data to *rosbag2* format for later retrieval and analysis.

3.1.2 Calibration

Prior to the experiments, sensors connected to the flight controller were calibrated with the PX4 autopilot software stack, largely eliminating the need to account for static biases

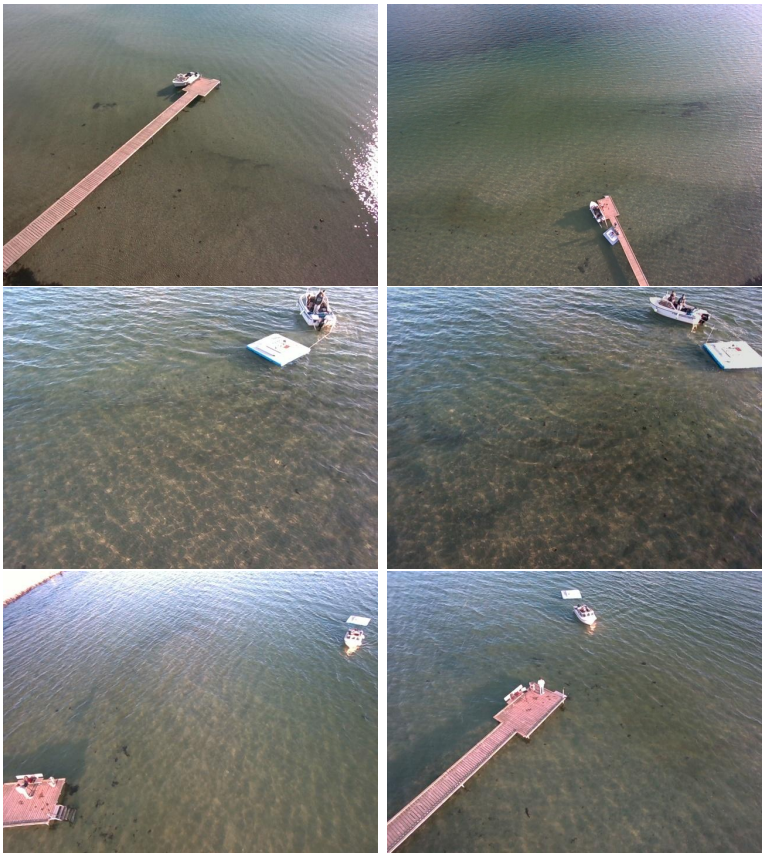


Figure 3.1: Video stream samples from dataset collection campaign.

while performing estimation. The calibration of the drone's Intel Realsense D455 camera, provided by the manufacturer, remained unchanged during the operations. Lastly, due to approximate measurements of static sensor alignment, adjustments need to be made within the filter to accommodate these inconsistencies.

3.1.3 Dataset

Data was collected in a span of 2 days in an offshore coastal environment closely resembling the scenario considered in problem formulation. Figure 3.1 shows number of samples of camera stream data during the experiments, giving a sense of the environment and the conditions in which the data were collected. The dataset, of course, does not cover full span of environments that might be encountered in offshore setting. Most notably, the water is quite shallow providing texture (although minimal) in visual data and the presence of pier (around 50% of the duration of the dataset) giving a static feature in the field of view. For instance, velocity estimation from dense optical flow could be influenced to give better estimates in the presence of the pier; however, it is empirically observed that the difference is not noticeable in the resulting measurements. Despite the mentioned traits of the video feed, conditions such as water surface movement (caused by wind) and sun reflection are present, complicating image processing and estimation tasks.

In total, 4 high-quality flight sequences (about $1-3$ minutes long each) were recorded while manually operating both vehicles. The boat is not present in the FOV of the camera at all times, violating one of the assumptions made in the beginning, and therefore subjecting the system to more adverse conditions. Two of the sequences feature a static boat parked at the dock/pier, and other two feature a moving boat. Due to limited resources (like vessel fuel, manual operation nature, etc.) and time, flight trajectories were not predetermined

with certain controlled conditions. Nevertheless, datasets contain various heights, speeds, manuevers and relative distances from the boat. This includes vessel velocities up to 1m/s, drone velocities up to 4m/s, and abrupt flight trajectories.

Lastly, as mentioned before even though the carried out experiments are conducted in an offline manner, the data are replayed at real-time rates to simulate the actual operation of the vehicle. This is done by publishing the recorded data on ROS topics and running the system on this data stream as it would be done if sensor data were coming from the actual sensor drivers during a real operation.

3.2 Results

The section presents state estimation results on the 4 mentioned sequences of data. Each with estimation of attitude (roll, pitch, yaw), relative position, drone velocity estimation, and boat velocity plotted against their respective ground truth data.

3.2.1 Ground Truth

It is important to discuss the quality and meaning of the GT data which the estimates are compared to. First, the relative Z position estimate should be considered with caution because it is taken as a difference between two GPS receivers, while the altitude sensors in the estimation measure the height from the ground. Hence some discrepancy is to be expected and important thing is that the altitude follows the trend with expected 1-2m error magnitude. Moreover, the GNSS signal is by no means perfect and has its own noise characteristics. This is illustrated in each plot with a $3\sigma = 3\sqrt{\sigma^2}$ bound (obtained from static data sequence) around the ground truth quantities. Note that for some variables, it is not attainable and not presented in the figures below. Lastly, note that attitude is compared to PX4 flight controller output which might be off from the actual orientation of the vehicle although these errors are expected to be small because of well tested and proven flight controller stack used by many commercial drones.

