Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Simen Stensrød Allum

# Camera-Based Perception System for Autonomous Drones in Search-and-Rescue Missions at Sea

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2023

NTNU
Norwegian University of
Science and Technology

Simen Stensrød Allum

# Camera-Based Perception System for Autonomous Drones in Search-and-Rescue Missions at Sea

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

# Abstract

One of the largest challenges in Search And Rescue (SAR) operations at sea is to locate people in possibly vast areas. While helicopters traditionally have been used to get an aerial view while searching the sea, aerial drones have in later years become a good alternative as they become both cheaper and more capable. The drones should advantageously be autonomous to deliver valuable information to the rescue team without being a burden.

This thesis aims to develop a camera-based situational awareness perception system for an autonomous drone. The perception system is split into four main submodules; a helipad tracker, a search module, a safe landing point generator module, and a module that performs georeferencing. The helipad tracker combines two camera-based position estimators in a model-based Kalman filter with GNSS, velocity, and altitude corrections in order to track the position of a helipad during the extent of a mission. The search module detects and tracks humans and floating objects by combining a deep learning-based detector with multi-object tracking. In the case of confirmed human detection, the system georeferences the detection together with a severity level. Finally, the safe landing point generator module combines real-time image data and offline map data to locate safe landing points for the drone. The safe points are also geo-referenced.

The system is tested in real-world experiments, and the thesis findings indicate that the full perception system is able to provide valuable information. The helipad is tracked with good accuracy for the extent of drone operations. Furthermore, the search module is able to detect and track both humans and floating objects given a large enough flight altitude. In order to increase the search modules' performance, the detector should be retrained with new data to increase its altitude range and a multi-object tracker with re-identification properties should be further investigated. Safe landing points are generated at reasonable locations, however, suggestions are proposed to further increase the real-time properties of the submodule in a larger altitude range. The accuracy of the geo-referenced locations is adequate given the accuracy of the drone's GNSS measurements.

# Sammendrag

En av de største utfordringene i søk og rednings-operasjoner til sjøs er å lokalisere mennesker i mulige store områder. Mens helikoptre tradisjonelt sett har blitt brukt for å få et fugleperspektiv under søk i sjøen, har luftbårne droner de siste årene blitt et godt alternativ ettersom de har blitt både billigere og mer anvendelige. Dronene bør helst være autonome for å kunne levere verdifull informasjon til redningsteamet uten å være en byrde.

Denne oppgaven har som mål å utvikle et kamerabasert situasjonsbevisst persepsjonssystem for en autonom drone. Persepsjonssystemet er delt inn i fire hoveddeler: en helipad sporingsmodul, en søkemodul, en modul for å generere sikre landingspunkter og en modul som utfører georeferering. helipad sporingsmodulen kombinerer to kamerabaserte posisjonsestimater i et modellbasert Kalman-filter sammen med GNSS-målinger, hastighetsmålinger, og høydemålinger for å estimere posisjonen til en helipad i løpet av et oppdrag. Søkemodulen oppdager og sporer mennesker og flytende objekter ved å kombinere en dyp læring-basert detektor med flerobjektsporing. I tilfelle der det blir bekreftet en menneskedeteksjon, georeferer systemet deteksjonen sammen med en alvorlighetsgrad. Til slutt kombinerer modulen for sikre landingspunkter sanntidsbildeinformasjon og kartdata for å finne trygge landingspunkter for dronen. De trygge landingspunktene blir også georeferert.

Systemet er testet i eksperimenter ute, og funnene i oppgaven indikerer at det fullstendige persepsjonssystemet er i stand til å levere verdifull informasjon. Helipadens posisjon estimeres med god nøyaktighet i løpet av dronens oppdrag. Videre er søkemodulen i stand til å oppdage og spore både mennesker og flytende objekter ved tilstrekkelig høy flygehøyde. For å øke søkemodulens ytelse bør detektoren trenes med nye data for å øke dens deteksjonsegenskaper i et større høydeintervall, og en flerobjekts-sporer med reidentifikasjonsegenskaper bør undersøkes videre. Sikre landingspunkter genereres på hensiktsmessige steder, men det foreslås forbedringer for å øke sanntidsegenskapene til modulen i et større høydeintervall. Nøyaktigheten til de georefererte posisjonene er tilfredsstillende gitt nøyaktigheten til dronens GNSS-målinger.

# Preface

This thesis was written in the spring of 2023 and is the conclusion of my master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway.

During the last two years of my studies, I have specialized in autonomous systems, an area I find very exciting. The choice, therefore, fell easily on a thesis where I could combine my areas of interest; computer vision and state estimation in a physical system. The aim of this task is to create a perception system for an autonomous drone. The perception system is intended to be used in combination with a mission planner who makes choices based on the information that the perception system can provide.

The work in this thesis is a continuation of the work carried out in their respective master's theses by Thomas Sundvoll [1], Peter Bull Hove [2], and Martin Falang [3]. All three explored the use of computer vision-based methods to estimate a drone's position during the autonomous landing of a drone on a helipad. The work in this thesis is also a direct continuation of my specialization project [4]. In the specialization project, I ported the code developed by Martin Falang [3] to a newer version of ROS and Python. Furthermore, a new communication interface between the Anafi drone and ROS was developed in collaboration with Øystein Solbø in his specialization project [5], which uses an updated version of the communication API of the Anafi drone. Furthermore, I improved parts of the position estimation algorithm developed by Falang by, for example, adding fiducial markers to the helipad and the position estimation algorithm gained more global properties by including GNSS measurements in the position estimation.

The work in this thesis was conducted under the supervision of Anastasios Lekkas from the Department of Engineering Cybernetics (ITK) at NTNU. He has contributed to the direction and goals of the thesis and has offered helpful advice throughout the process. Paolo De Petris from ITK at NTNU helped establish the fundamental operations of the drone lab at NTNU and its motion capture system. Stefano Brevik Bertelli from the ITK at NTNU gave me access to a laminating machine as well as laminating sheets, and the janitors at Elektrobygget at NTNU Gløhauen were generously able to provide me with two A3-sized laminating sheets. Bertelli also assisted me in locating an immersion suit that I could borrow in order to test how well the system could detect humans in the frigid waters off Trondheim. The department has provided me with two Parrot Anafi FPV drones and a Komplett Khameleon P9 Pro laptop. They also gave me access to an office with a Dell OptiPlex 7040 workstation computer which I have used to write this thesis. When testing outside without access to power outlets, Glenn Angell and the others working in the mechanical workshop at Elektrobygget at NTNU Gløshaugen kindly provided me with a portable generator. Tom Arne Pedersen from DNV generously assisted with

tests at sea and provided me with access to the DNV ReVolt vessel for testing with the helipad mounted on a boat. Jakob Horn Jernsletten and Øystein Solbø assisted in various tests at sea.

The Parrot Anafi FPV drones are used in conjunction with Parrot's Olympe API 7.5.0 software. In addition, a number of open-source software modules are used in this thesis. While not exclusively, it consists of ROS Noetic Ninjemys, Python 3.8.10, NumPy 1.23.5, SciPy 1.9.1, apriltag 0.0.16, seaborn 0.12.0, ultralytics 8.0.16, OpenCv 4.2.0, matplotlib 3.6.2, pyproj 3.4.0, osmnx 1.2.2, pandas 1.5.3, geopandas 0.12.2, and shapely 1.8.5.post1.

The reader is expected to have some basic background knowledge of the topics covered. All figures are made by the author unless otherwise stated.

# Acknowledgments

I would like to thank my supervisor, Anastasios Lekkas, for his guidance and oversight throughout the last two semesters. Additionally, a big thanks to Øystein Solbø, who, in parallel with this thesis, developed control and AI planning code for the Anafi drone. He has always been a huge asset when discussing how perception and planning are intertwined. Thanks to Tom Arne Pedersen from DNV who gave me access to ReVolt and helped during the testing of the system. Thanks to Stefano Bertelli, Paolo De Petris, and Glenn Angell for helping to provide the equipment used during testing. I would also like to thank my classmates for all the discussions and great collaboration during our studies at NTNU. Finally, I would like to thank my family and friends for all their support.

# Contents

# Figures

# Tables

# Acronyms

**AI** Artificial Intelligence. 2, 12

**ANN** Artificial Neural Network. 12–16, 23, 25, 26, 36

**API** Application Programming Interface. vii, viii, 3, 16, 22, 23, 31, 33, 38

**CNN** Convolutional Neural Network. 2, 14–16, 19

**CV** Constant Velocity. 2, 18, 31, 46, 49, 63, 78, 80, 88

**DNN-CV** Deep Neural Network Computer Vision. 32, 60–62, 77, 78, 88

**EKF** Extended Kalman Filter. 11, 23, 31, 49–51, 61, 63, 74, 78, 88, 89

**EWMA** Exponential Weighted Moving Average. 38–40, 67, 85, 87

**FOV** Field Of View. 22, 44, 45, 50, 51, 81, 82, 84, 88

**GNSS** Global Navigation Satellite System. vii, 3, 8, 22, 25, 33, 41, 42, 45, 46, 49–51, 59, 63, 70, 74, 78, 82, 84, 88, 89

**GPU** Graphics Processing Unit. 14, 23, 36, 37, 40, 81, 86

**IOU** Intersection-Over-Union. 18, 19, 38, 47, 48, 80

**KF** Kalman Filter. 2, 3, 7, 10, 11, 18, 39, 46, 49, 80, 88

**mAP** Mean Average Precision. 15, 38, 49, 53, 54, 69

**MOT** Multi Object Tracking. 4, 5, 18, 38, 43, 49, 53, 56, 81, 87

**NED** North, East, Down. 5, 8, 31, 33, 43, 45, 49, 59, 67, 78, 81, 89

**NMS** Non Maximum Suppression. 16

**ROS** Robot Operating System. vii, xvii, 3, 5, 23–25, 27, 31–34, 38–41, 45, 47, 86–89

**SAR** Search And Rescue. iii, 1–5, 25, 32, 35, 38, 80, 82, 84, 85, 91

**TCV** Traditional Computer Vision. 2, 3

**UAV** Unmanned Aerial Vehicle. 1–4, 7

**YOLO** You Only Look Once. 3–5, 15, 16, 18, 35–38, 40, 41, 47, 48, 53, 59, 61, 74, 77–81, 85–89

# Chapter 1

# Introduction

## 1.1 Background and Motivation

The goal of Search And Rescue (SAR) operations is to locate and help people in need or in immediate danger. Examples of various SAR operations types include combat SAR, urban SAR, mountain rescue, and air-sea rescue [6]. The main difficulty in SAR operations is locating victims in the shortest time possible. The "golden day" rule is a general guideline for SAR operations at sea, which posits that the initial 24 hours following a rescue request present the greatest opportunity for a successful rescue [7]. It is challenging to locate people because most SAR operations are carried out in vast areas, such as mountains or the sea. Hence, in SAR operations, localizations of potential victims or objects of interest are fundamental, and the time spent on detection and localization is crucial for saving lives. The likelihood of saving lives decreases with the time it takes to find a victim [7]. To plan the search activities at sea and liaise between all sea and air actors in the search, the International Aeronautical and Maritime Search and Rescue (IAMSAR) guidelines are followed [8]. The actors in SAR operations at sea are rescue vessels. While there aren't many obstacles in open water, several things, like waves, restrict a boat's field of vision. An aerial view is therefore necessary. Helicopters have historically been used to obtain this aerial perspective. Using potentially multiple helicopters in SAR operations necessitates the addition of extra personnel and significantly raises the mission's cost. Other factors may limit the use of helicopters in SAR operations, and the authors of [9] point out that in the period of 2000-2010, approximately 25% of all requested missions were not completed due to weather conditions, technical problems, duty time regulations, or a concurrent operation.

Using autonomous Unmanned Aerial Vehicle (UAV), also known as drones, could be a way to conduct aerial searches without tying up the limited helicopter resources. In addition to being piloted by a human operator, drones can also be programmed to fly autonomously and perform tasks. Autonomous programmed drones can then deliver valuable information while requiring little human interaction, freeing up precious resources. The drones can carry out a variety of jobs, including those that humans would consider dirty, dull, or dangerous, like the search and detection phase of a SAR mission [10]. The manual detection of people in aerial images is very laborious and prone to error, as the authors of [11] note, and it can therefore be beneficial to use a drone for this detection. Due to their high agility, portability, and aerial access, drones have been used in SAR missions for many years. They can quickly cover large areas, and most are equipped with high-resolution cameras that make it possible to analyze the image content using sophisticated algorithms. Additionally, because the drones can be oper-

ated from distant, secure locations, they lessen the risk for the operator during SAR operations [10].

Advanced detection and classification algorithms for image analysis are largely available as Artificial Intelligence (AI) technology develops. Deep learning methods based on deep neural networks have successfully handled computer vision tasks that were previously challenging because of limited computing power or data access. These neural network-based techniques offer accurate results and are now acknowledged as the established standard for UAV-based perception systems [12]. Real-time image analysis is possible with one-stage methods like YOLO [13]. Although deep learning methods produce useful results, the research area remains largely unexplored for detection in marine environments. Processing maritime images presents several challenges such as dynamic background, constant tides and waves, weather-related lighting, water reflections, the presence of small floating objects, and the unpredictable movement of marine objects [12].

A thorough understanding of the situation in the drone's environment is necessary for the drone to use the output of the perception system in potentially sophisticated AI planning algorithms. A functional perception system for an autonomous drone must include situational awareness as a key component. It alludes to the drone's ability to perceive its surroundings and spot potential threats. This situational awareness can be implemented in SAR operations through the creation of maps of the drone's surroundings, the tracking of scene elements like boats, people, and obstacles, the separation of safe and unsafe areas, and the observation of actor behavior [14].

## 1.2 Previous work

### 1.2.1 Previous project development

The work presented in this thesis builds upon the work presented in their respective master's theses by Martin Falang, Peter Bull Hove, and Thomas Sundvoll [1–3], as well as the individual specialization projects written by Øystein Solbø [5] and the author [4] before this thesis.

A drone helipad was developed by Sundvoll in [1] and was intended to be mounted on top of DNV's autonomous research vessel ReVolt[1]. The helipad had distinctive features that a perception algorithm could recognize. To determine the helipad's pose (Position and Orientation), Sundvoll employed more Traditional Computer Vision (TCV) techniques like color segmentation, edge detection, and corner detection. The performance of the perception-based pose estimation system was constrained in real-world experiments because of the lighting and image noise.

To create a more robust system, Hove [2] trained a deep Convolutional Neural Network (CNN) to detect the helipad and then combined the TCV output with the detected helipad's pose estimate from the CNN module in a Constant Velocity (CV) Kalman Filter (KF). Even though the KF output was noisier, it was more reliant than using the TCV output alone.

[1] `https://www.dnv.com/technology-innovation/revolt/index.html`

Falang [3] upgraded the drone system from the previously employed AR.Drone 2.0 to a more modern Anafi FPV both from the French drone company Parrot[2]. Falang also developed an updated pose-estimation algorithm based on the old TCV approach. The algorithm involved circle detection, corner detection, pattern matching, and homography-based pose estimation. The new algorithm replaced the old TCV pose estimation algorithm in the KF. The new algorithm was computationally heavy and had offsets along all axes. The final output from the KF gave promising results in outdoor and indoor experiments, but only when the helipad was stationary and not subjected to motions.

The author's specialization project [4] serves as the foundation for the work in this thesis. In order to extend the support for the developed code, the portion of the codebase focusing on the perception that Falang [3] developed was ported to a newer version of Robot Operating System (ROS) and Ubuntu, namely ROS Noetic and Ubuntu 20.04. Additionally, a new ROS bridge for the Anafi FPV drone was co-authored with Solbø [5]. The new ROS-bridge served as an interface between the new version of the Parrot Olympe API and ROS. Additionally, the author created a method for precise position estimation. A new system built around the detection of fiducial markers, AprilTags [15], replaced the TCV strategy Falang developed in [3]. The new sub-system allowed for faster and more precise position estimation, however, it had minor problems related to detection altitude and difficult light conditions. The operational range of the position-estimation system was increased by incorporating Global Navigation Satellite System (GNSS) measurements and Anafi's internal altitude measurement into the KF.

### 1.2.2 UAVs in SAR operations

UAVs have been utilized in SAR operations for several years, and the research within the field is large. In [16] Rizk et al. are using an UAV fitted with a digital camera and a GNSS receiver. The images are analyzed using a trained YOLOv3 network. When humans are detected in the images, the system sends a notification packet to the rescue team with the drone ID, date, time, latitude, longitude, and the number of detected human bodies with the corresponding coordinates of the bounding boxes and confidence ratio.

The same ground concept is implemented in [17] where a YOLOv5 network is trained for the detection of humans in flooded regions. When a human is detected, the GNSS coordinates of humans are marked on a map, and their locations are e-mailed to the rescue team.

A YOLOv4 and YOLOV4 tiny networks are trained on top-view images of marine objects in [12]. The models are deployed on small-size low-weight Nvidia edge devices mounted on drones.

Lygouras et al. [18] are using a fully autonomous UAV to detect and rescue swimmers. The drone autonomously navigates to a distressed coordinate, searches for swimmers using a trained YOLOv3 tiny network, and if swimmers are detected releases a life buoy.

In [19], to identify human life signs in disaster areas, the authors are using a drone equipped with an RGB camera. The authors take advantage of the fact that changes in reflected intensity values in image sequences are directly caused by motions of the chest wall caused by cardiopulmonary activity for live subjects.

Feraru et al. [20] are using a UAV with a thermal camera in order to detect persons in water during man overboard accidents at sea. A Faster R-CNN network is trained with transfer learning to detect persons in the water. Once a potential person is detected, the detection is validated by a human-in-the-loop.

[2] `https://www.parrot.com/en`

### 1.2.3 Situational Awareness for UAVs

The authors of [14] address situational awareness for drones. The topic is broken down into more minor problems that can be addressed separately to give UAVs situation awareness from a top-down visual perspective. From a low level of awareness to a higher level of understanding, each subproblem raises awareness to a different level. Semantic segmentation, object detection, action detection, event recognition, and anomaly detection are the five levels defined by the authors. All are resolved using various Deep Learning methods and subsequently combined into a single model using Multi-Task Learning.

Geraldes et al. [21] present the Person-Action-Locator situational awareness system, which is used for tasks like SAR. The three main parts of the system are an image processing system used to calculate the Global Positioning System (GPS) position of humans, a deep learning system that detects humans and analyzes their actions, and a visualization system that plots detected humans and their actions on a map. On-board edge computing systems are used for all calculations.

Vasilopoulos et al. [22] extend the object detection in marine environments by also incorporating tracking of detected objects. The tracking of detected objects is performed by using the *Deep SORT* algorithm presented in [23]. Bounding boxes from a trained YOLOv5 network are fed into the Deep SORT algorithm which combines Kalman Filtering and the Hungarian method together with deep appearance information to track bounding boxes over frames. Several papers have explored Multi Object Tracking (MOT) from aerial imagery for urban scenery [24–27] in the same way as Vasilopoulos et al. have done for marine scenery using detectors and Deep SORT tracker.