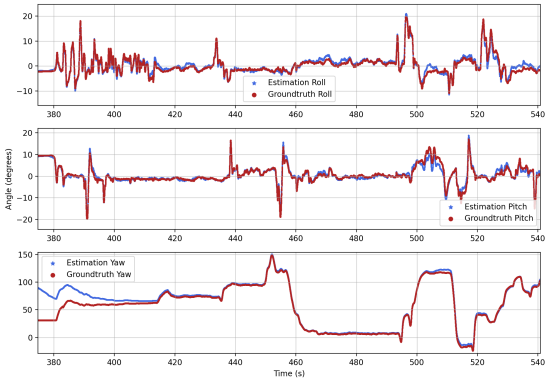
Attitude state estimates together with GT from the PX4 autopilot software suite are plotted over longer time horizon (compared to other state variables) due to the fact that state estimation is not initialized until the Madgwick filter converges after the initialization stage. In the beginning, the yaw angle starts off significantly different from the true value and, over a period on the order of magnitude of 10 seconds, it approaches the state estimate of PX4.

Additionally, relative position estimate plots have a green line on the top, indicating if a target is in the FOV of the camera. The measurement is the central information and one and only reliable reference to ground the filter. Hence the jumps in transitions from the target is not visible and track is regained again. Furthermore, they are surrounded by the bound 3σ, where the eigen values of the state covariance matrix represent the variance $\sqrt{\sigma^2}$ of each state variable, representing the uncertainty of the estimate.

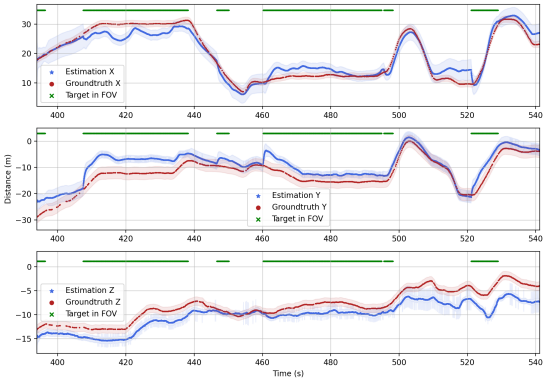
3.2.2 Experimental Results

The trajectory 1 estimation results are depicted in Figures 3.2a to 3.2d. One can see that after initialization period attitude estimation aligns well with the orientation given by PX4 controller. Although relative position estimates error does not exceed approximately the 5m bound which in offshore environment applications such as tracking is a reasonable error margin.

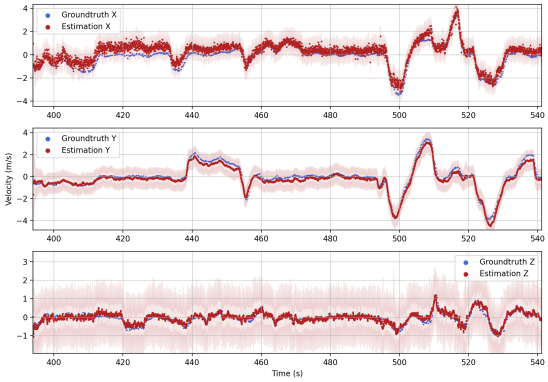
Note that even when the vessel visual track is lost (due to tracker failure) or the vessel is not in the FOV of camera state estimation can continue to track relative position with minimal deviation over prolonged periods of time. These gaps are mainly covered by velocity



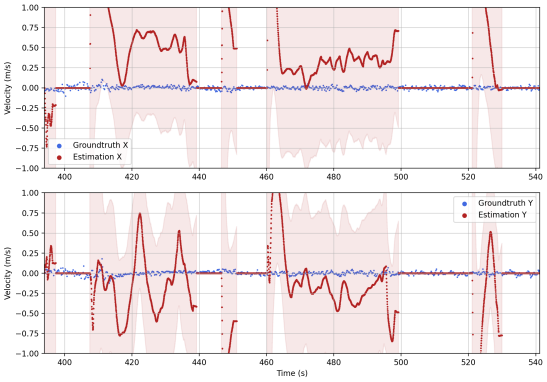
(a) Attitude estimation.



(b) Relative position estimation.



(c) Drone velocity estimation.



(d) Boat velocity estimation.

Figure 3.2: Sequence 1 results.

measurements from optical flow. Where relying solely on IMU the estimated relative position can reach errors of up to 100m in 20 seconds of lost track. This effectively lifts one of the assumptions made in the beginning on having vessel in the FOV of camera at all times. This effect is further illustrated by the example of the sequence 1 executed without drone velocity measurements in Figure 3.3 clearly showing performance degradation once the track is lost.

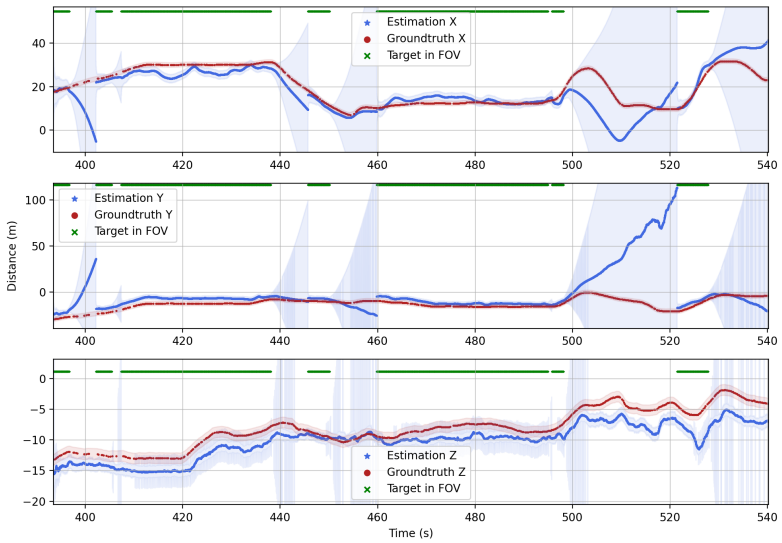
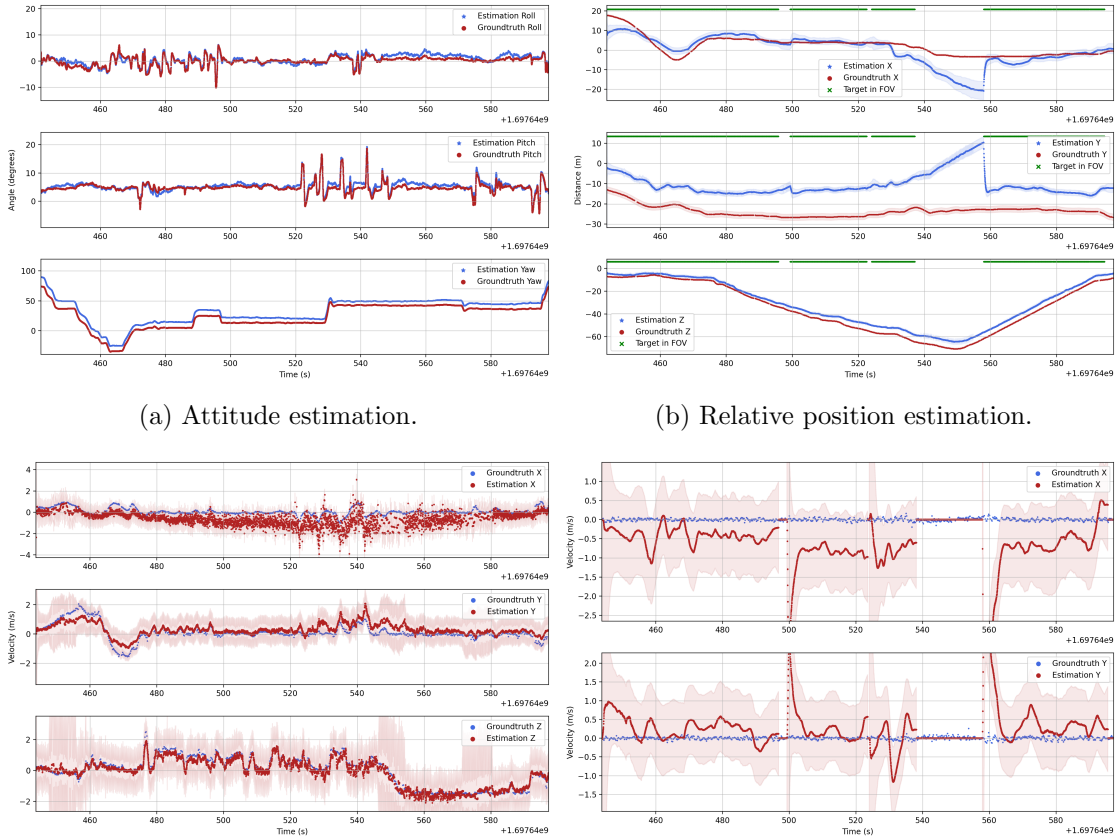


Figure 3.3: Sequence 1 relative position estimation without optical flow velocity.

The estimation of the drone velocity is confidently bounded in 3σ at all times. However, a slight systematic error can be seen in the velocity estimate, especially when the drone is almost standing still in the air. The hypothesis is that there is some residual optical flow due to surface water movement introducing a bias in the velocity estimate here. This is especially evident in the second sequence (Figure 3.4c and Figure 3.4b starting at 540s) where drone is at significant height of 60m above the ground and wind condition are more adverse. Recall that one of the assumptions made when computing velocity was that the environment largely static besides small areas of an image occupied by moving dynamic objects. While if image is mostly occupied with moving surface water and flow caused by drone motion is not dominating the flow field, then the assumption is violated thus introducing this systematic, un-modeled error. If one were to model the bias (augmenting the system with velocity bias state), these biases would not be observable. This was validated through the investigation of singular values of the observability matrix of the system. The singular values associated with velocity-bias terms are zero.

For the same reason the boat velocity estimation becomes very problematic because it is modeled as an unknown disturbance where the relative position is a combination of boat and drone velocities. Thus, it is highly influenced by the values taken by drone velocity estimates. Additionally, the errors in these terms propagate into the relative position estimate and cause large deviation once the vessel track is lost (which is the one and only information source to arrive at the velocity of the boat). In this case, the boat velocity gets stuck at the estimated values at the last relative position measurement timestamp. If there is any leftover error this gets quickly integrated into position estimates and results in system divergence. Therefore, the vessel velocity associated states are reset at these points (seen in plot as jumping to zero) and later re-established with a large estimation covariance to try to estimate it again. Discussion chapter will provide more insight on

how to mitigate these issues.