Lenka et al. [28] are focusing on determining the extent of flooding by examining aerial photographs taken by a UAV. The authors use textural features required by local binary patterns rather than deep learning to identify the flooded zones. The textural features were combined with K-means segmentation to identify the flooded areas. Zaffaroni et al. [29] are using deep learning to identify water segments for the same purpose of water segmentation in the case of floods. On a unique dataset created for water segmentation, various neural network architectures were trained and evaluated.

In [30] the use of attention-based semantic segmentation on aerial datasets from UAVs to assess the damage caused by natural disasters is investigated. Instead of focusing on images, the authors of [31] concentrate on the segmentation of aerial videos from UAVs. They use BiSeNet [32], which was trained on a dataset of sequentially labeled aerial images and videos taken by UAVs, to accomplish this.

## 1.3 Objectives

This thesis' primary objective is to investigate how perception systems can enable search-and-rescue operations using autonomous unmanned aerial vehicles (UAVs). Within this broad context, this thesis aims to create a camera-based perception system for an autonomous drone for the use case of maritime SAR missions. The perception system should be capable of identifying objects of interest, providing their positions, and supplying simple situational awareness for the drone. The perception system should provide precise position estimates with respect to a helipad during mission execution and when the drone is landing on the helipad. The information from the perception system should be valuable for a mission planner for autonomous drones.

## 1.4 Contributions

The contributions of this thesis are as follows:

- Aggregated two publicly available datasets of objects in marine environment into a single dataset for the training of detection networks.
- Created a publicly available dataset consisting of 2014 manually annotated images of a helipad.
- Wrapped two different Multi Object Tracking (MOT) into ROS1 and tested both for the purpose of tracking objects in marine environments.
- Trained two different YOLOv8 networks using transfer learning from pre-trained YOLOv8 models. One for the detection of a helipad, and one for the detection of humans and floating objects.
- Wrapped the YOLOv8 framework into a public available ROS1 wrapper.
- Updated and improved a sensor-fusion system based on a constant velocity Kalman filter for precise tracking of a helipad.
- Trained two different deep-learning network structures for the purpose of sea-land segmentation.
- Providing safe points in a world NED frame for the drone to land during flight based on both real-time and offline segmentation masks.
- Created a module for geo-referencing points in images.
- Combined the modules mentioned above in a full perception system able to provide valuable information during a search-and-rescue mission at sea using autonomous drones.

Two publicly available datasets of objects in the marine environment were aggregated into a single dataset. In addition, a new dataset consisting of 2014 manually annotated images of a helipad was created and made publicly available. To track the detected objects in marine environments, two different multi-object tracking algorithms were integrated into the Robot Operating System (ROS) framework and tested. Transfer learning was applied to train two distinct YOLOv8 networks. One network was trained for detecting a helipad, while the other focused on identifying humans and floating objects. Additionally, a ROS wrapper was developed to enable easy integration of the YOLOv8 framework. Improvements were made to a sensor-fusion system from the author's project thesis in [4]. These enhancements enhanced the system's precision in tracking the helipad accurately. To address the task of sea-land segmentation, two different deep-learning network structures were trained. The segmentation network output was used in combination with an offline map segmentation to provide the drone with robust and real-time-based safe landing points in a world NED frame during flight. A module was created to geo-reference points in images, enabling localization and mapping of objects detected by the system. All the aforementioned modules were combined to form a perception system designed explicitly for SAR missions at sea using autonomous drones. This integrated system provides valuable information for a possible planning system for autonomous drones in marine environments.

## 1.5 Outline

Chapter 2 of this thesis introduces the theory and background material for the work done in this thesis. After Chapter 3 presents the experimental setup used in this thesis, Chapter 4 discusses the construction techniques used to create the various subsystems. The results of testing the perception system both individually and collectively are provided in Chapter 5 along with a

description of the results. These results are examined and discussed in Chapter 6, and the thesis is then concluded in Chapter 7 with a few closing remarks that point to possible future research on the subject.

# Chapter 2

# Theory

The underlying theory behind the concepts in this thesis is presented in this chapter. The operation of a quadcopter, the frames of reference used in this thesis, and the conversion between them are all described at the outset of the chapter. The chapter continues by presenting a camera model and discussing the fundamental idea of using a camera as a sensor. After outlining the fundamentals of the Kalman Filter (KF), the chapter delves deeply into deep learning. Topics such as neural networks and segmentation are presented, after which the chapter closes with a description of multi-object tracking.

## 2.1 Quadcopters

This section is based on Section 2.1 from the specialization project [4]. A quadcopter is a class of helicopters with four rotors. Quadcopters can also be Unmanned Aerial Vehicle (UAV), then often with rotors mounted in an X-shape, parallel to the ground. A sketch of a quadcopter can be seen in Figure 2.1. Because a quadcopter has fixed-pitch blades as opposed to conventional helicopters, the attitude of the aircraft is managed by adjusting the propeller speed [33]. Each of the four propellers is controlled by supplying varying voltages to DC motors. Each of the four propellers generates thrust proportional to the square of the propeller rotation speed [34], denoted as $F_i$, in Figure 2.1. Angular moments are also produced by the propellers. The non-diagonal propellers rotate in the opposite direction to cancel the angular moments and control the quadcopter's attitude in the horizontal plane. By adjusting the force produced by each of the four propellers, the quadcopter's attitude can be altered. The attitude is then used to control the quadcopter's position.

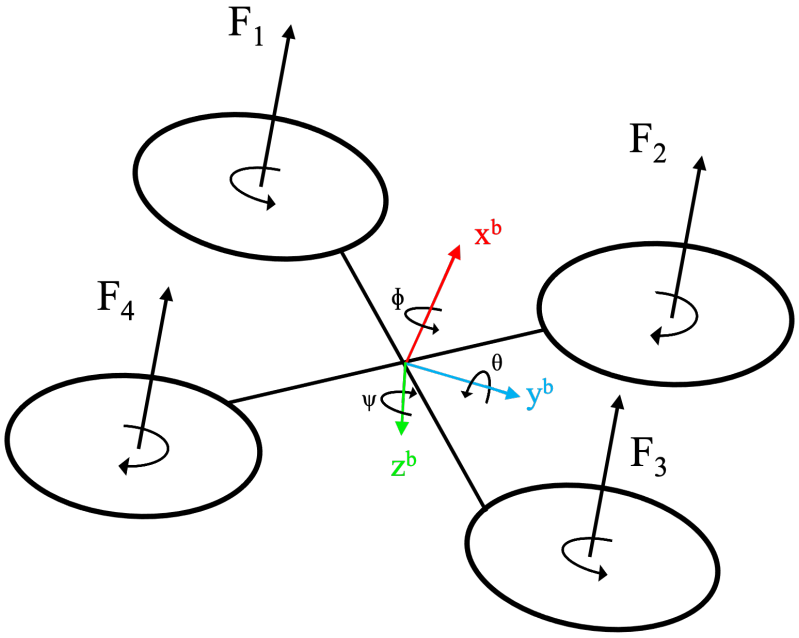



**Figure 2.1:** Model of the quadcopter. Figure from [4], and inspired by [3]. Revised with angle definitions in roll and yaw according to the right-hand rule.

### 2.1.1 Frames of reference

There are three different main frames of reference used in this thesis in addition to the WGS84 system [35] used to describe GNSS coordinates. The position and attitude of the quadcopter are described in the first frame, which is a world frame. The helipad's center is where the frame is defined as a tangent plane to the earth's surface. The earth's surface's curvature is neglected for this thesis. The frame is a North, East, Down (NED) frame, denoted by the letter $\{n\}$, and it uses the same axis definitions as the one Fossen is using in [36]: $x^n$ points towards true north, $y^n$ points towards true east, and $z^n$ pointing downwards normal to earth's surface.

A body frame that is attached to the quadcopter serves as the second frame of reference. The quadcopter's linear and angular velocities are expressed in this frame, denoted $\{b\}$. The definition of this frame's axis is the same as that of those used by Fossen [36]. $x^b$ is the longitude axis, pointing from the back to the front nose of the quadcopter. $y^b$ is the transversal axis, pointing to the right from the longitude axis. Lastly, $z^b$ is the normal axis, directed downwards through the quadcopter. The body frame axis is illustrated in Figure 2.1.

The camera frame serves as the third and final main frame of reference. Objects visible in images captured with the quadcopter's camera are described in this frame. The frame is denoted $\{c\}$. The image plane's center serves as its origin. Additionally, this frame's axis is defined in the same manner as it is in [37] with $x^c$ being right in the image plane, $y^c$ being down in the image plane, and $z^c$ pointing straight out of the image plane.

To transform between the frames of references this thesis is using the same notation as the one Fossen is using in [36]. The rotation matrix $\mathbf{R}$ between two frames of reference $\{a\}$ and $\{b\}$ is denoted $\mathbf{R}_b^a$. The full transformation of a vector $\mathbf{v}$, including both rotation and translation, given in frame $\{a\}$ to frame $\{b\}$ is then given as:

$$\mathbf{v}^b = \mathbf{R}_b^a \mathbf{v}^a + \mathbf{t}_b^a. \tag{2.1}$$

In (2.1), $\mathbf{t}_b^a$ is the translation from frame frame {b} to frame {a} given in frame {a}.

## 2.2 Camera modeling

In this thesis, a three-dimensional world is mathematically projected into a two-dimensional world by a widely known camera image formation model, the central perspective model [38]. Drawing a ray from a point in the three-dimensional world, represented as (X, Y, Z) in Figure 2.2, through a fixed point in space, the center of projection $O_c$, an image of the point is represented by the ray's intersection with an image plane, (u,v). Each point on the image plane will collect light from a cone of rays rather than just a single ray because the camera's optical center will have a finite size in reality. In addition, cameras frequently come with lenses, which further complicates matters. The model can therefore be said to be a mathematical simplification of the image-forming process. Nevertheless, the model often provides acceptable approximations of reality [39].

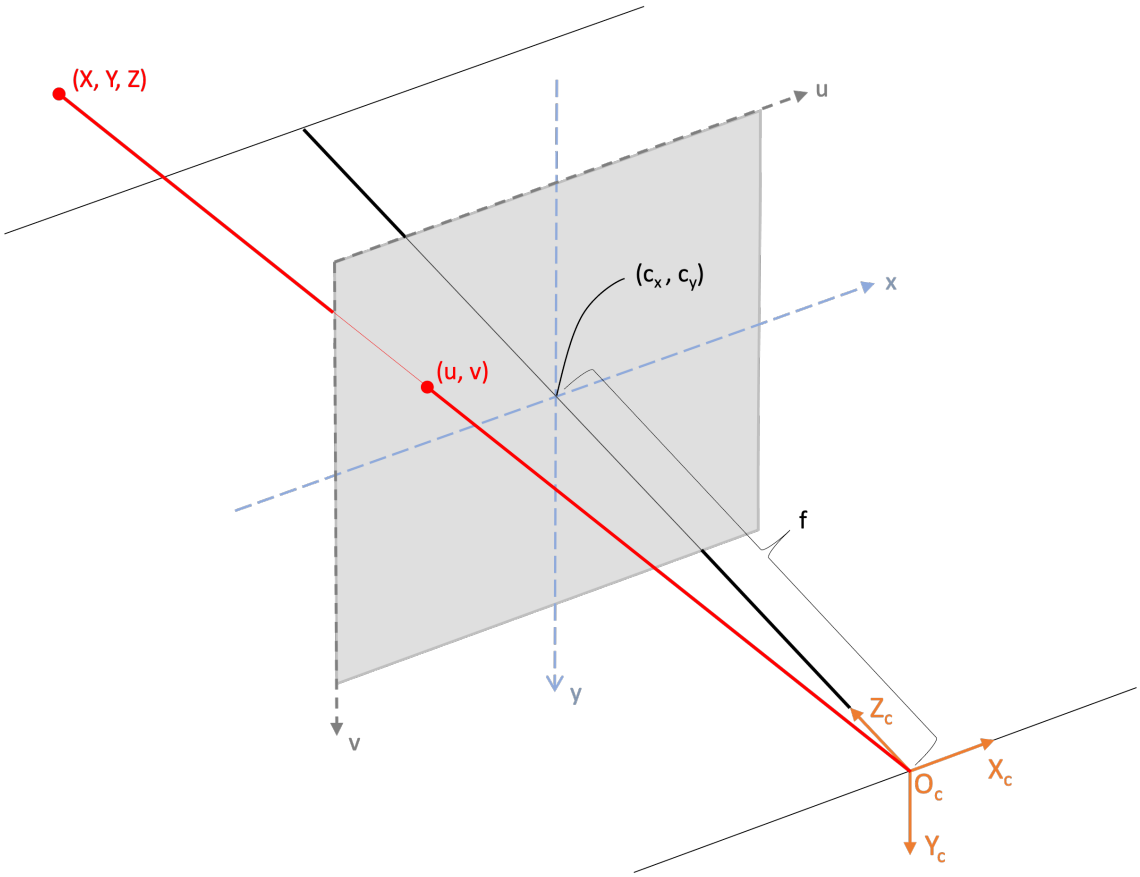



**Figure 2.2:** Camera model. Figure inspired from [4]

The introduction of *homogeneous coordinates* simplifies the mathematical description of the perspective camera model. Let a point $\mathbf{X}$ in Euclidian 2-space be defined by the real numbers, (X, Y). Let then the homogeneous coordinate representation, $\tilde{\mathbf{X}}$, of the point $\mathbf{X}$ be defined as ($k\tilde{X}$, $k\tilde{Y}$, k). The conversion between the point $\mathbf{X}$ and $\tilde{\mathbf{X}}$ can then be performed by dividing by

k to get (X, Y) [39]. The concept also applies to points in Eucliduan 3-space.

Let $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^\top$ be a 3-D world point, with corresponding image point $\tilde{\mathbf{u}} = \begin{bmatrix} \tilde{u} & \tilde{v} & \tilde{w} \end{bmatrix}^\top$ described in homogeneous coordinates. By similar triangles in Figure 2.2, the mapping can be mathematically described in matrix notation as follows:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{2.2}$$

where the image point $\mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^\top$ can be found by

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \tilde{u}/\tilde{w} \\ \tilde{v}/\tilde{w} \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix}. \tag{2.3}$$

The projection in (2.2) assumes that the principal point, being the point on the image plane located in a straight line from the center of projection, is the same as the origin of the image plane. This may not hold in practice, and hence two offset parameters $(c_x, c_y)$ compensate for this in the projection. The focal length's unit is millimeters, however, the image coordinates are given in pixels. Hence, pixel densities $(s_x, s_y)$ are needed. To account for sensor skewness the skew parameter, $s$, is also added. This will for normal cameras be set to 0. The final linear mapping that compensates for all mentioned effects is shown in (2.4). The matrix $K$ contains the so-called intrinsic of the camera [37].

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \underbrace{\begin{bmatrix} s_x f & s & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathrm{K}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.4}$$

Typically, the world coordinate frame will be used to represent points in space. A transformation between the camera coordinate frame and the world coordinate frame is therefore required [39]. Let $\mathbf{R}$ be the rotation from the world frame $n$ to the camera frame $c$, and $\mathbf{t}$ be the translation vector from the world origin to the camera origin. By defining the world 3D homogeneous coordinates as $\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{X} & \tilde{Y} & \tilde{Z} & \tilde{W} \end{bmatrix}^\top$, the general perspective camera model can be described through a matrix of dimensions 3x4, $\mathbf{P}$:

$$\tilde{\mathbf{u}} = \underbrace{\mathbf{K}\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}}_{\mathbf{P}} \tilde{\mathbf{X}}. \tag{2.5}$$

## 2.3 Kalman filtering

This section is largely based on Section 2.6 from the specialization project [4]. A series of potentially noisy measurements are used in the Kalman Filter (KF) to estimate the state of a stochastic dynamic system. By definition, every part of knowledge that describes a system is contained in the state of the system. The process model and the measurement model are the two models used to formulate the classic filtering problem. According to the process model, the process evolves over time. The relationship between the measurement and the state is described by the second model, the measurement model. Under the supposition that the process model, the measurement model, and the initial density are all Gaussian and linear, the KF has

a closed-form formulation [40]. With $\mathbf{x}_k$ being the state in timestep $k$, $\mathbf{z}_k$ being the measurement in timestep $k$, and $\hat{\mathbf{x}}_0$ and $\mathbf{P}_0$ being the initial state and covariance, the models and initial densities are in that case as follows:

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k; \mathbf{F}\mathbf{x}_{k-1}, \mathbf{Q}) \\ p(\mathbf{z}_k|\mathbf{x}_k) &= \mathcal{N}(\mathbf{z}_k; \mathbf{H}\mathbf{x}_k, \mathbf{R}) \\ p(\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_0; \hat{\mathbf{x}}_0, \mathbf{P}_0). \end{aligned} \tag{2.6}$$

The transition from state $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$ is described by the matrix $\mathbf{F}$, which is known as the transition matrix. The measurement matrix, $\mathbf{H}$, describes how the measurement relates to the state. Additionally, the statistical characteristics of the process noise and measurement noise are described by the two positive definite symmetric matrices $\mathbf{Q}$ and $\mathbf{R}$, respectively. The KF is the optimal solution to the estimation problem under the condition that all noise vectors are assumed to be independent of one another [40].

The formulation above covers only linear system dynamics. In the Extended Kalman Filter (EKF), the process dynamics and/or measurement dynamics are linearized around the state estimates, expanding on the traditional KF formulation. When non-linearities are introduced and linearizations are carried out, the optimality proof is lost. However, the EKF has been demonstrated to produce good performance [40]. Let the state transition and measurement dynamics for a non-linear system be given as:

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k; \mathbf{f}(\mathbf{x}_{k-1}), \mathbf{Q}) \\ p(\mathbf{z}_k|\mathbf{x}_k) &= \mathcal{N}(\mathbf{z}_k; \mathbf{h}(\mathbf{x}_k), \mathbf{R}). \end{aligned} \tag{2.7}$$

Comparing (2.7) with (2.6), $\mathbf{f}$ and $\mathbf{h}$ are know non-linear. By linearizing around the posterior estimate $\hat{\mathbf{x}}_{k-1}$ and the current prior estimate $\hat{\mathbf{x}}_{k|k-1}$ using a Taylor series expansion, the following matrices gives the linearized system dynamics:

$$\mathbf{F}(\hat{\mathbf{x}}_{k-1}) = \mathbf{F}_k = \frac{\partial}{\partial \mathbf{x}_{k-1}} \mathbf{f}(\mathbf{x}_{k-1}) \bigg|_{\mathbf{x}_{k-1}=\hat{\mathbf{x}}_{k-1}} \tag{2.8}$$

$$\mathbf{H}(\hat{\mathbf{x}}_{k|k-1}) = \mathbf{H}_k = \frac{\partial}{\partial \mathbf{x}_k} \mathbf{h}(\mathbf{x}_k) \bigg|_{\mathbf{x}_k=\hat{\mathbf{x}}_{k|k-1}}. \tag{2.9}$$

The prediction step and the update step make up the two steps of the KF filtering algorithm. The following Kalman Filter formulation is the same as the one in [40]. The state estimate $\hat{\mathbf{x}}_{k|k-1}$ and the state covariance $\mathbf{P}_{k|k-1}$ are updated in the prediction step in accordance with:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1}) \tag{2.10}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^\top + \mathbf{Q}. \tag{2.11}$$

The update step cannot be carried out in an ideal manner without the predicted measurement $\hat{\mathbf{z}}_{k|k-1}$. The innovation, $\boldsymbol{\nu}_k$, is defined as the difference between the received measurement $\mathbf{z}_k$ and the predicted measurement $\hat{\mathbf{z}}_{k|k-1}$:

$$\boldsymbol{\nu}_k \equiv \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) = \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1}. \tag{2.12}$$

The weight that should be given to the measurement must be carefully selected in order to update the predicted state estimate in the best possible way. This weight, also known as the Kalman gain $\mathbf{W}_k$, is determined by:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R} \tag{2.13}$$

$$\mathbf{W}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}. \tag{2.14}$$

The state estimate $\hat{\mathbf{x}}_k$ and state covariance $\mathbf{P}_k$ for timestep $k$ can now be determined in accordance with the following:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{W}_k \nu_k \tag{2.15}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{W}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \tag{2.16}$$

## 2.4 Deep learning

Machine learning, a subfield of Artificial Intelligence, includes deep learning [41]. According to Mitchell, "The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience" [42, p. xv]. Traditionally, computers were taught numerous complex and elaborate rules on how to handle a problem. In contrast, machine learning involves the computer acquiring a model that enables it to evaluate instances and a limited set of instructions on how to update the model when it makes a mistake. In this way, the model is anticipated to solve problems with a high degree of accuracy if it is given enough examples to learn from and enough time to do this learning [41]. Let the model be defined as a function, h($\mathbf{x}$, $\theta$). The model takes in two parameters, the input vector $\mathbf{x}$ and the model parameters $\theta$. By changing the values of these model parameters until they are at their ideal value, the model learns from the examples. The models must be quite complex to be able to represent the relationship between the data and the anticipated outcome as the problem grows increasingly complicated. As a result, the idea of deep learning involves extremely complicated models that are motivated by a brain-inspired structure. When used to tackle problems in the disciplines of computer vision and natural language processing, the models have shown excellent results. Deep learning algorithms frequently match, and in some cases even surpass, human precision [41].

### 2.4.1 Deep Neural Networks

The data structure called Artificial Neural Network (ANN) is largely inspired by how the brain functions [43]. The way the human brain uses billions of neurons, tiny computational units connected together, to perceive the world is the background for the ANNs. Each neuron is receiving inputs ($\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$) which are multiplied by weights ($\mathbf{w}_1$, $\mathbf{w}_2$, ..., $\mathbf{w}_n$). The weighted inputs are then summed together and added with a bias b. The neuron output is then constructed by passing the summed input with the added bias through an activation function, f. The artificial neuron with its input, weights, bias, activation function f, and output y is illustrated in Figure 2.3.

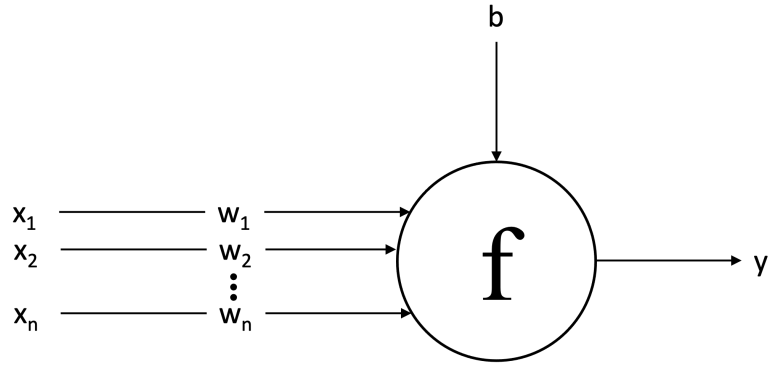



**Figure 2.3:** Artificial neuron. Inspired from [41]

Mathematically the artificial neuron can be described by introducing the input and weights as vectors $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^\top$. The neuron output can then be expressed as:

$$y = f(\mathbf{x} \cdot \mathbf{w} + b). \tag{2.17}$$

This output, y, can then be used as input by another neuron. Hence, the neurons are arranged in layers, each of which is possibly connected to neurons in the layer next to it. This construction is called a feed-forward ANN and is illustrated in Figure 2.4. The input layer is the layer on the left, while the output layer is the layer on the right. The hidden layer is the middle layer in the figure. It's possible for an ANN to have a significant number of hidden layers. The figure does not include any biases, yet each and every neuron has a bias in the same way as in Figure 2.3.

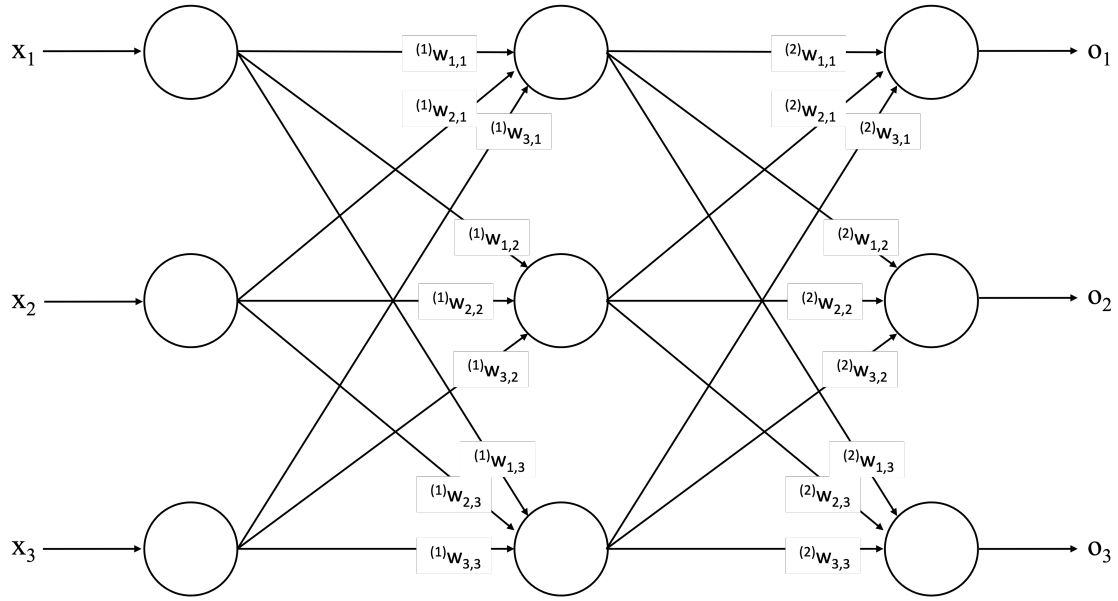



**Figure 2.4:** Artificial neural network

In the ANN, the parameter vector $\boldsymbol{\theta}$ is the same as the vector of individual weights ${}^{(k)}w_{i,j}$ connecting the $i^{th}$ neuron in the $k^{th}$ layer with the $j^{th}$ neuron in layer $k+1^{st}$ layer. Finding

the best values for the weights in $\boldsymbol{\theta}$ is essential for the Artificial Neural Network's capacity to solve problems.

The network must go through a process known as training in order to establish each weight in the parameter vector. The ANN is trained by presenting it with a large number of training examples that correspond to the desired output. If the activation function, *f*, is linear the weights could be calculated by solving a linear system of equations. However, there can be no solvable system of linear equations if the function f is non-linear. The efficiency of the network's learning is assessed using a cost function, commonly referred to as a loss function or an objective function. For instance, the mean square error between the input and output of the network may be represented by this function. In general, it is tough and computationally challenging to minimize a function with several variables. The network weights are therefore updated iteratively, for instance by using gradient descent to minimize the loss function [41]. These ANNs could potentially have millions or billions of weights, many hidden layers, and a deep level of complexity. The training process can become very computationally expensive due to the possibly enormous amount of mathematical calculations. Hence, in order to speed up both training and inference a Graphics Processing Unit (GPU) may be used [44]. The GPU has the advantage of greater parallelism capability and dedicated memory.

### 2.4.2 Convolution Neural Networks

Convolutional Neural Network (CNN) are very similar to conventional ANN in that they are made up of neurons that optimize their weights during training. However, CNNs are primarily employed in the field of image pattern recognition [45]. One of the biggest drawbacks of ANNs is that they frequently have trouble processing image data because of the enormous number of training weights required by the sheer number of pixels in an image. The connection between each and every pixel in the image would be required. A 64 x 64-pixel image with three color channels would for every single neuron in the input layer require 12288 weights. By dividing the neurons into the three dimensions of height, width, and color channel, CNNs try to take advantage of the spatial relationships between the pixels in the image [45]. The spatial relationship between pixels is captured by only connecting the neurons within a given layer to a small area of the layer preceding it. Convolutional layers, pooling layers, and fully connected layers are the three types of layers that make up a typical CNN structure.

Data that are represented as matrices are subjected to convolutional mappings in the convolutional layers. The data go through a process known as convolutionally mapping [45]. By taking the dot product of a moving grid in the data matrix and another matrix of weights known as a kernel, the values are transferred to the following layer. Figure 2.5 shows an illustration of a convolution operation with data matrix A and kernel K.

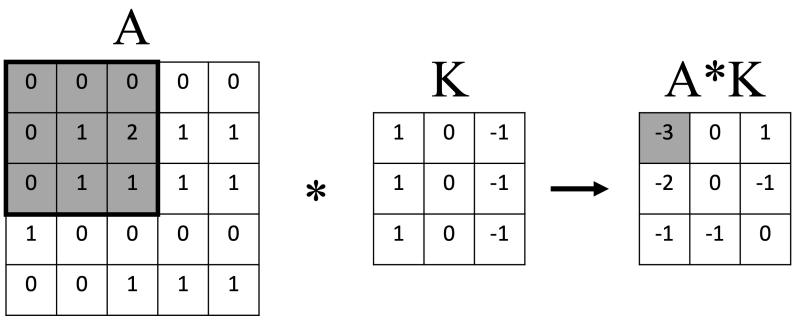



**Figure 2.5:** Convolution operation with a 3x3 convolution filter K

Pooling layers aim to minimize the data representation's dimension. A 2x2 max-pool layer will propagate the highest value in a 2x2 grid onto the next layer, halving the matrix size. Max-pooling is the most used pooling technique, however, pooling methods such as average pooling and min-pooling could also be employed [45]. Figure 2.6 illustrates the max-pooling operation.

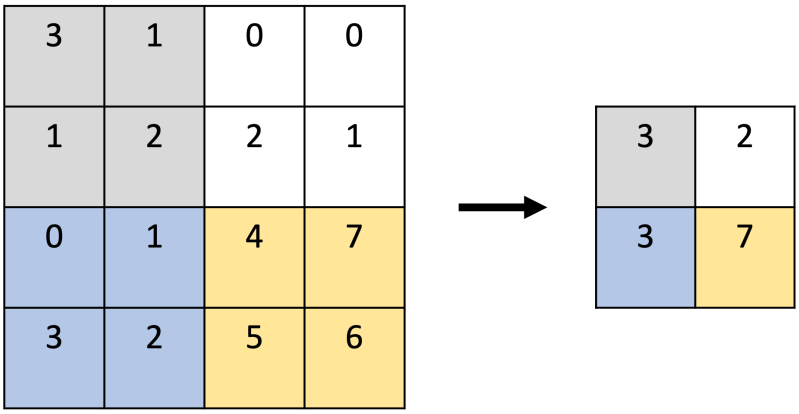



**Figure 2.6:** Max-pooling operation on a 4x4 data matrix

A CNN structure's final layers frequently include one or more fully-connected layers. Neurons in the fully connected layer have direct connections to the neurons in the two adjacent layers. This is comparable to how neurons are connected in conventional ANN models.

### 2.4.3 YOLO - You Only Look Once

In [13], Redmon et al. first developed the real-time bounding box detection algorithm known as YOLO (You Only Look Once). The YOLO detector has become very well known in the field of computer vision due largely to its extremely fast inference speed and high mean Average Precision (mAP). Based on a single neural network, the YOLO algorithm generates a set of bounding boxes and corresponding class probabilities from the entire input image. YOLO performs object detection in a single stage, which makes it much faster and more effective than other object detection systems that use a two-stage approach of object proposal and object classification. A CNN, also called the backbone network, is the first component of the YOLO algorithm, which splits the input image into a feature map. The detection layer, which is the second component, outputs bounding boxes and class probabilities after receiving the feature map as input.

#### YOLOv8

The most recent version of the YOLO object recognition and image segmentation is called YOLOv8. It is a state-of-the-art (SOTA) model that adds new features and enhancements to increase detection performance. It is substantially faster and more accurate than earlier YOLO iterations [46].

The original YOLO structure created by Remon et al. [13] was maintained in a custom-made deep learning environment written in C code called Darknet. After shadowing the original YOLO repository in PyTorch, the developers of the new YOLOv8 released version 5 of YOLO. An updated version of the YOLOv5 network called YOLOv8 was released on January 10th, 2023. It features a new backbone network with new convolutions. Moreover, the YOLOv8 model is anchor-free. That is, rather than predicting an object's offset from a known anchor box in the image, it predicts the center of the object directly. Anchor boxes were a difficult component of early YOLO models because they may represent the distribution of the boxes in the target benchmark but not the distribution of the custom dataset. As a result of fewer box predictions due to anchor-free detection, the Non Maximum Suppression (NMS) process may proceed more quickly. NMS is the process of sorting out bounding boxes with weaker confidence scores in favor of bounding boxes with higher confidence. YOLOv8 includes online image augmentations during training. Hence, the model sees a slightly different variety of the images it has been given at each epoch. One example of these online image augmentations is called mosaic augmentations. By stitching four images together, the model is a more robust learner and forces the model to learn objects in new locations, with partial occlusions, and against different surrounding pixels. The YOLOv8 framework includes a Python API [46].

### 2.4.4 Segmentation

Image segmentation in computer vision involves dividing an image into multiple segments, where segments present different objects or areas of interest [47]. Semantic segmentation and instance segmentation are the two main types of image segmentation. Semantic segmentation categorizes each pixel in accordance with a class label rather than distinguishing between distinct instances of segmented objects. Contrarily, instance segmentation also defines boundaries between objects belonging to the same class and gives each distinct object a unique identifier [47]. Deep learning models can be used for the purpose of image segmentation. One of the possible main structures usable is the Encoder-Decoder structure. The encoder is a CNN structure that takes in an image and produces a feature map. The feature map from the encoder is then fed into the decoder that reconstructs a segmentation mask. The output and input of an encoder-decoder structure, therefore, have the same dimensions, possibly with a different number of color channels. A general encoder-decoder segmentation structure is illustrated in Figure 2.7. The encoder-decoder structure is trained similarly as for a general ANN-structure. By presenting the network with images and expected segmentation masks, the network updates its weights using an iterative scheme in order to minimize a cost function.

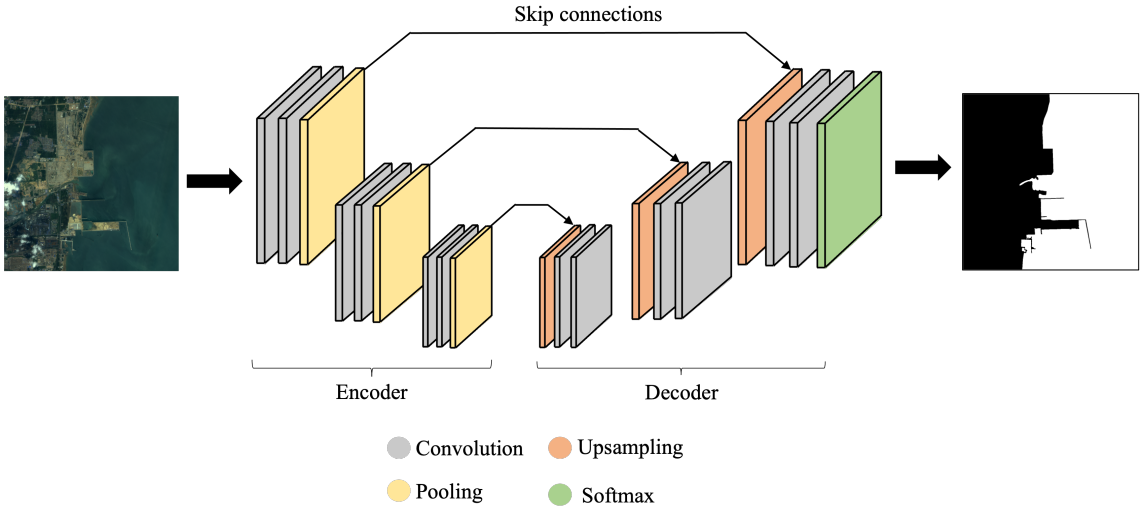



**Figure 2.7:** Encoder-Decoder segmentation model. Images in illustration from sea-land dataset[48]

As the encoder contains multiple hidden layers, the feature map is effectively shrunk and hence the encoder may discard valuable information. To preserve some of the possible valuable information from the encoder to the decoder, skip connections are introduced in the structure. In the UNet encoder-decoder structure presented in [49], the skip connections contain the feature maps from the corresponding layers in the encoder. In this way, the decoder has access to both the more general features from the deepest encoder layer as well as the more detailed features from the skip connections. The skip connections in SegNet presented in [50] contain the max pooling index from the corresponding max-pool layer in the encoder. In this way, the upsampling can be performed using this information as illustrated in Figure 2.8. As the information provided in the skip connections in SegNet is much sparser than in UNet, the SegNet model is smaller and requires less memory [50].

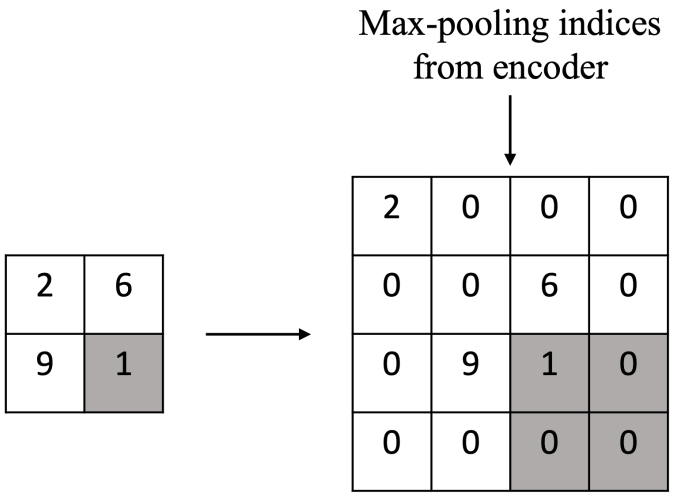



**Figure 2.8:** Upsampling using max-pooling indices

## 2.5 Multi Object Trackers

Multi Object Tracking (MOT) is the process of identifying and tracing the movements of multiple objects of interest within a given area while maintaining their identities. Over the years, MOT has become an increasingly significant concept in the field of computer vision, with practical applications ranging from monitoring pedestrian activity on streets and roadways to tracking vessels in a harbor or even different parts of a single object, such as the body parts of a human [51].