(a) Attitude estimation.

(b) Relative position estimation.

(c) Drone velocity estimation.

(d) Boat velocity estimation.

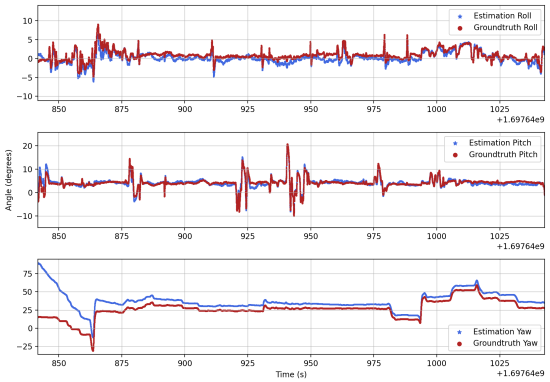
Figure 3.4: Sequence 2 results.

Another evident source of error can be seen in attitude estimation plots of trajectories 1 & 2 in Figures 3.4a and 3.5a where, after initialization, there is some offset in the yaw angle. This translates into static errors in (x, y) axes relative position estimate (see Figures 3.4b and 3.5b) because it is directly dependant on the correctness of drone's orientation. The main cause is the initialization of the Madgwick filter. The implementation requires the initialization step to run in stand still condition to allow the filter to arrive at an accurate initial yaw estimate before operation. However, the recorded sequences contain many intervals where the vessel is out of sight for periods of multiple minutes, and thus the system is launched in the air. This is mainly done for practical purposes because state estimation is started once a first measurement of relative position is taken. In real world operation this would be avoided and filter would start-off with a good estimate of orientation as for example seen in Figure 3.2a where drone start off the ground.

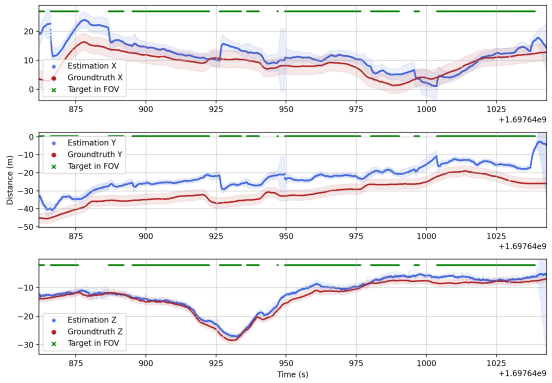
Nevertheless, due to the relative nature of the tracked position quantities, the error in real operational conditions is not a problem. As long as the yaw angle is consistent, even if it carries a constant offset the relative position is still correct from drones perspective. The deviation is only evident in the plots because the ground truth is compared to the ENU frame axes.

The rest of the sequences, with their respective estimation results, can be further investigated in Figures 3.4 to 3.6. In addition to the aforementioned sources of error, the state

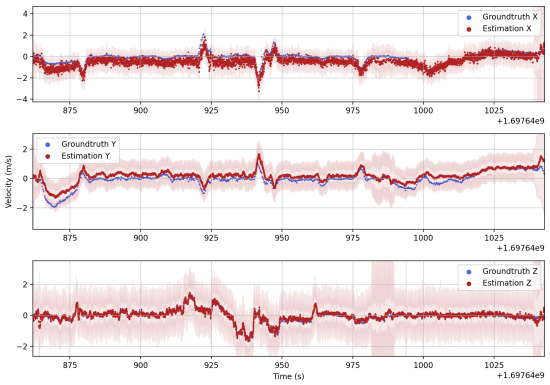
tracking is well aligned with the expected values in their respective uncertainty bounds.



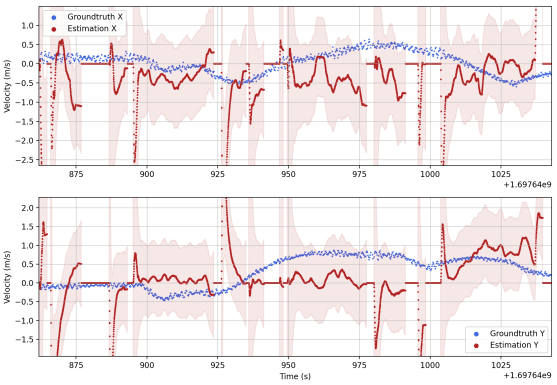
(a) Attitude estimation.



(b) Relative position estimation.

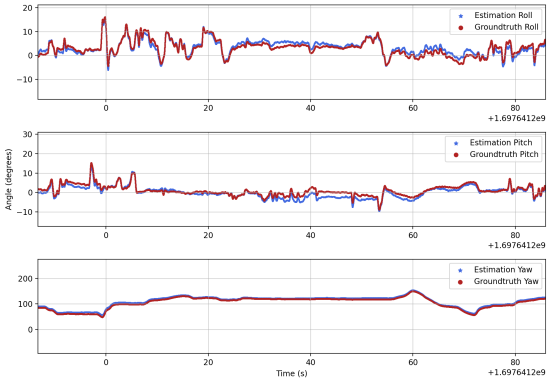


(c) Drone velocity estimation.