In contrast to Single Object Tracking (SOT), which focuses primarily on developing intricate appearance and motion models to address challenges like scale, rotations, and illumination changes, Multi Object Tracking (MOT) presents two additional obstacles: determining the possibly varying number of objects present over time and preserving their identities. While both SOT and MOT face their challenges, there are unique difficulties associated with MOT, such as occlusions, the initiation, confirmation, and termination of tracks, indistinguishable appearances of objects, and interactions between multiple objects [51].

### 2.5.1 SORT

To track objects in a video, SORT (Simple Online and Realtime Tracking) employs techniques like the KF and the Hungarian method. Four fundamental parts make up the SORT algorithm. The first module is a bounding box detector such as a YOLO network. The bounding boxes are tracked by the SORT algorithm.

To predict the bounding boxes from one frame to the next, the second module uses an estimation technique that involves a linear CV KF which is independent of other objects and camera motion. The state of each of the tracked bounding boxes is modeled as:

$$\mathbf{x} = \begin{bmatrix} u & v & s & r & \dot{u} & \dot{v} & \dot{s} \end{bmatrix}, \tag{2.18}$$

where u and v denote the target's center's horizontal and vertical pixel locations, respectively. The s and r denote the target's bounding box's scale (width times height) and aspect ratio. The dotted variables denoted the time derivative. If a bounding box is associated with the target, the state is updated with the detection. If no measurement is associated with the target, the prediction is simply used as the bounding box state [52].

The data association module is the third one in the algorithm. By predicting where each target's bounding box geometry will be in the current frame, the association between tracks and new detection is achieved. The Intersection-Over-Union (IOU) between each bounding box detection and all projected bounding boxes from the existing targets is then used to calculate the assignment cost matrix. The IOU is illustrated in Figure 2.9. The Hungarian algorithm is used to solve the assignment optimally by creating a matrix of costs, identifying the assignment with the lowest cost, and eliminating assignments with the same lowest cost until all measurements have been assigned to a track. Moreover, if the detection to target overlap is less than $\text{IOU}_{min}$, assignments will be rejected.

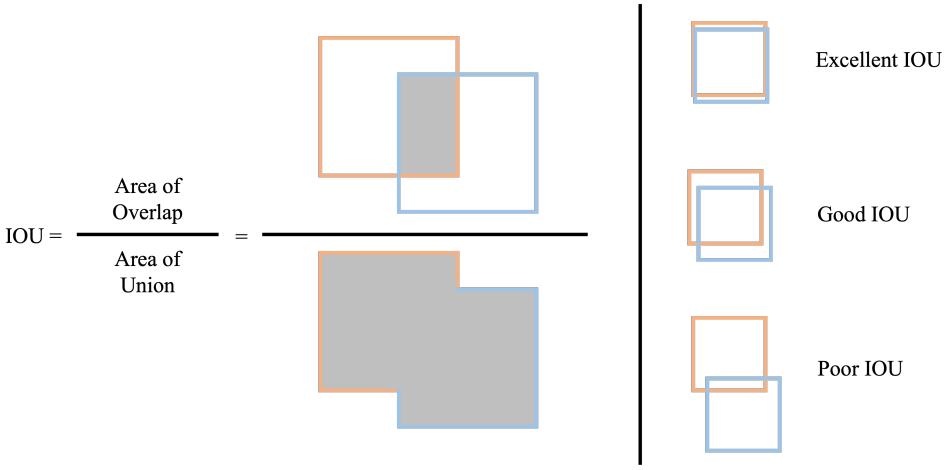



**Figure 2.9:** Intersection-Over-Union (IOU) illustration

The construction and deletion of tracks are handled by the fourth and final module. If a track is not associated with any measurement for a predetermined number of frames, it is terminated. If an object reappears, tracking of the object will start over with a new track id [52].

### 2.5.2 DeepSORT

The DeepSORT method is an addition to the standard SORT algorithm. It stands for Simple Online and Realtime Tracking with a Deep Association Metric. The authors of the DeepSORT method in [23] incorporate an appearance feature vector, a new distance measure based on the object's appearance, in the algorithm's data association module. A well-discriminating feature embedding is trained offline before tracking is started. An example of a feature embedding is a trained CNN network with the final classification layer removed and replaced with a dense layer that generates a single feature vector that is awaiting classification. When a new detection is subjected to the association module, the module runs the content of the bounding box through the feature embedder to get an appearance feature vector for the detection. The feature vector is then used together with the IOU metric in the data associations to map detections with tracks. It has been shown that the DeepSORT framework can reduce ID swaps between tracks and strengthen the algorithm's resistance to occlusion. The additional step of producing feature embeddings has, however, increased the DeepSORT algorithm's runtime [23].

# Chapter 3

# Experimental setup

## 3.1 Parrot Anafi FPV drone

The same drone used in this thesis was also used in the author's specialization project [4]. The drone is an Anafi FPV drone from the French drone manufacturer Parrot. The drone has four foldable rotor arms and weighs 320 grams. The Anafi also has a gimbal-mounted 4K HDR camera. The gimbal has a 180-degree tilt capability making it possible to point the camera both directly upwards and downwards. Figure 3.1 shows a picture of the Anafi FPV drone.


**Figure 3.1:** Parrot Anafi FPV

In table 3.1, some of the key characteristics from [53] of the Anafi FPV drone are listed.

### 3.1.1 Sensors

All the characteristics and data from this section are from the Anafi White Paper [53].

| Parrot Anafi FPV | |
|---|---|
| Weight | 320 g |
| Dimensions (unfolded) | 175x240x65 mm |
| Maximum wind speed resistance | 50 km/h |
| Maximum speed | 15 m/s |
| Flight time | 25 min |
| Max flight distance | 4000 m |

**Table 3.1:** Key characteristics of the Anafi FPV drone

#### Camera

A camera module with a Sony IMX230 1/2.4" sensor is mounted on the front-facing gimbal of the drone. The camera module can record video at 4K Cinema resolution (4096x2160 pixels at 24 frames per second). In addition to local storage, the drone can stream images with 1280 x 720 resolution at 30 frames per second to a phone or computer with an end-to-end delay of 280 ms. The camera has a horizontal FOV of 69 degrees.

#### Other sensors

The drone's Inertial Measurement Unit (IMU) is an Invensense MPU-6000. This system comprises a 3-axis gyroscope for tracking angular velocity. In addition, the IMU has a 3-axis accelerometer. The drone comes equipped with an AKM AK8963 magnetometer, providing a measurement of magnetic fields for the determination of the drone heading. Additionally, it features an ST Microelectronics LPS22HB barometer for altitude measurements. The drone utilizes the U-BLOX UBX-M8030 GNSS receiver. The receiver has a Time-To-First-Fix of 35 seconds, a positional standard deviation of 1.2 meters, and a velocity accuracy of 0.5 meters per second. Moreover, the receiver is designed to work with four constellations, namely GPS L1, Galileo E1, Glonass L1, and BeiDou B1C. The drone is equipped with both an ultrasonic sensor and a vertical camera, enabling it to measure altitude and velocity more accurately. The vertical camera is an MX388 sensor with a resolution of 640x480 pixels. The camera utilizes optical flow to measure horizontal velocity and altitude.

### 3.1.2 Communication

The default user interface for Anafi FPV drones is the *FreeFlight 6*[1] application. The application offers users an interface for viewing the drone's status and viewing the camera feed. The application can also be used to set the max allowed velocity, altitude, and range of the drone. Additionally, the drone could also be connected wirelessly to the *SkyController 3* by Parrot. By doing this, the user can operate the drone manually using a handheld controller.

The Parrot Olympe API[2] can be used to control the drone. The drone states and camera feed can be accessed on a Linux-based computer by writing custom Python code and then running the scripts. By setting waypoints or attitude references, for example, the API also enables computer programs to control the drone. The Anafi drone creates a WiFi access point,

[1] `https://www.parrot.com/en/apps-and-services`
[2] https://developer.parrot.com/docs/olympe/index.html

which has a limited range as discussed by Falang in [3]. Hence, for longer control distances, the Skycontroller 3 can be used as a relay by connecting the SkyController 3 to the computer running the Python scripts through USB. Parrot states in one of their forum postings[3] that the API is not designed for closed-loop command control. The author experienced large delays using the API in the project thesis [4], however a software update from Parrot released before this thesis mitigated these latency issues.

### 3.1.3 Control

The drone internal estimation algorithm is an Extended Kalman Filter (EKF) with 18 states such as velocity, attitude, position, and wind. An internal control loop handles all instructions sent to the motors and the camera gimbal. The control loop runs at a frequency of 200 Hz.

The drone has two types of controls for its altitude and position. For altitude control, it uses a PID controller that separates the drone's desired altitude from other factors that could affect its altitude. For position control, the drone uses a similar approach to wind correction.

The internal attitude control uses a PID controller. Similar to altitude control, attitude control also uses a model to separate dynamics from disturbances. Additionally, the system compensates for aerodynamic torque, which is the force that can push the drone off balance. It also estimates external torques, such as those caused by wind or other external factors, to further improve its stability.

## 3.2 Other software and hardware

The experiments in this thesis and all ANN training were conducted on a Komplett Khameleon P9 Pro provided by the Department of Engineering Cybernetics (ITK) at NTNU. Table 3.2 provides the specifications of the computer. The computer was installed with NVIDIA's CUDA and CudNN software for GPU acceleration during ANN training and inference. Moreover, the software versions specified in Table 3.3 were installed on the computer.

| **Computer specifications** | |
|---|---|
| Manufacturer | Komplett |
| Computer type | Laptop |
| Model name | Khameleon P9 Pro |
| CPU | Intel Core i7-9750H |
| GPU | Nvidia GeForce RTX 2070, 8GB |
| RAM | 32 GB DDR4 |

**Table 3.2:** Computer specifications for laptop used in thesis

### 3.2.1 Robot Operating System

Robot Operating System (ROS) is an open-source system designed to provide a common platform for developers who build and program robot systems. It is widely used in the field of

[3]https://forum.developer.parrot.com/t/positionchanged-trigger-rate/10051/2

| Software spesifications | |
|---|---|
| Operating system | Ubuntu 20.04.5 Focal Fossa |
| Python | 3.8.10 |
| ROS | Noetic Ninjemys |
| Parrot Olympe API | 7.5 |
| Nvidia driver | 515.65.01 |
| CUDA | 11.7 |
| CudNN | 8500 |
| YOLOv8 | 8.0.36 |
| OpenCV | 4.2.0 |

**Table 3.3:** Software versions used in thesis

robotics programming [54]. ROS is designed for implementing modular software systems, where a possibly large amount of independent scripts run simultaneously and interact with each other. In ROS, the software components that interact with each other are called *nodes*, which communicate by sending messages to each other. A *topic* in ROS is a named stream of messages with a specific type. Nodes can either subscribe to a topic to read messages from it or publish messages to it. The message types can be either the pre-installed standard messages that come with ROS or they can be user-defined custom message types. In order for the nodes to find each other, and be able to communicate with each other by message passing they need to connect to a *roscore* [54]. The roscore is only used by nodes to find their peers. A communication graph can be used to visualize the data flow between various ROS nodes. In this graph, topics are represented by square boxes, to which nodes represented by oval-shaped circles subscribe or publish. Figure 3.2 illustrates such a communication graph.

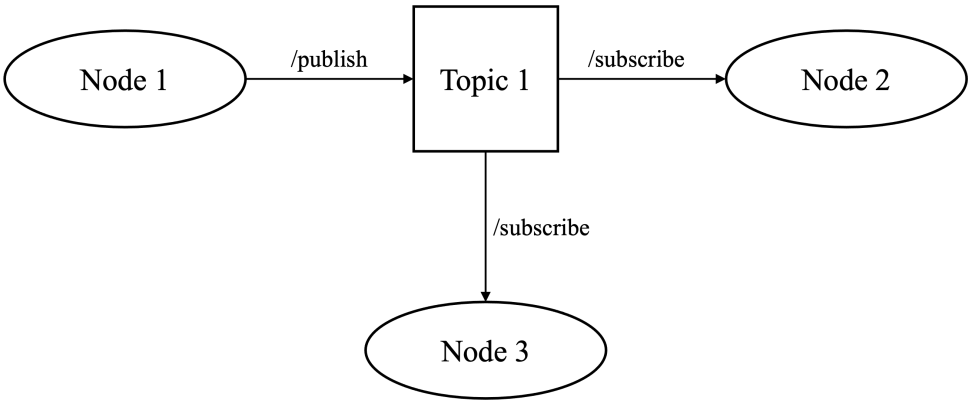



**Figure 3.2:** Example ROS communication graph

In addition to publishing and subscribing to topics, ROS nodes can communicate with each other through *services* and *actions*. These methods enable the execution of a function in one node while calling it from another. Actions are better suited for time-extended, goal-oriented behaviors, while service calls are ideal for tasks that are only occasionally performed and take a limited amount of time to complete [54].

ROS is designed to be multilingual, allowing nodes to be written in the most appropriate language as long as the topics have consistent standardized ROS messages. Another advantage of ROS is its modular structure, which makes it easy to change or swap out single modules. For instance, sensor modules can easily be replaced by data simulation modules that simulate sensor data, allowing the program to be tested using reference data in addition to real sensor data. Additionally, the entire runtime of a ROS program can be recorded to a *bagfile*, which can be used to replay all messages published to topics. Finally, multiple ROS nodes can be run from a single terminal using *launchfiles* in .xml format, with each node configured to accept parameters from a parameter server. This enables the parallel operation of two nodes of the same type but with different parameters.

RViz is a 3D visualizer package for ROS. RViz is used in this thesis to visualize images from the drone, intermediate results, and coordinate transformations for debugging purposes.

## 3.3 Datasets

For the purpose of training ANN for detection and segmentation, multiple datasets were used in this thesis. They will be summarized in the subsections that follows.

### 3.3.1 SeaDroneSea

Constructed to aid in the development of systems for SAR missions utilizing unmanned aerial vehicles in maritime scenarios, SeaDroneSea is a large dataset [55]. The datasets include three label categories: single-object tracking, multi-object tracking, and object detection all taken with a drone-mounted camera. The multi-object track, which contains images taken from 22 videos totaling 54105 images, is the track that was used in this thesis. The reason is to train the ANN using images that are the most representative of actual SAR mission images featuring multiple objects from continuous data as well as the multi-object track being the one with the largest amount of images. The images in the dataset have a resolution of 3840x2160. Furthermore, the dataset contains meta-data from the data generation, featuring GNSS position, velocity, gimbal pitch and heading, and compass heading. The train-validation-test split of the dataset is 50.4%, 15.9%, and 33.7% respectively.

### 3.3.2 AFO

The AFO - Aerial dataset of floating objects dataset was created for ANN model training in maritime SAR applications [56]. The dataset includes images from fifty videos of objects floating on the water shot by different drones with cameras mounted on them in resolutions ranging from 1280x720 to 3840x2160. The datasets include 39991 annotated objects across 3647 total images. The training, validation, and test splits of the dataset are 67.4%, 13.48%, and 19.12%, respectively.

### 3.3.3 MOBdrone

A significant amount of aerial images is available in the Man OverBoard (MOBDrone) dataset [57]. The dataset was created for the purpose of training ANN models to recognize humans in the water in man-overboard situations. The dataset consists of 126170 images from 66 videos. It includes more than 180 000 hand-written annotations. The train-validation split for the

dataset is 70.2-29.2%. Due to its enormous size, this dataset is not used for ANN training in this thesis; instead, videos are used to validate and assess the object detectors and multi-object trackers.

### 3.3.4 Sentinel-2 Water Edges Dataset

The Sentinel-2 Water Edges Dataset (SWED) is a dataset made available by the UK Hydrographic Office at [58]. The dataset contains images and corresponding segmentation labels from the Copernicus SENTINEL-2 project, which comprises two polar-orbiting satellites [59]. The segmentation labels are binary with 1 for water, and 0 for non-water. The images have 12 bands, where the bands 2-4 correspond to the BGR bands respectively. The geospatial resolution of the RGB bands is 10 meters per pixel.

### 3.3.5 Sea Land Segmentation Dataset

The Sea-Land segmentation dataset contains 1726 hand-annotated images of various coastlines in Lianyungang, China [48]. The segmentation labels are binary, with water and non-water values. The images are from the Gaofen-1 satellite, with 4 bands and a geospatial resolution of 8 meters per pixel. The 1-3 bands contain the BGR color bands respectively.

## 3.4 DroneLab at NTNU

Some of the proposed algorithms in this thesis were tested and evaluated in the NTNU drone lab. The drone lab is located in Elektrobygget at NTNU Gløshaugen in Trondheim on the second basement floor. The drone lab has a safe flying zone that is surrounded by a net and equipped with foam mattresses on the floor. The drone lab setup can be seen in Figure 3.3.

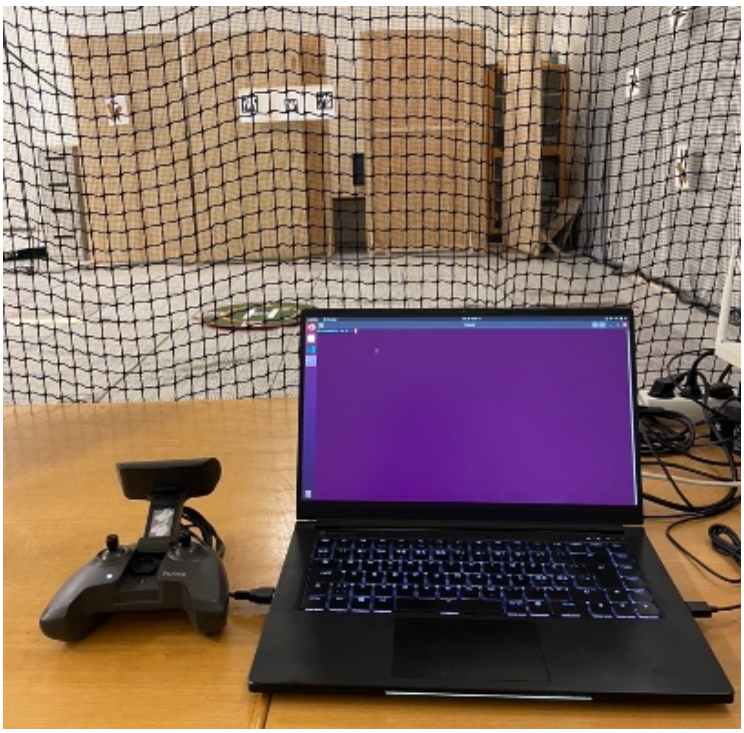


**Figure 3.3:** Setup at the drone lab at NTNU. Image from project thesis [4].

### 3.4.1 Qualisys motion capture system

The drone lab at NTNU has a Qualisys motion capture system installed [60]. To obtain precise motions and the precise locations of 3D objects within the scene, this system uses multiple cameras to detect tiny round reflective markers in the scene. The system can operate in real-time. With location estimates down to 1 mm and orientation accuracy of 0.1 degrees, Qualisys claims to be able to estimate motion and locations with extreme precision [61]. The system enables the definition of objects that have multiple mounted markers on them, providing the object's pose (Position and Orientation) in relation to a predefined coordinate system. Furthermore, the system is wrapped in a ROS-wrapper by Paolo De Petris from The Department of Engineering Cybernetics (ITK) at NTNU. The wrapper provides the orientation, location, and velocity of defined objects in the scene on separate topics.

The Qualisys motion capture system is connected to another computer than the project laptop. Consequently, ROS requires a multimachine configuration. Both computers need to be connected to the same network for this to function. To reduce communication channel latency, a router set up specifically for the two devices was used. Additionally, the Qualisys Motion Capture system ROS wrapper is set up to use a *roscore* that is launched in one terminal by the project laptop. The project laptop will therefore have access to the topics containing the pose and velocity data of the objects in the scene.

It is crucial to asymmetrically place the reflective markers around the object's axis when attaching them to objects for the Qualisys motion capture system to track. This is done to avoid any ambiguity in the tracked object's pose. Figure 3.4 depicts the Anafi FPV drone with the markers attached to it.

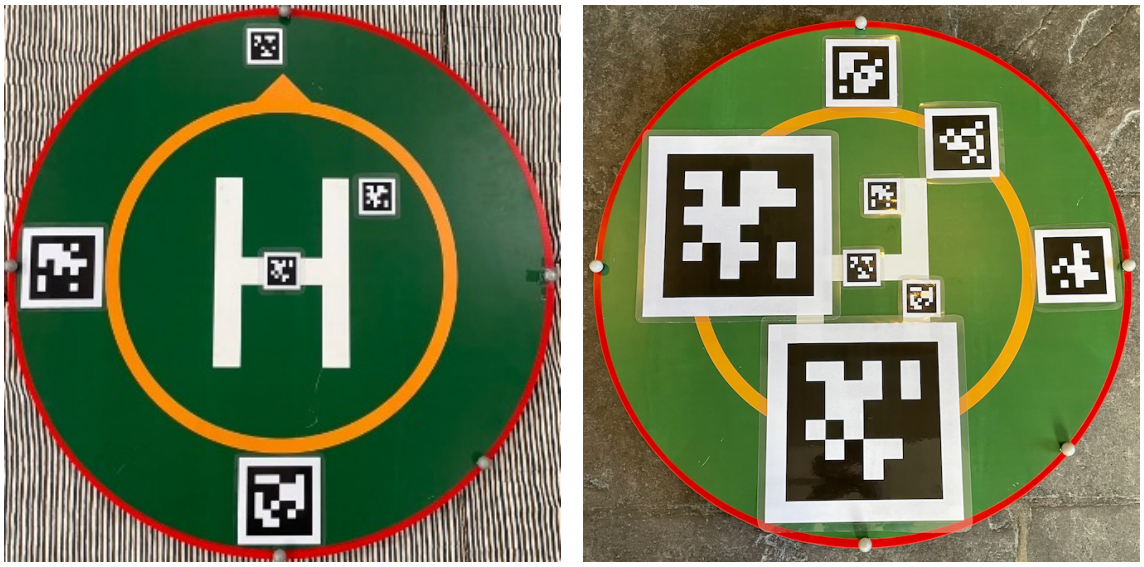

**(a)** Top down view **(b)** Sideways view

**Figure 3.4:** Anafi FPV drone with reflective markers attached

## 3.5 Objects and outdoor testing

### 3.5.1 Helipad

Sundvoll [1] created the original design and manufacture of the drone helipad used in this thesis. In the author's specialization project [4], the helipad design was changed by adding fiducial markers, AprilTags [15], to the helipad. The helipad design used in the project thesis can be seen in Figure 3.5a. In this thesis, new and larger AprilTags were added to the helipad to increase detection performance. The helipad with the new AprilTags can be seen in Figure 3.5b.



**(a)** Project thesis helipad design **(b)** New helipad design

**Figure 3.5:** Helipad design

The helipad is designed to be mounted on a 1:20 scale model of the DNV ReVolt[4] autonomous marine vessel. The helipad can be seen mounted on the ReVolt vessel in Figure 3.6.

[4] https://www.dnv.no/karriere/employee-stories/interview-articles/revolt.html


**Figure 3.6:** Helipad mounted on ReVolt. Image courtesy of Tom Arne Pedersen from DNV

### 3.5.2 Location and equipment for outdoor testing

Three distinct areas in Trondheim, identified on the map in Figure 3.7, are used for outdoor testing of the system. The three locations are Hurtigbåtterminalen, Nyhavna, and Korsvika.


**Figure 3.7:** Outdoor testing locations. Map from Kartverket [62]

Furthermore, when testing outside a portable power generator were used to power the computer. The generator can be seen in Figure 3.8a. In addition, the immersion suit in Figure 3.8b was used when the system's properties to detect people in the water were evaluated.


**(a)** Portable power generator **(b)** Immersion suit

**Figure 3.8:** Equipment used for outdoor testing

### 3.5.3 Drone license

The rules set by the Civil Aviation Authority regulate drone usage in Norway [63]. The rules state that all drones that are to be flown outside with a weight over 250 grams or have a mounted camera need to be covered by insurance and be registered. In addition, the operator of the drone has to be a registered drone operator with a passed online drone exam in the appropriate drone class for the drone flown. With completed A1 and A3 certifications and drone insurance through home contents insurance, the author of this thesis is a licensed private drone operator. The drone is also marked with the author's drone operator number in compliance with regulations.

# Chapter 4

# Methodology

The system architecture overview, design goals, and development strategy will all be covered in this chapter.

## 4.1 Software development

This section will focus on the software development methodology utilized to construct the code base for this thesis. The development of this project was built upon the foundation established by the previous specialization project [4]. In the subsequent sections, an explanation of the system architecture and design goals that corroborate the contributions of this thesis will be presented. Furthermore, the ROS Olympe bridge that was developed in [4] will be briefly covered.

### 4.1.1 System architecture

The high-level system architecture is presented in Figure 4.1. The nodes in the project are as follows:

- *ROS Olympe bridge* - An interface between ROS and Parrot Olympe API.
- *Search subsystem* - Consisting of two nodes.
  - *YOLOv8 Search node* - A YOLO node detecting humans and objects of interest in the water.
  - *Multi Object tracker* - Tracking of detected objects with confidence calculations.
- *Safe point generator subsystem* - Consisting of three nodes.
  - *Offline map segmentation* - Segmentation using an offline map source.
  - *Deep learning based segmentation* - Segmentation using real-time image data.
  - *Segmentation master* - Generating safe points based on segmentation masks from the two segmentation nodes.
- *Pixel-to-coordinate-transformer* - Georeferences detections and safe points by transforming points from pixels in image coordinates to NED world coordinates.
- *Helipad tracker subsystem* - Helipad tracker developed in the specialization project [4]. Modified to be compatible with the code written for this thesis. Consisting of three nodes.
  - *Constant Velocity Extended Kalman Filter* - Updated with new toggle logic.

- *AprilTag-based position estimator* - Updated with new AprilTag design.
- *DNN-CV-based position estimator* - Updated with new YOLOv8 network architecture.

- *Transform Publisher* - Publishing transforms between the different coordinate frames.
- *Perception master* - Responsible for gathering information from other nodes, processing it, and distributing it out from the perception system.

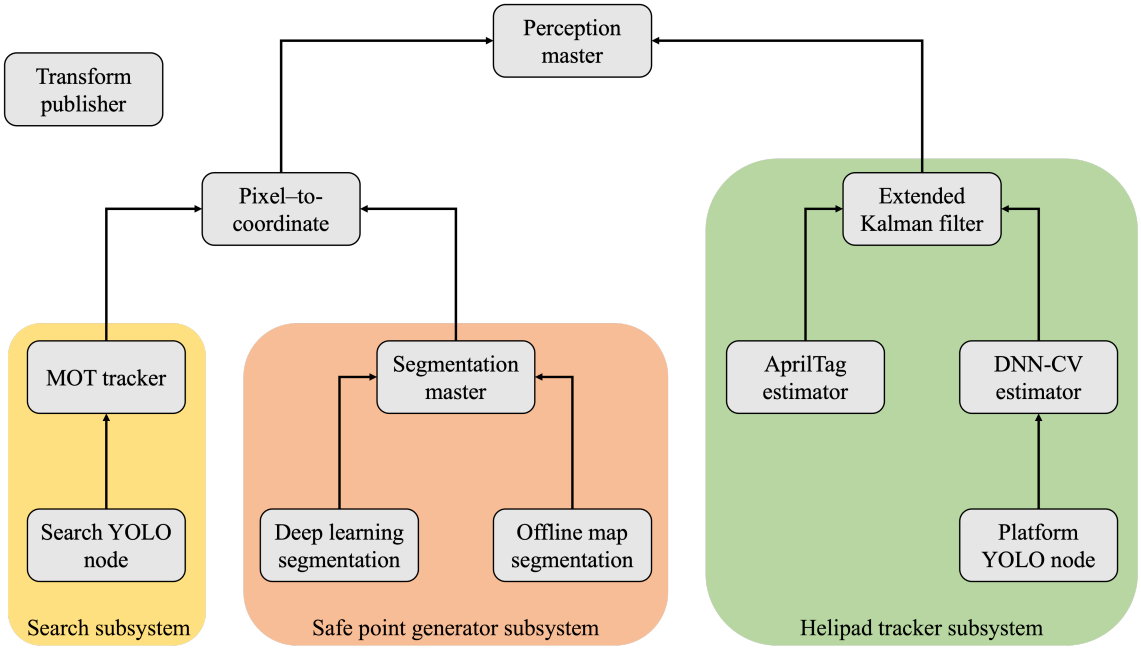



**Figure 4.1:** Software architecture describing high level interactions between nodes

The system is designed to be compatible with a mission planner for autonomous drones in SAR missions. One such planning system is the one developed concurrently with this thesis by Solbø in his master's thesis [64]. Therefore, decisions related to message formats from the perception system, as well as determining which information is valuable for such a mission planning system, are jointly discussed and made in collaboration with Solbø.

### 4.1.2 Design

During the specialization project [4], the aim was to follow three key design principles: modularity, platform independence, and testability. In this work, these same guiding principles will be followed in the development of the codebase, and the following section is largely based on the same section from the specialization project [4].

#### Modularity

Modularity in software engineering refers to the ability of a code base to be broken down into distinct parts, each with standardized inputs and outputs. This separation of code components helps minimize dependencies and simplifies maintenance and troubleshooting without disrupting the entire system. To achieve modularity in this work, every code module is divided into its own ROS package, and inputs and outputs rely on standardized ROS topics. ROS, being built on this principle, makes it a good tool for this process [54]. Additionally, the interface

between each node in ROS is messages on topics, and standardized messages are used where appropriate to make the code more understandable to engineers who are unfamiliar with the codebase, but have some experience with ROS

**Platform independence**

Platform independence is an aspect of software development that ensures that the developed code is not bound to specific hardware systems, and can easily be migrated to different hardware platforms. In this project, achieving platform independence is made possible through the use of separate configuration files for each node. This separation of parameters makes it straightforward to change a node's parameters. Moreover, a configuration file for the drone contains all its specifications, including the camera specifications. This allows for quick modifications to the system in order to be compatible with different drones and equipment.

**Testability**

Ensuring that the codebase is easy to test is referred to as code bases testability. This means for instance that it is easy to use simulated data for testing instead of real data. In the developed codebase, this can be done with minimal configuration, allowing for easy offline testing to identify any critical bugs that could affect and possibly destroy expensive hardware. In this project, the use of ROS also facilitates testability. Test nodes can be created to publish simulated sensor data such as camera feeds or GNSS signals. In this work, the code is identical for both drone lab, outdoor testing, and simulations with the only difference being the ROS Olympe bridge's configuration file parameters. For example, a simulated GNSS signal based on the Qualisys motion capture system can be used when flying in the GNSS-denied environment at the lab.

During a drone flight, all data is recorded into rosbags for later evaluation and transformation of ground truth data. This allows for the evaluation of the algorithm's performance during post-processing without unnecessarily loading the CPU during flight. By separating the computational pipeline into a recording and post-processing stage, it is possible to evaluate the algorithms' performance without affecting the computer's performance significantly.

### 4.1.3 ROS Olympe bridge

The Parrot Olympe API and ROS are interfaced using the ROS Olympe bridge. The ROS Olympe bridge was developed in collaboration with Øystein Solbø for the specialization project [4]. The ROS Olympe bridge publishes states from the Anafi FPV drone on output topics at a rate determined by the Olympe API. This publishing rate cannot be changed. Similarly, the ROS Olympe bridge subscribes to command topics that receive commands, causing the bridge to call the appropriate API function. This allows commands to be sent to the drone via ROS topics. The ROS Olympe bridge is mainly left unchanged from the project thesis, however, a new topic that publishes the NED world frame origin's GNSS coordinate is added. All the topics of the ROS Olympe bridge are as shown in Table 4.1.

| Command topics | | |
|---|---|---|
| **Topic name** | **Message type** | **Description** |
| /anafi/cmd_takeoff | std_msgs/Empty | Commands a takeoff |
| /anafi/cmd_land | std_msgs/Empty | Commands a landing |
| /anafi/cmd_emergency | std_msgs/Empty | Immediately cut all motors |
| /anafi/cmd_control_source | std_msgs/Bool | Selects the control source. If message data is true, select the the Olympe API as source. If false, use the SkyController as source |
| /anafi/cmd_rpyt | olympe_bridge_msgs/AttitudeCommand * | Command reference for the internal PID-controller: (roll, pitch, yaw rate, thrust) |
| /anafi/cmd_moveto | olympe_bridge_msgs/MoveToCommand * | Command a desired pose for the drone: (latitude, longitude, altitude, heading) |
| /anafi/cmd_moveto_ned_position | geometry_msgs/PointStamped | Command a desired position in NED for the drone: (x, y, z) |
| /anafi/cmd_moveby | olympe_bridge_msgs/MoveByCommand * | Move the drone relative to the current pose: (dx, dy, dz, dyaw) |
| /anafi/cmd_camera | olympe_bridge_msgs/CameraCommand * | Command the camera gimbal orientation, zoom level, and start/stop recording to internal memory card |
| **Output topics** | | |
| **Topic name** | **Message type** | **Description** |
| /anafi/image | sensor_msgs/Image | Image from the gimbal camera |
| /anafi/time | std_msgs/Time | Anafi drone timestamp |
| /anafi/attitude | geometry_msg/QuaternionStamped | Attitude of Anafi drone in quaternions (x,y,z,w) |
| /anafi/gnss_location | sensor_msgs/NavSatFix | GNSS location (lon, lat, alt) |
| /anafi/ned_pose_from_gnss | geometry_msgs/PointStamped | GNSS location in NED. The first GNSS-message used to initialize the frame origin (x, y, z) |
| /anafi/ned_frame_gnss_origin | geometry_msg/Vector3Stamped | The NED frame's origin GNSS coordinates. |
| /anafi/height | olympe_bridge_msgs/Float32Stamped * | Distance from the ground from barometer (z) |
| /anafi/optical_flow_velocities | geometry_msg/Vector3Stamped | Speed estimate from the optical flow camera (u, v, w) |
| /anafi/polled_body_velocities | geometry_msg/TwistStamped | Speed estimate from the internal EKF of the Anafi drone (u, v, w) |
| /anafi/link_throughput | std_msgs/UInt16 | Connection throughput (b/s) |
| /anafi/link_quality | std_msgs/UInt8 | Signal quality [0=bad, 5=good] |
| /anafi/wifi_rssi | std_msgs/UInt8 | Signal strength [-100=bad, 0=good] (dBm) |
| /anafi/msg_latency | std_msgs/Float64 | Message latency of rpyt-commands (s) |
| /anafi/battery | std_msgs/UInt8 | Battery percentage [0=empty, 100=full] |
| /anafi/state | std_msgs/String | Anafi state ['LANDED', 'MOTOR_RAMPING', 'TAKINGOFF', 'HOVERING', 'FLYING', 'LANDING', 'EMERGENCY'] |
| /anafi/pose | geometry_msg/PoseStamped | Timestamped message with the drone pose |
| /anafi/odometry | nav_msgs/Odometry | Drone odometry |
| /anafi/rpy | geometry_msg/Vector3Stamped | The roll, pitch yaw angled from the internal EKF of the Anafi drone |
| /skycontroller/command | olympe_bridge_msgs/SkyControllerCommand * | If the SkyController3 is used to control the drone, it publishes the commands from the controller |

**Table 4.1:** Command topics and output topics from the ROS Olympe bridge. Message types marked with '*' are custom messages. The table is a cooperation with the author of [5] and is originally presented in [4].

## 4.2 YOLO network used in search

### 4.2.1 Network training

The YOLOv8 network architecture was chosen to detect people and objects of interest in the water. The network was selected as a result of its high performance while at the same time having small models and a fast inference speed. The YOLOv8 ROS wrapper, network training, and dataset preparation will all be covered in the following section.

#### Dataset preparation

For model training and validation, the two datasets SeaDroneSea and AFO from sections 3.3.1 and 3.3.2 were selected. It was necessary to convert both datasets to the YOLOv8 dataset format. The YOLOv8 dataset format requires that the dataset be split into two separate folders: train and validation. In each of the two folders, there must be two additional folders: images and annotations. The images in the train set should be placed in the 'train/images' folder. In the corresponding 'train/annotation' folder, there should be .txt files that contain the annotations. The same thing holds for the validation images. The .txt files should have the same name as the

images they are annotating and should contain one line for each annotation in the following format:

class_id x y width height

Here, class_id represents a unique number assigned to the object class, while x and y denote the normalized image coordinates of the center of the bounding box. These coordinates can range in the interval (0.0 to 1.0]. Width and height represent the width and height of the bounding box, respectively, and does also range between (0.0, 1.0] The dataset format is illustrated in Figure 4.2.