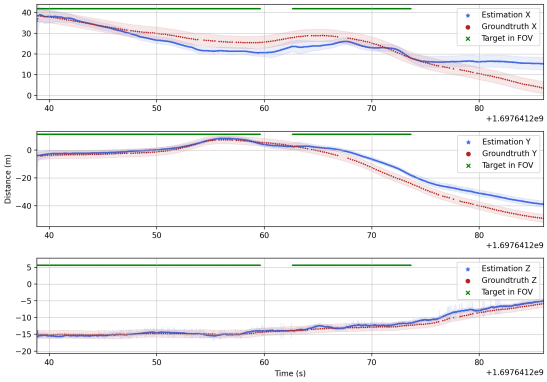


(d) Boat velocity estimation.

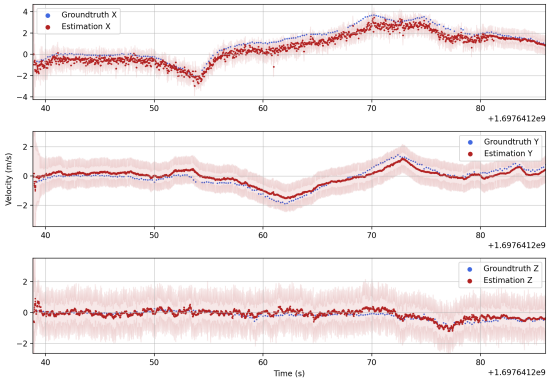
Figure 3.5: Sequence 3 results.



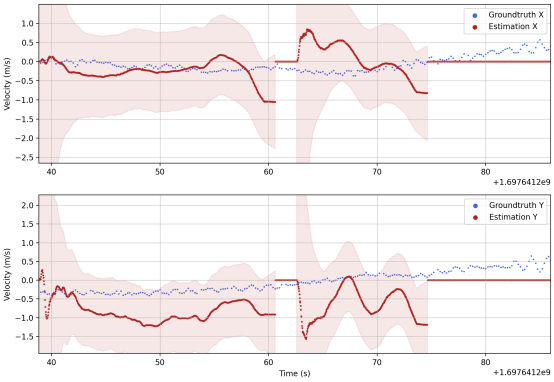
(a) Attitude estimation.



(b) Relative position estimation.



(c) Drone velocity estimation.



(d) Boat velocity estimation.

Figure 3.6: Sequence 4 results.

3.3 Computational Analysis

The computational evaluation of the system is done on a commodity laptop CPU (AMD Ryzen 7 4800H), thus it is important to asses viability of deployment of the system on companion computer CPU like Intel® Celeron® N5105, i.e., the one used on the UAV for data recording. Base clock frequency of both computing units is identical at 2.9GHz thus by measuring the CPU usage on the laptop one can get a rough estimate of the CPU usage on the companion computer. The *psutil* library [47] is used to measure the program process statistics and manual time sampling is used to assess latency of the estimation pipeline.

3.3.1 Resource Usage

The laptop has 8 cores and 16 virtual threads distributed over those physical cores, while LattePanda has the same number of physical and virtual threads which is 4. Figure 3.7 shows the histogram of CPU usage over the course of the experiment on the laptop. The average CPU usage is 1/4 of total of 16 threads, in physical terms this is 25% of the CPU. If assuming linear scaling this translates to 50% utilization of the companion computer CPU. The margin indicates that the system could be comfortably deployed on the companion computer that exhibited real-time capabilities.

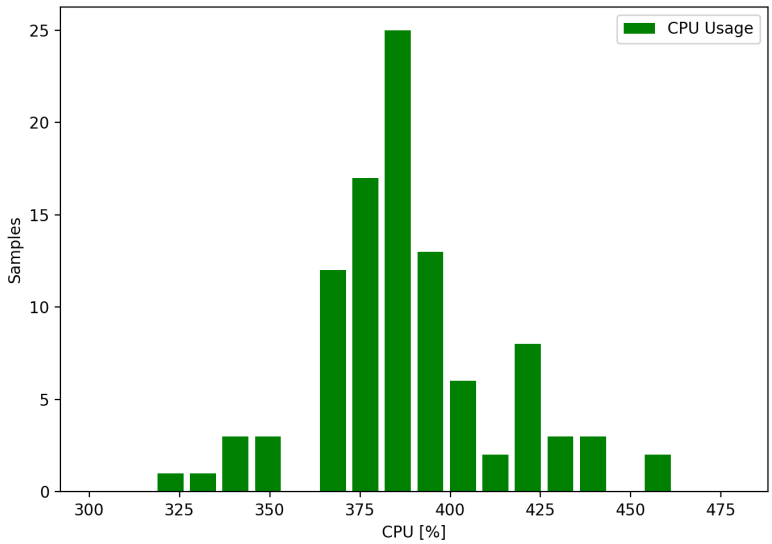


Figure 3.7: CPU usage, 100% corresponds to a single virtual thread full utilization.

The majority of CPU load is caused by parts of the vision processing. This could be decreased by lowering the quality of dense optical flow or reducing the frame rate of object detection. Even though the accuracy of estimates would be affected by these changes, this would reduce computational load, enabling the system to be deployed on resource-constrained platforms.

Another important computational dimension is the memory usage of the program. The total memory usage by state estimation pipeline is shown in Figure 3.8. It can be seen that program memory usage does not exceed 1GB of RAM which in today's standards is not a problem even on low end devices. LattePanda companion computer is equipped with 8GB of RAM thus the system would be able to fit in the limitations of the hardware.