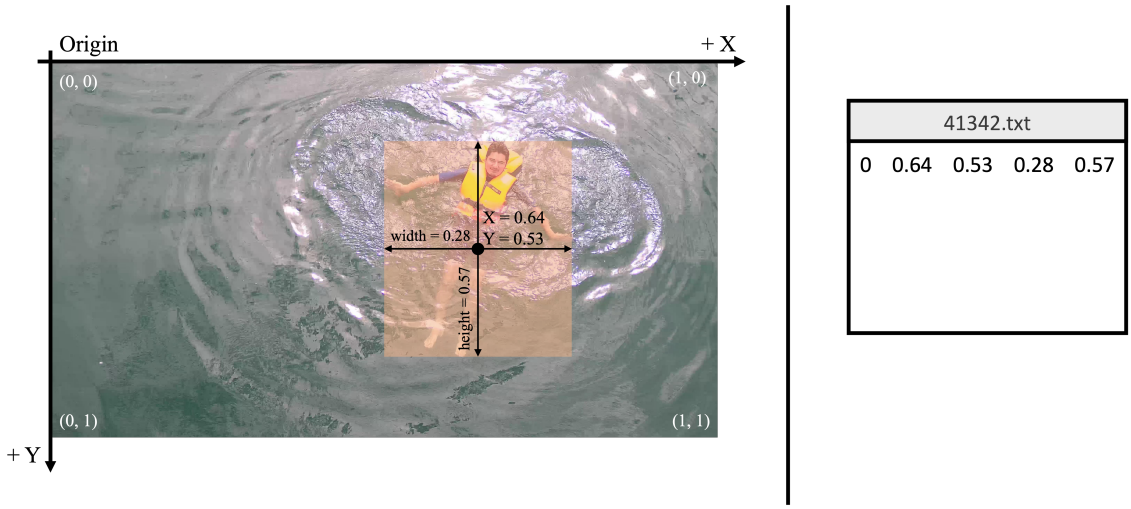



**Figure 4.2:** YOLOv8 annotations format. The image in the illustration is from the SeaDroneSea dataset [55]. Image inspired by [46].

The dataset format also requires a .yaml file in the dataset-root folder that lists the number of classes and the class names. The names of the classes in the class names list in the .yaml file should correspond to their class ids. The paths to the train/images folder and the validation/images folder should also be included in the .yaml file.

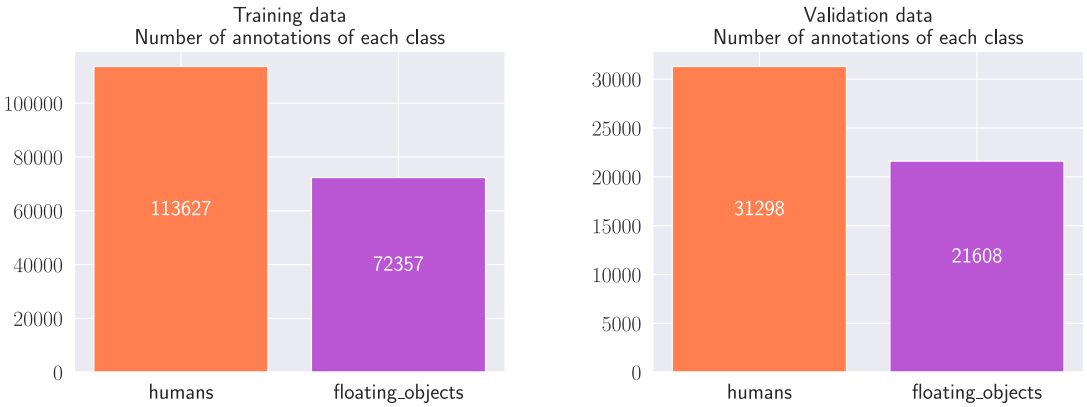
The raw AFO dataset was divided into three subfolders, each containing several images. The train-validation-test split of the images, along with the annotations, are described in a .json file. To transform the dataset into the YOLOv8 dataset format, a Python script was developed.

The train and validation subfolders of the SeaDroneSea dataset each have their annotation .json file in COCO[1] annotation format. Using a GitHub package [65] created by YOLOv8's creators, Ultralytics, the COCO format was converted to the YOLOv8 format. Finally, the two datasets that were both in the YOLOv8 format were combined. There are 29325 train images and 8911 validation images in the dataset, with 185984 and 52906 annotations in each respectively.

The dataset was reduced to two classes, namely humans and floating objects, due to the use-case of detections in SAR operations. This decision was based on two main factors. First, the original datasets were constructed with different annotation classes, which made it challenging to merge them entirely without getting hardly uneven classes. As a result, the floating_objects class was introduced as a superclass that encompasses a variety of floating objects,

[1] `https://cocodataset.org/#format-data`

such as kayaks, boats, sailboats, and surfboards. Second, irrelevant annotated classes were removed. The distribution of the two classes in the train and validation set can be seen in Figure 4.3




**(a)** Training data

**(b)** Validation data

**Figure 4.3:** Class distributions in YOLOv8 dataset used to train detection network used in search

### Training platform and parameters

When training an ANN, this time a YOLOv8 network, the hyperparameters play an important role in the network's detection capabilities [46]. As the objects in the maritime scene can potentially be very small, it is important to keep the resolution of the images used in the network training large enough. At the same time will very large image resolutions lead to larger inference time and more visual memory (VRAM) usage. The resolution of the training images were therefore chosen to be 1280 pixels, the same as the resolution of the images streamed from the Anafi drone. The YOLOv8 network can only train on rectangular images, however, the implementation will automatically resize images in such a way that objects in the image won't be stretched or warped. Furthermore, as mentioned in Section 2.4.3, the YOLOv8 implementation uses mosaic augmentations, meaning it will stitch four training images together in a mosaic. Hence, the training images are stitched together in a mosaic such as the resolution becomes 1280 x 1280. Potentially empty regions in the image in the stitching process will be colored in a uniform gray color.

Another parameter of importance when training a YOLO network is the batch size. Batch size in the training of a ANN refers to the number of training examples it can pass through the network in one iteration. One pass of all the images through the network is known as an epoch. After each iteration, the gradient is calculated and the network parameters are updated accordingly. The choice of batch size in the training is directly dependent on the VRAM available on the training platform. The VRAM usage in the GPU when training is dependent on several other factors than only the batch size, including image size, network size, and the number of annotations in the training image. The authors of the YOLOv8 algorithm recommend using the largest batch size possible that the training platform allows. This is to reduce training time and get better resource utilization rather than getting better detection performance. In addition, Keskar et al. mention in [66] that large batch sizes will tend towards sharp minimizers of the cost function, in which will lead to poor generalization. On the other hand, the

authors state that a smaller batch size will tend towards flat minimizers for the cost function, and hence generalize better. The training platform in this thesis was as described in Section 3.2 a Komplett Khameleon P9 with 8GB VRAM. Hence, the maximum batch size the hardware could handle with the image size of 1280 and the number of annotations in the training data was 4. The low batch size resulted in a slow training process, where one epoch took an average of 40 minutes. It should be noted that it was initially tested to train the network on a free GPU back-end such as Google Colab[2] or Kaggle Notebooks[3] with larger VRAMs available, however, the large dataset caused troubles with loading speeds and resulted in running time of single epochs to several hours.

The number of epochs in the network training is also a parameter of great significance. In the training of the network, the max amount of epochs was set to 400. With large epoch amounts, each image is processed through the network multiple times and the network is more prone to overfitting. Overfitting happens when the network no longer generalizes well and adjusts its weights to get good detection performance on the training dataset. Hence, in order to prevent this overfitting, the early stopping parameters of the training are set to 5 epochs. This will stop the training process before the max amount of epochs is reached when no significant improvements on the validation dataset score have been seen for 5 epochs.

The YOLOv8 architecture comes in five different model versions; nano, small, medium, large, and x-large. Here the nano model is the smallest with only 3.2 million parameters, and the x-large model is the largest with 68.2 million parameters. In this thesis, only the nano model and the small model were the models tested. This is due to two main reasons, the first of which is that the smaller models are capable of extremely fast inference speeds. The second reason is that the smaller models are using the least amount of VRAM when running on a GPU. Hence, valuable GPU resources are not tied up in the detection process. Furthermore, the network is trained with transfer learning from a pre-trained model. The authors of YOLOv8 have released versions of all five models pre-trained on the COCO detection dataset [67]. When training a network on custom datasets, the training can either start from these pre-trained networks or be trained from scratch with randomly initialized weights. The authors of YOLOv8 recommend always starting training from these pre-trained networks. When using transfer learning, the pre-trained network can be used as a feature extractor by freezing the weights in the feature extraction layers and only updating the weights in the classification layers. Another option is to fine-tune the pre-trained model by only freezing some of the dense feature extraction layers or opening up the complete network [68]. Layer freezing was not used in the training process in this thesis for two reasons. Firstly, it was not supported by the official YOLOv8 implementation. Secondly, users who have manually implemented layer freezing by using callback functions report on the Ultralytics forum[4] that the network performance is not increased with frozen layers. This is also supported by that layer freezing did not improve performance for YOLOv5 either, as reported on the forum[5].

The optimizer used when training a neural network may also impact the accuracy of predictions from the network. In the process of training the network, both adaptive methods such as AdamW which adjust the learning rate during training, as well as the classical Stochastic

[2] https://colab.research.google.com/

[3] https://www.kaggle.com/

[4] https://github.com/ultralytics/ultralytics/issues/793#issuecomment-1510299997

[5] https://github.com/ultralytics/yolov5/issues/1314

Gradient Descent (SGD) algorithm with a fixed learning rate were tested and compared. To visualize the metrics during training, as well as save model parameters over several training runs the MLOps tool called *ClearML* was used. ClearML easily integrates with the YOLOv8 framework.

To evaluate the performance of the networks, the Mean Average Precision (mAP) was used. In this thesis, the mAP50 was the one used in particular. This means that the mAP is calculated at IOU threshold 0.5. Hence, all detections where the predicted bounding box overlaps the true bounding box by at least 50% are accepted as a true positive, and the mAP is calculated accordingly.

### 4.2.2 YOLO ROS wrapper

In order to integrate the YOLOv8 framework into ROS a YOLO ROS wrapper was developed. The wrapper is using the pip installed *ultralytics*-package released and maintained by the developers of the YOLOv8 framework. The Python YOLOv8 API is easy to use and has function calls for example 'train()' and 'predict()' taking a set list of parameters that can be found in the online YOLOv8 documentation[6]. The wrapper is launched as a ROS node with a configuration file. The configuration file sets the path to the YOLO-model weights, topic names, and service names relevant to the wrapper. The wrapper subscribes to an image topic with images the YOLO detector should be applied to, and it publishes the detections on a different topic using a custom message format. The custom output message contains a list of bounding boxes of detected objects together with the detection class, class id, detection confidence, and the image containing the detections. The wrapper also provides a service to start/ stop detections on the input image topic. The wrapper has two additional parameters in the configuration file; confidence_threshold and iou_threshold. The confidence threshold is used to filter out bounding boxes with low detection confidence. The IOU threshold is used to filter out low detection confidence bounding boxes in the case where two or more bounding boxes overlap with an IOU of more than the threshold. The wrapper is set up in such a way that several wrapper nodes can be launched simultaneously running with a different network configuration and publishing the result on different topics. The ROS YOLOv8 wrapper is public available at GitHub[7].

The YOLO ROS node which is used in the search is set up to publish all detections with a confidence level larger than 10% and will filter away all lower confidence bounding boxes with an IOU overlap with a higher confidence bounding box of more than 50%. The reason for choosing a low confidence threshold for the search node is due to all bounding boxes being processed through the trackers. In this thesis, the search YOLO node is manually toggled on and off with a terminal service call to the node. However, the toggling of the search YOLO node is intended to be controlled by a mission planner for autonomous drones in SAR missions.

## 4.3 Multi Object tracking

In this thesis, a Multi Object Tracking is employed to link multiple detections to the same target and increase confidence in bounding box detections. Both the SORT tracker and the DeepSORT tracker are tested. The confidence in the tracks is computed using an Exponential Weighted Moving Average (EWMA) model. The formula for the EWMA-model is as in (4.1).

[6] https://docs.ultralytics.com/

[7] https://github.com/simenallum/yolov8_ros

Here the $\alpha$ is the weight, and $r$ is the confidence of the detection in the current timestep. Older observations are given less weight in the model than more recent ones, and as observations age, this weighting decreases exponentially.

$$EWMA_t = \alpha r_t + (1-\alpha)EWMA_{t-1} \tag{4.1}$$

Additionally, track logic is required to determine when a track is confirmed. In this thesis, this can be adjusted from a configuration file in both of the trackers. A track must receive a certain number of associations to be considered a confirmed track and published. On the other hand, a track is deleted if no measurements have been consecutively associated with it for the maximum age iterations. The configuration files also allow for the change of this variable.

### 4.3.1 SORT tracker

The SORT tracker implementation used is based on the official implementation by the authors of the original algorithm Bewley et al [52]. The original implementation is available on GitHub[8]. The track-handling logic is incorporated by altering the implementation. Additionally, the implementation has been modified so that measurements and tracks will only be associated together if they belong to the same class. To integrate the algorithm into ROS, a ROS wrapper is developed. The wrapper gets bounding box detections on a topic, runs one iteration of the algorithm, and then publishes confirmed tracks with bounding boxes, confidence from the EWMA-model, track ID, and class name. All parameters of importance to the algorithm, such as KF tuning parameters and EWMA-$\alpha$, are adjustable through a configuration file.

### 4.3.2 DeepSORT tracker

The DeepSORT tracker is tested in order to strengthen the data association process. The version used is a combination of two forks of the original GitHub repository[9] developed by the authors of the DeepSORT paper [23]. The first fork is from the GitHub user michael-camilleri[10] which implements the original cost metric from the paper. In the official implementation, the cost metric is calculated by only using the appearance information from the feature embedder. The location of the predicted KF bounding boxes are only used to filter out infeasible assignments based on Mahalalobnis distance. The fork implements the cost metric from the paper which weights the appearance information and the distance between predicted KF bounding boxes and measurements to associate measurements to tracks. The second fork is from the GitHub user levi92[11] who added support for custom feature embedders. The original implementation is using a feature embedder for extracting features for the appearance of pedestrians. The fork makes it easy to use other feature embedders. In this thesis, the embedder used is a CLIP (Contrastive Language-Image Pre-Training) embedder [69]. The CLIP network is a better option than the pre-trained embedder for pedestrian appearance because it has been demonstrated to generalize very well on arbitrary image classification [69]. Therefore, even though the embedder has not been trained on a dataset for feature extractions for this particular use-case of human and floating-object detections in marine environments, it can provide good features for the detections. The two forks are combined and wrapped into a ROS wrapper with the same

[8] `https://github.com/abewley/sort`
[9] `https://github.com/nwojke/deep_sort`
[10] `https://github.com/michael-camilleri/deep_sort`
[11] `https://github.com/levan92/deep_sort_realtime`

functionality as the one for the SORT ROS wrapper with track logic and EWMA confidence calculation.

## 4.4 Safe point generator

Sea-land segmentation is carried out to analyze the drone's surroundings and provide safe landing points. In order to achieve and combine both robust and real-time segmentation, the sea-land segmentation is based on information from two different segmentation modules which are combined in the segmentation master node.

### 4.4.1 Deep Learning based segmentation

The PyTorch deep learning framework was used to utilize GPU acceleration in both the training and inference of the segmentation. This is the same framework as YOLOv8 is using. A GitHub repository from the user *milesial*[12] was used as a starting point for the training, prediction, and evaluation of PyTorch segmentation models. The repository included a very simple model-training iteration loop with logging to the MLOps platform *Weight & Biases*[13] and inference code in addition to the model for the UNet segmentation structure [49]. The author of this thesis added the SegNet segmentation structure [50] to the framework, made available by the user *say4n*[14] on GitHub. Furthermore, the training framework was extended by adding logging of validation images to the MLOps tool, early stopping, the addition of different optimizers, and saving of best models during training.

The cost function used for the training loss calculation was a combination of the binary cross-entropy (BCE) and the dice score. The dice score is calculated according to (4.2), being two times the area of the intersection divided by the sum of the two areas. The BCE for one pixel is calculated according to (4.3), where t is the true class (either 0 or 1), and p is the predicted class probability (ranging from 0 to 1). Using this combination, according to the author of [70], is the best combination of loss functions for image segmentation, providing good pixel-accuracy and model generalization.

$$Dice = \frac{2|A \cap B|}{|A| + |B|} \tag{4.2}$$

$$BCE = -(t \log(p) + (1-t)\log(1-p)) \tag{4.3}$$

The two datasets SWED and Sea-Land segmentation dataset described in Section 3.3.4 and 3.3.5 respectively were used for the training of the segmentation network. The datasets consist of the previously mentioned satellite images, which have a resolution of 8–10 meters per pixel. This is not ideal because the segmentation network's use case is on drone-produced images, where the meters per pixel may be much lower. However, the author did not find any publicly accessible online datasets of sea-land segmentation images from drones in marine environments. It takes a lot of work to create a custom dataset with images from an aerial drone and the corresponding image mask, and the use of satellite images is justified because the features learned while training the network on these images can in most cases be applied

[12] https://github.com/milesial/Pytorch-UNet
[13] https://wandb.ai/site
[14] https://github.com/say4n/pytorch-segnet

to the use case of this thesis as the images are from the same kind of environments.

The images with a resolution of 256 x 256 were converted from the 12-banded and 4-banded .tiff image formats to RGB-colored .png images and saved in a folder. The single-channel image segmentation masks were converted from .tiff images to .png and saved in another folder. The SWED dataset contained a lot of images of either only land or sea. In an effort to balance the dataset to contain an equal amount of images with only sea or land, only the images from the SWED dataset containing between 20 and 80 percent water were used. The final dataset used in the training of the segmentation models included 2964 images with corresponding labels. Furthermore, a test set of 288 images and labels were constructed in the same manner.

During the training of the segmentation networks, the dataset is split into training and validation sets. The dataset split is a deterministic seeded split, with 80% in the training set and 20% in the validation set. The concept of mini-batch training is used, where the training set is split into batches determined by the batch size. The gradient is then calculated after all images in one batch are passed through the network, and the weights are updated accordingly after each batch. Mini-batch training has been shown to provide improved network generalization as well as use much less memory [71]. The batch size during the training is decided in a similar manner as for the YOLO network training and is adjusted to suit the VRAM of the training platform. As the training of UNet structures requires more memory than for the SegNet models, the batch size in the UNet training is, therefore, smaller at 16. The batch size for the training of the SegNet is 32.

Finally, the deep learning-based segmentation structures are wrapped into ROS. The ROS wrapper is set up to use services such that when called, the node responds with the image segmentation mask based on the latest received image from the drone. The wrapper can load both SegNet models and UNet models trained using the framework mentioned earlier. The images are reshaped to 256 x 256 before inference and the generated masks are upsampled to the original resolution using bilinear interpolation.

### 4.4.2 Offline map segmentation

As the image segmentation masks provided by the deep learning segmentation module possibly aren't reliable enough alone, a second source of information is used. The second source of information is downloaded from OpenStreetMap[15] and contains bodies of water in the form of polygon objects. The data is downloaded in WGS84 format. The polygons used in this thesis are cropped to only contain water polygons for the Trondheim municipality to speed up mask generation. The node subscribes to the GNSS location, drone altitude measurement, and the pose from the drone. The node provides a service, and when called it produces a segmentation mask equal to what a segmentation mask from the camera image would look like, however, it is based on the offline water polygon file. To do this the metric ground coverage of the drone's camera is first calculated using the field of view of the camera and the drone's altitude as follows for the horizontal coverage:

$$\text{h\_coverage} = 2 \cdot \text{altitude} \cdot \tan(\frac{\text{hfov}}{2}) \tag{4.4}$$

[15] `https://osmdata.openstreetmap.de/data/water-polygons.html`

Furthermore, by extracting a subsection of the map, centered around the drone's GNSS location, converting the submap to a pixel grid of fixed size, rotating the map according to the yaw orientation of the drone, and then calculating the pixels per meter in the pixel grid, the map of correct metric size according to the ground coverage can be extracted. The pixel grid can finally be downsampled or upsampled using cubic interpolation to make a mask with the same size as the camera resolution. The process of creating the masks can be seen in Figure 4.4, starting from the top left corner.