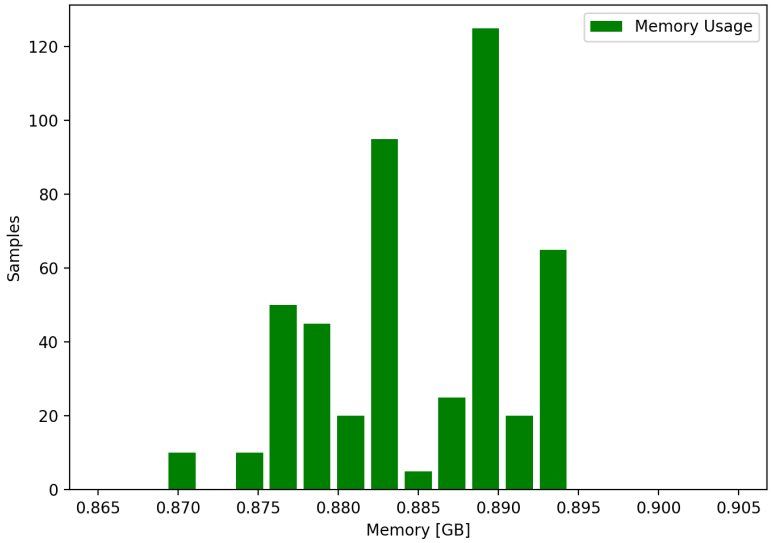


Figure 3.8: Memory (RAM) usage.

3.3.2 Estimation Latency

The latency of the estimation is of utmost importance in real-time autonomous systems to ensure safety and determinism of the system at hand. This helps downstream control tasks and safety protocols to be designed with a known delay in mind. There are two aspects of latency to be considered: the time it takes for the state estimator to respond to new sensory inputs and the uncertainty of the response delay.

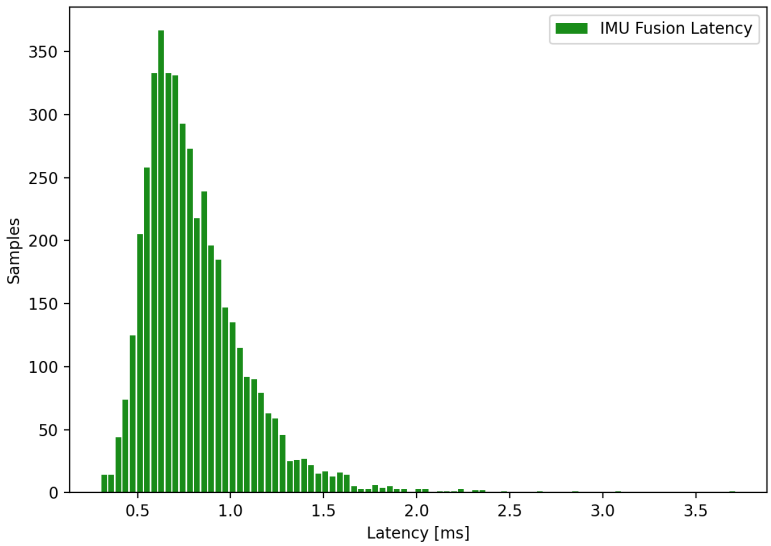


Figure 3.9: Latency of IMU fusion to the state estimator.

IMU is the highest frequency sensor thus it dictates the rate at which the state estimates are produced. Measurements were taken to assess the time spent in the IMU fusion path, i.e., from the sensor data arrival moment to the output of the state estimator. This includes both Madgwick filter orientation and Kalman filter processing times, but excludes the period it takes for data to be passed to the system from sensor drivers or replayed *rosbag*. The latter is negligible compared to state estimation pipeline runtime

and is difficult to obtain, therefore communication overhead is not considered. Figure 3.9 shows empirically derived latency distribution obtained from one of the experiments.

IMU sampling frequency is around 130 Hz, which means that the system must be able to process 130 measurements per second if it is to take all of them into account. For instance, if latency would reach 10ms then the system would be only able to deal with 100 measurements per second thus dropping around 20% of the IMU data, in turn slowing down the system. To be sure that all measurement are taken into account, a virtual boundary can be set for the worst case scenario latency to not exceed $\frac{1}{130} \approx 8$ ms. It is clear from Figure 3.9 that the worst case latency is around 3.5ms and median around 1ms giving a strong assurance that the system is able to keep up with IMU measurement frequency.

Another important aspect is width of the latency distribution. Factors such as jitter introduced by the underlying operating system scheduler impact the uncertainty in the latency distribution. Ubuntu 22 OS used for the experiments does not provide any real-time guarantees, therefore, the deviation from the mean latency is quite high and virtually unpredictable. A real-time kernel would shrink the width of the distribution and would aid the controller design ideally having nearly constant delay to compensate for. More in-depth analysis would be required for production grade system in order to ensure more determinism in the systems response time.

Chapter 4

Discussion

Final chapter presents the project outcomes, emphasizes the advantages & limitations of the system and proposes possible directions for future work to build upon.

4.1 Results

4.1.1 Advantages

The developed system has several features that make it an effective solution for the intended application. Some key properties include:

1. **Effectiveness**. The solution maintains state estimation with minimal deviation, ensuring accuracy in a GPS-denied setting.
2. **Robustness**. The system sustains state estimation even when the vessel is momentarily outside the field of view (FOV), challenging the initial assumption of constant visibility.
3. **Low Computational Complexity**. The system operates in real-time using onboard sensors and computing resources, ideal for UAV deployment.
4. **Extensibility & Generalization**. The implementation is adaptable and deployable across various UAV platforms with the specified sensor setup.

4.1.2 Limitations

A thorough understanding of limitations in the developed system is crucial for contextualizing the system's performance and guiding the future improvements. The identified constraints are as follows:

1. **Inaccuracy of UAV Model**. The UAV model used here is very simplistic and does not model the system fully. A more refined approach, incorporating a known drone dynamics model, could enhance the tracking performance.
2. **Camera Limitations in Low Light Conditions**. The camera solution implemented in this system is likely to under-perform in low-light conditions. This drawback suggests the need for additional sensors to maintain system efficacy in varying lighting environments, thus ensuring consistent performance.
3. **Unmodeled Errors in Drone Velocity Measurements**. The current design does not account for potential errors in drone velocity measurements. This oversight significantly impairs the ability to accurately track vessel velocity, thereby affecting relative position tracking accuracy and reliability.

4. **Assumption of Zero Acceleration in Vessel Movement**. The system model operates under the assumption of zero acceleration in vessel movement. This assumption may not hold for all types of vessels and maneuvers, particularly in cases of rapid acceleration or deceleration.

5. **Inability to Differentiate Multiple Vessels**. The current system is not able to distinguish between multiple vessels within the same scene. This poses a significant challenge in environments with high vessel traffic, which can lead to erroneous tracking or data interpretation.

6. **Unexplored Generalization of Detection System**. The system was only tested on a single vessel type; it is yet to be seen how well it would generalize to other vessel types. This motivates the need for further testing and refinement to ensure the system's efficacy across a diverse array of vessel types and environmental conditions.

In conclusion, while the developed system demonstrates potential, these identified limitations underscore the need for further refinement and development to enhance its applicability and reliability in complex maritime environments.

4.2 Future Work

The sections lists several possible direction for future extensions to build on top of the project and improvements on the already discussed methods.

4.2.1 Generalization to Arbitrary Object Tracking

In future endeavors, the system can be enhanced to track a diverse array of objects, alleviating its current limitation to localize relative to only a vessel. While the existing framework supports effortless swapping of the object detection model weights, an advantageous addition would be the integration of a user-friendly interface for retraining the model to recognize and track any specified object. This improvement would empower the system to adaptively track and accurately localize a UAV relative to any moving target under various GPS-denied environmental conditions. This extension would significantly broaden the applicability and utility of the system.

4.2.2 Compensation of GPS failures

As previously mentioned in motivation the system could aid traditional GPS/INS systems in case of sensor failures or spoofing. Enabling the system to retain localization capabilities after failures thus making system more robust and resilient to the conditions. Future work could investigate fallback strategies, how to integrate the system with existing state estimation pipeline, failure detection, and recovery strategies.

4.2.3 Control of the UAV

An intuitive extension of the system would be closing the loop under real operation of the vehicle and using the localization as feedback to control the UAV. This could take upon multiple dimensions:

- Developing system dynamics model & performing system identification
- Extending the system with a controller
- Utilizing an exiting control system on board of the PX4 flight controller
- Developing downstream applications (like tracking, inspection, etc.)
- Thorough testing of the efficacy of the systems under adverse conditions in the real world.

4.2.4 Systematic UAV Velocity Error

A possible direction to improve the estimation would be to estimate the systematic error in the velocity measurements. As mentioned before from observability analysis, it can be determined that if system is augmented by a bias term in the velocity measurements those are not observable in current configuration. However, there is additional information to be extracted about velocity of the vessel whenever it is in the FOV of the camera. The object bounding box combined with optical flow vectors laying inside it could be used to measure relative velocity directly expanding measurement vector $\boldsymbol{z}$ and making the problem trackable. Due to limited time, this was not implemented but is a promising approach for future work.

4.3 Conclusions

In summary, this project introduces a novel solution for GPS-denied localization of a UAV relative to a moving vessel, effectively overcoming the challenges posed by the non-inertial frame of the vessel and the homogeneity of the offshore environment. The system successfully achieves its objectives, showcasing the effectiveness of a vision-based approach, incorporating various computer vision techniques, to address the localization challenge. While there were inherent challenges and limitations, these hurdles provided valuable insights for future improvements. The work lays a solid foundation for further research to enhance the system's performance and robustness, especially under adverse conditions. Furthermore, the developed system can be used as a base to build upon its capabilities for other applications like control, tracking, inspection, etc.

Bibliography

[1] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[2] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: *IEEE Transactions on Automatic Control* 53.5 (2008), pp. 1203–1218. DOI: 10.1109/TAC.2008.923738.

[3] Sebastian Madgwick et al. "An efficient orientation filter for inertial and inertial/-magnetic sensor arrays". In: *Report x-io and University of Bristol (UK)* 25 (2010), pp. 113–118.

[4] Nicolas Carion et al. "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer. 2020, pp. 213–229.

[5] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[6] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 7464–7475.

[7] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[8] Ming Tang et al. "High-speed tracking with multi-kernel correlation filters". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4874–4883.

[9] David S. Bolme et al. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[10] Shanggang Lin, Matthew A Garratt, and Andrew J Lambert. "Monocular vision-based real-time target recognition and tracking for autonomously landing an UAV in a cluttered shipboard environment". In: *Autonomous Robots* 41 (2017), pp. 881–901.