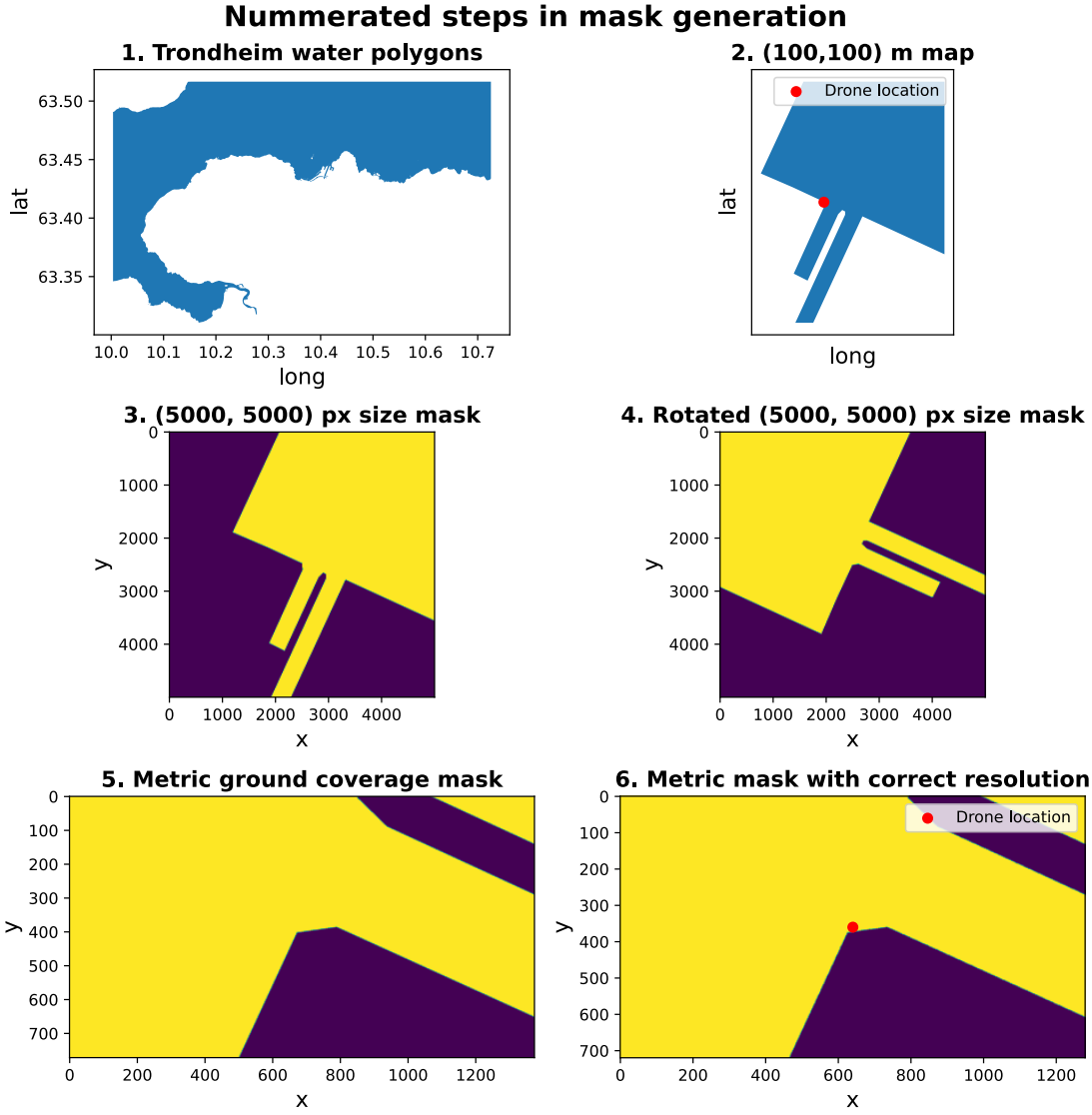



**Figure 4.4:** Nummerated process of offline map mask generation. Drone altitude 40m, and 90 degrees yaw orientation

### 4.4.3 Segmentation master

The segmentation master has service calls to the two nodes to generate masks. The master node calls for mask generation at a fixed frequency adjustable from the configuration file, which for this thesis is set to 1 second. The segmentation master will only generate the two segmentation masks when the drone altitude exceeds 2 meters. When the two masks are generated and

received by the master node, it compares the two masks by calculating their pixel-to-pixel overlap percentage. If they are sufficiently equal, it is assumed that the deep learning mask contains more information than the offline map mask and is therefore used. On the other hand, if they differ very much, it is assumed that using the deep learning mask would be risky and the offline mask is used instead. To analyze and find safe areas in the masks, a user-input minimum safe metric size is converted to a pixel size by using the camera focal length and the drone's altitude by using the idea of similar triangles. A safe pixel is a point that is only surrounded by land pixels with a safe pixel size in both directions. Initially, the center pixel of the image is checked to determine if it is a safe pixel. If not, then a sliding window method with a fixed stride size is used on the segmentation mask to find safe points. The image center together with a safe pixel is illustrated in Figure 4.5. Utilizing the striding, the search is sped up by looking at every n-th point in both mask dimensions and checking if they are safe pixels. As a result, the algorithm will run more quickly because fewer points need to be analyzed; however, this does not necessarily mean the algorithm will find all the safe points in the mask. It could be argued that finding all safe points in each mask is less significant than the algorithm's faster runtime, which enables the generation and analysis of more masks and hence more safe points. The safe points are then sorted according to how far they are from the image's center. The node then publishes the closest point as a secure landing point in image coordinates. The point closest to the center will be the point that is physically closest to the drone under the assumption the camera is facing parallel to the ground and that the ground is sufficiently flat.

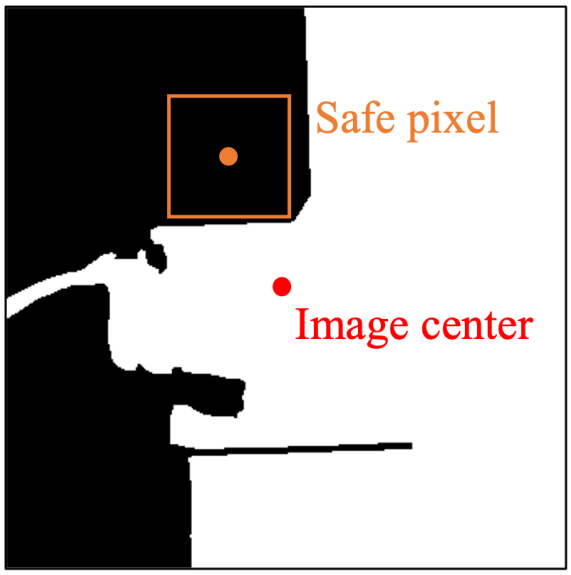



**Figure 4.5:** Safe pixel in segmentation mask. The orange safe pixel is surrounded by only black "land" pixels.

## 4.5 Pixel to coordinate transformer

This node performs georeferencing by transforming points given in pixels in image coordinates to metric world coordinates given in the NED frame $n$. It transforms both safe points generated by the segmentation master node as well as confirmed detections from the MOT tracker node. Points are first transformed into the camera frame $c$ and then into the world frame $n$. To convert points from image coordinates given in pixels to a metric distance given in camera coordinates $c$ two different algorithms were tested. One based on an algorithm originally proposed by

Sundvoll in [1], and a new algorithm using the camera's Field Of View (FOV).

### 4.5.1 Image coordinates to camera coordinates

#### Using similar triangles

This is the algorithm originally proposed by Sundvoll in [1]. It was used to transform bounding box detections given in pixels to a metric distance in the camera coordinate frame $c$ in the author's specialization project [4], as well as in the theses [2, 3]. The algorithm is modified to use drone altitude and can be seen in algorithm 1. The algorithm is using the drone altitude as an estimate for the distance from the object to the drone, under the assumption that objects are located on the ground and that the world is sufficiently flat. This assumption will be valid in the cases of human and floating object detection as they are located in the water, however, may not always be true for the safe points.

**Algorithm 1** Image coordinates to camera coordinates using similar triangles.

Let $x^i$, $y^i$ be the image coordinate of the point
Let $o_x$, $o_y$ be the image coordinate of the image center
Let a be the measured drone altitude
Let f be the camera focal length given in pixels
**procedure** IMAGETOCAMERACOORDINATES($x^i$, $y^i$, $o_x, o_y, a, f$)
    $\hat{z}^c \leftarrow a$ ▷ Object is located on the ground
    $d_x \leftarrow o_x - x^i$
    $d_y \leftarrow o_y - y^i$
    $\hat{x}^c \leftarrow \frac{\hat{z}^c d_x}{f}$ ▷ Similar triangles
    $\hat{y}^c \leftarrow \frac{\hat{z}^c d_y}{f}$ ▷ Similar triangles
    **return** $\hat{x}^c, \hat{y}^c, \hat{z}^c$
**end procedure**

#### Using the camera's field of view

This algorithm is using the camera's FOV to transform from pixels in image coordinates to a metric distance given in the camera coordinate frame $c$. The algorithm can be seen in algorithm 2. It is based on two assumptions. The first one is the same assumption as algorithm 1, namely that objects are located on the ground and that the ground is sufficiently flat. The second one is that the FOV degrees are linearly distanced throughout the cameras FOV in both the horizontal and vertical direction.

**Algorithm 2** Image coordinates to camera coordinates using FOV.

Let $x^i$, $y^i$ be the image coordinate of the point
Let $o_x$, $o_y$ be the image coordinate of the image center
Let a be the measured drone altitude
Let $v_{fov}$ be the camera vertical field of view given in radians
Let $h_{fov}$ be the camera horizontal field of view given in radians
**procedure** IMAGETOCAMERACOORDINATES($x^i$, $y^i$, $o_x, o_y, a, v_{fov}, h_{fov}$)
    $\hat{z}^c \leftarrow a$ ▷ Object is located on the ground
    $d_x \leftarrow x^i - o_x$
    $d_y \leftarrow y^i - o_y$
    x_rad_per_pixel $\leftarrow \frac{h_{fov}}{2 \cdot o_x}$
    y_rad_per_pixel $\leftarrow \frac{v_{fov}}{2 \cdot o_y}$
    x_rad_from_center $\leftarrow x\_rad\_per\_pixel \cdot d_x$
    y_rad_from_center $\leftarrow y\_rad\_per\_pixel \cdot d_y$
    $\hat{x}^c \leftarrow a \cdot tan$(x_rad_from_center)
    $\hat{y}^c \leftarrow a \cdot tan$(y_rad_from_center)
    **return** $\hat{x}^c, \hat{y}^c, \hat{z}^c$
**end procedure**

### 4.5.2 Camera coordinates to world coordinates

The camera of the drone is aimed straight down in the world frame, and the internal controller of the Anafi drone running at 200Hz is responsible for controlling the gimbal angle such that it is pointing straight down at all times. Hence to convert from the camera frame to the world frame the following sequence is performed:

1. Rotate 90 degrees counterclockwise around the z-axis.
2. Translate 70 mm backward in the x direction as the camera is mounted 70 mm in front of the center of the drone.
3. Rotate yaw degrees around the z-axis. Yaw angle is reported by the drone relative to magnetic north.
4. Translate frame with the GNSS position given in NED frame.

Mathematically, using the notation from Section 2.1.1 this can be described as:

$$\mathbf{p}^w = \mathbf{R}_{z,\psi}(\mathbf{R}_{z,90}\mathbf{p}^c + \mathbf{t}_c^b) + \mathbf{t}_b^w. \tag{4.5}$$

All the transforms between the different frames of reference are handled by the transform publisher node. The transform publisher node is an extended and rebuilt version of the same node from the project thesis [4]. The node is utilizing the native ROS functionality for creating and applying transforms, the *tf2* package. The transform publisher node creates the necessary transformations based on the drone pose, the NED coordinates, and the drone altitude. The transformations are available in a "tf-tree" as a time-stamped transformation. When other nodes require the transformation of a point or a pose to a different frame of reference, they utilize the tf-tree and verify the availability of a viable transformation between the frames for the requested timestamp. If a viable transformation is found, the transformation is applied. By adjusting the maximum allowed time difference between the transform and the time stamp, the criteria for what constitutes a valid transformation can be changed. This way the processing time of the developed algorithms can be compensated for in the transforms. Both safe

points and the detection of humans and floating objects are time-stamped with when they were detected. When the detections are to be converted from the camera frame $c$ to the world frame $n$ the appropriate transformation from the time of detection can be applied. This way the processing time of the tracker and the safe point generation is less important as long as the tracker and safe point generator are able to produce correct results. The process of converting points from pixels in image coordinates to a metric distance in camera coordinates requires the drone altitude. Hence in order to access this information as a function of time, the transform publisher also creates a transform that stores the altitude as a function of time. Hence, when the algorithm needs the drone altitude for a given time, it can access the tf-tree.

The quality of the transformations in the system is ultimately dependent on the accuracies of both the drone's ability to deliver fast pose and GNSS measurements.

## 4.6 Helipad tracker

The author created a helipad tracking system for the specialization project [4] that was able to track the drone's position in relation to a helipad. The estimations are done directly in the drone body frame to make guidance easier when preparing the drone for landing. The system employs a CV KF, with measurements from two different camera-based position estimators, GNSS measurements, velocity measurements, and an internal drone altitude estimate. Although largely unchanged, this system has been modified to be compatible with the code written for this thesis. For more details, the reader is referred to the specialization project in [4]; this section will only cover the changes to this subsystem.

### 4.6.1 AprilTag based position estimation

In the specialization project [4], this module provided position estimates with great accuracy and speed. The module detects fiducial markers, AprilTags [15], in the camera image with known 3D coordinates in the world coordinate system. The position of the drone in relation to the world coordinate system can be deduced by solving the Perspective-n-Point problem. One of the suggested improvements from the specialization project was to use larger AprilTags in order to increase the detection performance at larger altitudes. This has been enhanced for this thesis. Figure 4.6 shows the new helipad layout along with the AprilTag IDs. Table 4.2 shows the distances between the AprilTag centers and the origin of the world coordinate system being the helipad's center. The new AprilTags are from the tag-family *tag36h11* and was made available for download from *AprilRobotics* on GitHub[16]. The node has been updated with a control signal service that can turn the node on and off. The input images into the node are downsampled to 10Hz to reduce the computational requirements of the node while at the same time giving position estimates at a high frequency.

[16] `https://github.com/AprilRobotics/apriltag-imgs`

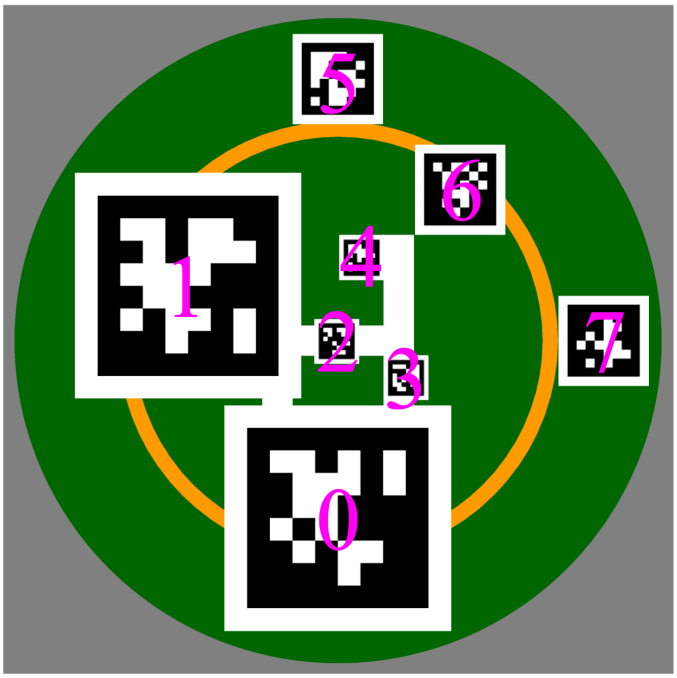



**Figure 4.6:** Helipad with AprilTags and IDs

| AprilTag ID | X | Y | AprilTag side lengths |
|---|---|---|---|
| 0 | -22 cm | 0 | 28 cm |
| 1 | 6.9 cm | -18.5 cm | 28 cm |
| 2 | 0 cm | 0 cm | 5.6 cm |
| 3 | -4.8 cm | 8.9 cm | 5.6 cm |
| 4 | 11.1 cm | 3.3 cm | 5.6 cm |
| 5 | 31.2 cm | 0 cm | 11.2 cm |
| 6 | 20 cm | 15.5 cm | 11.2 cm |
| 7 | 0 cm | 31.8 cm | 11.2 cm |

**Table 4.2:** Distances to centers of AprilTags. Given in the world coordinate system

### 4.6.2 DNN-CV based position estimation

This node also infers position estimates by detection of the helipad using deep learning in the form of a YOLO network. The drone's position in relation to the helipad can be estimated by using similar triangles. The suggested improvement for this node was to retrain the YOLO network used to detect the helipad in order to remove an offset in the estimates. The YOLO network used in the specialization project was a YOLOv4 architecture trained by Hove in [2] with low-resolution images from the previously used AR2.0 drone. For the bounding box detections in this thesis, a new YOLOv8 network was trained using labeled images of the helipad with the new AprilTag design. The images were captured using the frontal gimbal-mounted camera on the Anafi drone.

The YOLO ROS node which is used to detect the helipad is configured to publish all detections with confidence larger than 95% and will filter away all lower confidence bounding boxes with an IOU overlap with a higher confidence bounding box of more than 1%. By setting

the confidence threshold large at 95%, low confidence false detections are filtered away. Furthermore, with a 1% IOU threshold, the node will only publish the most dominant bounding box enclosing the helipad.

### Building dataset

To train the new YOLOv8 network, a dataset containing images of the helipad is required. The images have to be annotated with labels in the form of bounding boxes encapsulating the helipad. The author captured and hand-annotated 2014 images from the frontal gimbal-mounted camera of the Anafi drone. All images have a resolution of 1280 x 720 pixels, which is the maximum resolution available while streaming images from the drone. The images were captured by manually controlling the drone 0.7m to 30m over the physical helipad while saving images at 1 Hz. In an effort to generalize the dataset, it contains images from several different environments:

- Lab environment with and without mattresses on the floor.
- Indoor environment with dark stone floor.
- Outside with cloudy weather.
- Outside with sunny weather.