[11] Gabryel Silva Ramos et al. "EKF-Based Vision-Assisted Target Tracking and Approaching for Autonomous UAV in Offshore Mooring Tasks". In: *IEEE Journal on Miniaturization for Air and Space Systems* 3.2 (2022), pp. 53–66.

[12] Zi-Hao Wang, Wen-Jie Chen, and Kai-Yu Qin. "Dynamic target tracking and ingressing of a small UAV using monocular sensor based on the geometric constraints". In: *Electronics* 10.16 (2021), p. 1931.

[13] Haotian Zhang et al. "Eye in the sky: Drone-based object tracking and 3d localization". In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 899–907.

[14] Xiaoyue Zhao et al. "Detection, Tracking, and Geolocation of Moving Vehicle From UAV Using Monocular Camera". In: *IEEE Access* 7 (2019), pp. 101160–101170. DOI: 10.1109/ACCESS.2019.2929760.

[15] J.-Y. Bouguet. "Pyramidal implementation of the lucas kanade feature tracker". In: 1999. URL: https://api.semanticscholar.org/CorpusID:9350588.

[16] Tobias Senst, Volker Eiselein, and Thomas Sikora. "Robust local optical flow for feature tracking". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.9 (2012), pp. 1377–1387.

[17] Tobias Senst et al. "Robust local optical flow estimation using bilinear equations for sparse motion estimation". In: *2013 IEEE International Conference on Image Processing*. IEEE. 2013, pp. 2499–2503.

[18] Tobias Senst et al. "Cross based robust local optical flow". In: *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2014, pp. 1967–1971.

[19] Tobias Senst, Jonas Geistert, and Thomas Sikora. "Robust local optical flow: Long-range motions and varying illuminations". In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2016, pp. 4478–4482.

[20] Till Kroeger et al. "Fast optical flow using dense inverse search". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer. 2016, pp. 471–488.

[21] Pyojin Kim, Hyon Lim, and H Jin Kim. "6-dof velocity estimation using rgb-d camera based on optical flow". In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2014, pp. 4008–4013.

[22] Volker Grabe, Heinrich H Bülthoff, and Paolo Robuffo Giordano. "On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 491–497.

[23] François Chaumette and Seth Hutchinson. "Visual servo control. I. Basic approaches". In: *IEEE Robotics & Automation Magazine* 13.4 (2006), pp. 82–90.

[24] Peter I Corke, Witold Jachimczyk, and Remo Pillat. *Robotics, vision and control: fundamental algorithms in MATLAB*. Vol. 73. Springer, 2011.

[25] Vincenzo Lippiello, Giuseppe Loianno, and Bruno Siciliano. "MAV indoor navigation based on a closed-form solution for absolute scale velocity estimation using optical flow and inertial data". In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE. 2011, pp. 3566–3571.

[26] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter". In: (1995).

[27] Bernt M Åkesson et al. "A tool for kalman filter tuning". In: *Computer Aided Chemical Engineering*. Vol. 24. Elsevier, 2007, pp. 859–864.

[28] Ido Greenberg, Netanel Yannay, and Shie Mannor. "Optimization or architecture: how to hack Kalman filtering". In: *Advances in Neural Information Processing Systems* (2023).

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[30] Ashish Vaswani et al. "Attention is All You Need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.

[31] Wilson Xu Weixuan. *MARITIME Dataset*. Visited on 2024-01-03. Roboflow Universe. Jan. 2022. URL: https://universe.roboflow.com/wilson_xu_weixuan-outlook-com/vais_rgb-smd-maritime-wsodd-marvel (visited on 01/03/2024).

[32] Ernestas Simutis. *Vessel Detection Dataset*. https://universe.roboflow.com/dtu-treus/project-bwpr8. Open Source Dataset. visited on 2024-01-03. 2023. URL: https://universe.roboflow.com/dtu-treus/project-bwpr8.

[33] Chun-Su Park. "2D Discrete Fourier Transform on Sliding Windows". In: *IEEE Transactions on Image Processing* 24.3 (2015), pp. 901–907. DOI: 10.1109/TIP.2015.2389627.

[34] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[35] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[36] Morten R. Hannemose. *Camera model and homographies*. 02504 Computer vision course lectures, DTU. 2022.

[37] Simon Baker and Iain Matthews. "Lucas-kanade 20 years on: A unifying framework". In: *International journal of computer vision* 56 (2004), pp. 221–255.

[38] Thomas Brox et al. "High accuracy optical flow estimation based on a theory for warping". In: *Computer Vision-ECCV 2004: 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part IV 8*. Springer. 2004, pp. 25–36.

[39] Friedrich Fraundorfer and Davide Scaramuzza. "Visual odometry: Part ii: Matching, robustness, optimization, and applications". In: *IEEE Robotics & Automation Magazine* 19.2 (2012), pp. 78–90.

[40] AIAA Awards Gala et al. "Fundamentals of Kalman Filtering-A Practical Approach". In: (2005).

[41] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[42] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[43] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[44] x-io Technologies. *Fusion*. Github. 2021. URL: https://github.com/xioTechnologies/Fusion.

[45] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[46] ONNX Runtime developers. *ONNX Runtime*. Nov. 2018. URL: https://github.com/microsoft/onnxruntime.

[47] Giampaolo Rodola. "psutil". In: URL: https://github.com/giampaolo/psutil.

Technical
University of
Denmark

Elektrovej, Building 326
2800 Kgs. Lyngby

https://electro.dtu.dk/