The outdoor images were captured in Høyskoleparken, in front of Hovedbygningen at NTNU Trondheim. The dataset was labeled using the online labeling tool called *Roboflow* [72]. Each image is annotated by manually drawing a bounding box around the helipad in the image and specifying the class of the helipad. The use of the Roboflow software during the labeling process can be seen in Figure 4.7.


**Figure 4.7:** The helipad dataset labeling process in Roboflow

The dataset is split into the train-validation-test split of 70-20-10 percent respectively. Furthermore, the trainset is augmented by applying an upside-down rotation of all images effectively doubling the number of trainset images. Both the dataset split and the augmentation are done in Roboflow. The image augmentations are kept at a minimum in the dataset-preparation phase as the YOLOv8 framework also performs online image augmentation in the training

phase. The image augmentations result in a final dataset with 2100 images in the training set, 400 images in the validation set, and 206 images in the test set. This corresponds to a train-validation-test split of 78-15-8 percent respectively. The dataset is publicly available for download at Roboflow and can be found at [73].

#### Model training

Very much of the reasoning for the choice of the parameters is the same as for the training of the search network in Subsection 4.2.1, hence this section will just shortly summarize the training parameters.

As the helipad potentially can be very small in the images, the resolution is kept high at 1280 pixels. Furthermore, the batch size can this time be larger at 8 as the images do not contain that many annotations in each image. Two different networks are trained and tested; the nano network and the small network. The reason why only the smaller network architectures are tested is due to the requirement of real-time image processing, while at the same time not requiring too much VRAM usage. The helipad detection networks are trained using transfer learning by fine-tuning the pre-trained networks released by the authors of YOLOv8. The networks are trained for 400 epochs, with early stopping patience set to 5 epochs. Lastly, the performance of the network is again evaluated by looking at the mAP for the test dataset.

### 4.6.3 Kalman filter estimation

The Kalman Filter is a Constant Velocity model filter used to fuse the two camera-based position estimators together with GNSS-measurements, velocity measurements, and in the landing phase an altitude measurement. The Kalman filter is largely unchanged, and the filter tuning is the same as in the project thesis. However, a new service is added to enable and disable GNSS measurement correction in the filter. The reason is that lower-accuracy GNSS measurements can act as noise into the filter when the camera-based estimators are delivering high-accuracy measurements to it. This was an enhancement based on the results in the specialization project [4]. Furthermore, as the GNSS measurements are mainly used in the filter to prevent drift it is used in the filter at a rate of 2Hz.

## 4.7 Perception master

This module, which serves as the perception system's "brain" is in charge of gathering information from other nodes, processing it, and distributing it. It performs three specific tasks: stores safe points, stores both human locations and floating-object locations, and handles toggle logic for correction measurements in the helipad tracker EKF.

The node receives the NED world coordinates of all new detections of humans and floating objects from the MOT processed through the pixel to coordinate transformer. If the detection has a confidence that is lower than a fixed confidence threshold, the track is discarded. The tracks for humans and floating objects are handled in two different ways. If the track is a floating object track, the track_id is checked to see if it already exists. If it does the distance between the old location of the track and the new location is compared and updated if it is larger than a metric threshold. If the track_id is new, the distance is checked between all the previously

seen tracks and the new track. The track is only added to the table if the distance is larger than the same metric threshold. It is in this case assumed that the detection is a re-detection of a previously seen track. The algorithm pseudo-code for adding and updating tracks can be seen in algorithm 3.

**Algorithm 3** Handle new tracks of either human of floating-object type

```
procedure HANDLEINCOMMINGTRACKS(track)
    if track.id seen before then
        if dist(old_location, new_location) > threshold then
            Update table track location
        end if
    else
        if shortest_dist(all_prev_tracks, new_location) > threshold then
            Add new track to table
        end if
    end if
end procedure
```

The human tracks are handled in a similar way as the floating object tracks, however with a different threshold. If a new human track is added to the table or a previously seen track is updated with a new location the human detection location is published. Before the human location is published, the distance from the human to the closest floating object is calculated. This is done to adjust the severity level of human detection. The further away a human detection is from floating object detection, the higher the risk and the higher the priority. The severity level is published together with the location and track_id as a level between 0-2 where 0 represents the lowest risk. The Euclidian distance thresholds used in severity level determination may change dependent on the mission type and is for the testing of this thesis set low at [0m, 5m, 10m] corresponding to level [0, 1, 2]. When new floating objects are detected, the severity level of human detection may change. Hence, all severity levels are checked and the detections are re-published with new severity levels if changed.

A similar approach is taken for the safe points table. When new safe points are received, their distance to all previously safe points is calculated. If the new safe point is located sufficiently far away from the other points, the safe point is added to the table of safe points, and the point is published.

The perception master also implements toggle logic for the correction measurements in the EKF. The toggle logic is needed for two main reasons. First, the GNSS will not act as noise in the filter when the camera-based estimators are available. Second, the camera-based estimators will not run in the background and use valuable computer power when they are not needed. The toggle logic is handled by calculating a heuristic for if the helipad is present in the camera image or not. At 1Hz, the FOV is used to calculate the expected ground coverage of the camera given the estimated EKF altitude. It is then checked whether half the ground coverage in both the vertical and horizontal direction is larger than the absolute value of the EKF estimate in the same direction. If this holds for both directions, it is assumed that at least one-quarter of the helipad is present in the image from the drone. In addition, it is checked if the drone's altitude is low enough for the camera-based detectors to produce estimates. If

all of these three conditions hold, it indicates that the camera-based estimators should be able to produce estimates and hence they are used in the filter. In addition, in order to prevent turning off the camera-based estimators in situations where the drone is really close to the helipad and suddenly disturbed such that the helipad is not visible for a brief movement, the camera-based estimators are always used when the Euclidian distance of the EKF estimate is smaller than 1 meter. On the other hand, the GNSS measurement is used in the filter when the helipad is likely to be too far away from the drone for it to be visible in the camera images. The FOV, ground coverage, and EKF estimate are shown in the XZ plane in Figure 4.8. In the figure, the GNSS measurement will be used as corrections as the EKF estimate is larger than half the ground coverage. It is therefore not likely that the camera-based estimators will be able to produce any estimates. The ground coverage can be calculated as in (4.4).

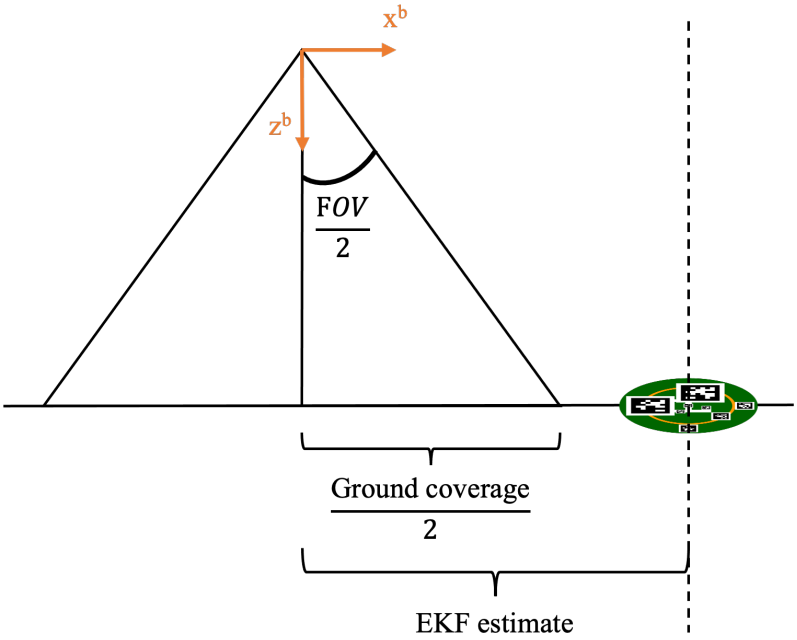



**Figure 4.8:** Toggle logic for correction measurement in helipad tracking EKF. XZ plane visualized.

# Chapter 5

# Results

This section contains the result of the testing and evaluation of the different modules made for this thesis. The chapter presents the training results from both YOLO networks and the segmentation network training. Then the results from testing the multi-object trackers on idealized detection data are presented before the lab testing of the pixel-to-coordinate transformer is evaluated. Further, the helipad tracker, safe point generator, and search subsystems are tested on outside real-world data and evaluated in their own sections, before the chapter finishes with a section where the performance of the complete perception system is presented.

## 5.1 YOLO networks training

This section contains the results of the YOLO network training for the network used in the search to detect humans and floating objects as well as the network used to detect the helipad.

### 5.1.1 Search detection network

The results from training three different networks are summarized in table 5.1. All three networks are trained with the same dataset with the same train-validation split. The parameters in the model training are as described in Section 4.2.1. Based on the data in the table, the nano-network trained with SGD with a mAP50 score of 0.733 was the one used in this thesis. This is due to having one of the highest mAP50 scores, while at the same time having lower VRAM usage and faster inference speed than the small model. The confusion matrix for the chosen model can be seen in Figure 5.1. Based on the confusion matrix it is clear that the model has most problems with human detection. The model is more prone to false detections of humans rather than false detections of floating objects. Furthermore, the model also makes miss detections of humans. However, the model output is used in the MOT in order to build confidence in detections and link multiple detections together. Hence single miss detections or false detections will not impact the full detection pipeline that much.

| Model type | Optimizer | Inference speed | VRAM usage | Learning rate | mAP50 on test set |
|---|---|---|---|---|---|
| Nano-model | SGD | 9.7 ms | 1.161 GB | 1e-3 | 0.733 |
| Nano-model | AdamW | 9.7 ms | 1.161 GB | Adaptive | 0.704 |
| Small-model | SGD | 22.5 ms | 1.337 GB | 1e-3 | 0.740 |

**Table 5.1:** Training results from YOLO network for search

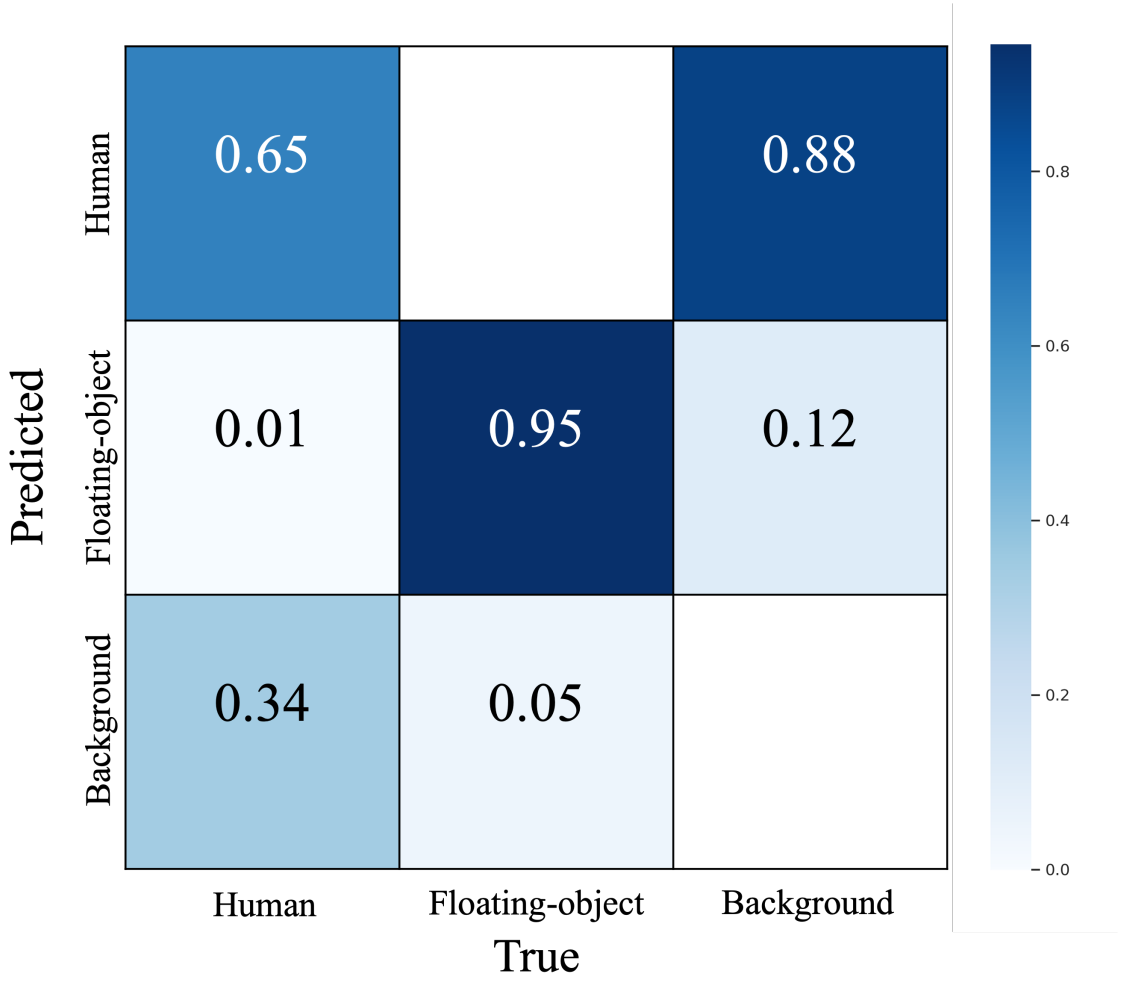



**Figure 5.1:** Confusion matrix for YOLO network used in search

### 5.1.2 Helipad detection network

All models for detecting the helipad performed equally well with very high mAP-scores on the test set. The model training results can be seen in table 5.2, and are all trained with the SGD optimizer with a learning rate set to 0.01. In order to only train on images where the whole helipad is visible, mosaic augmentations were turned off for one of the models. However, no change to the mAP is found by turning off mosaic augmentations in the training process. Furthermore, the small model has equally good mAP scores to the nano-models, however with double the inference time. Based on these results, the nano model with mosaic augmentations is the one used in this thesis.

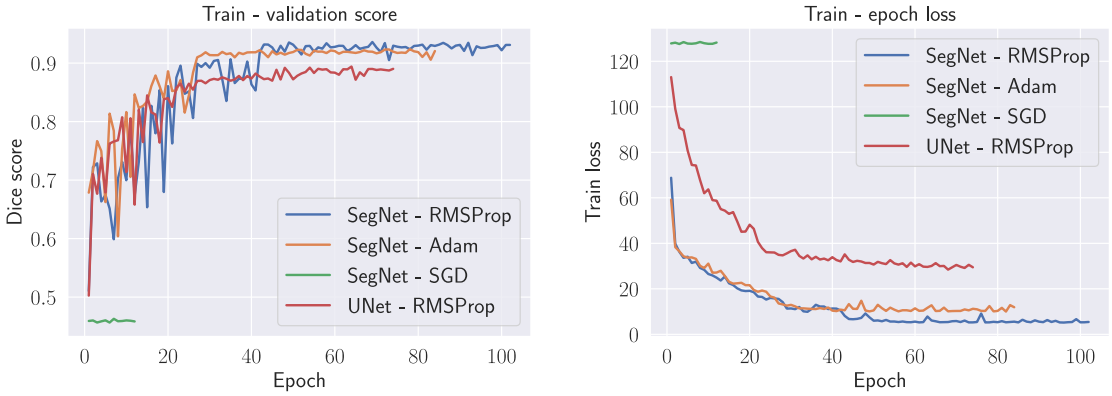
| Model type | mAP50-95 on test set | mAP50 on test set | inference speed |
|---|---|---|---|
| Nano model | 0.970 | 0.995 | 7.2 ms |
| Nano model without mosaic | 0.970 | 0.995 | 7.2 ms |
| Small model | 0.976 | 0.995 | 15.8 ms |

**Table 5.2:** Training results from YOLO network for helipad detection

## 5.2 Segmentation network training

The deep learning-based segmentation network training results will be presented in this section. Using test-set data from the datasets used for network training, the performance of the networks is evaluated. The networks will be evaluated on images from the drone in a later section.

The two different types of networks, SegNet and UNet, were trained with different optimizers. The training epoch loss and the training validation scores are shown in Figure 5.2b and 5.2a respectively. Furthermore, Table 5.3 displays the trained models with numbers 1 through 4. When evaluated on the independent test set, the performances of Models 1, 2, and 4 are all comparable. Model 3, a SegNet model trained with SGD, failed to converge as shown in Figure 5.2, leading to a poor evaluation score as well. As the three other models (1,2, and 4) are trained with adaptive optimizers, the learning rates are reduced when the models' train losses are flatting out. This causes the validation score to fluctuate at the start of the training when there are higher learning rates and to converge to a more stable value when there are lower learning rates.




**(a)** Train validation score

**(b)** Train epoch loss

**Figure 5.2:** Deep learning segmentation models training

| Model Number | Model type | Optimizer | Learning rate / Initial learning rate | Dice score |
|---|---|---|---|---|
| 1 | SegNet | Adam | Adaptive / 0.001 | 90.05 |
| 2 | SegNet | RMSProp | Adaptive / 0.001 | 90.06 |
| 3 | SegNet | SGD | 0.0001 | 44.97 |
| 4 | UNet | RMSProp | Adaptive / 0.001 | 90.79 |

**Table 5.3:** Segmentation models training results

Figure 5.3a shows the network output from model 2 (the SegNet with RMSProp), Figure 5.3b the network output from model 4 (the UNet with RMSProp), and Figure 5.3c the ground truth mask. While both models accurately predict the general shape of the ground truth mask, they differ in their level of detail and their emphasis on various features.

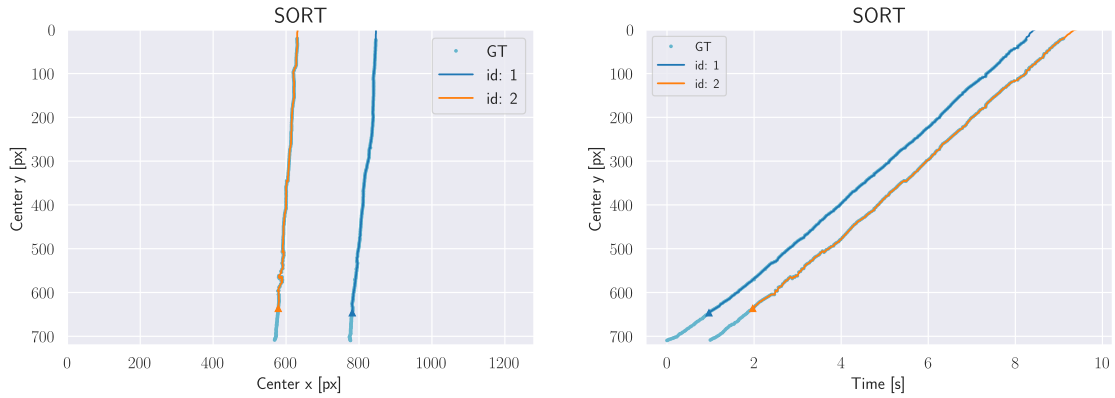
**(a)** Model 2 prediction **(b)** Model 4 prediction **(c)** Ground truth

**Figure 5.3:** Deep learning segmentation models output

## 5.3 Multi Object Trackers on idealized detection data

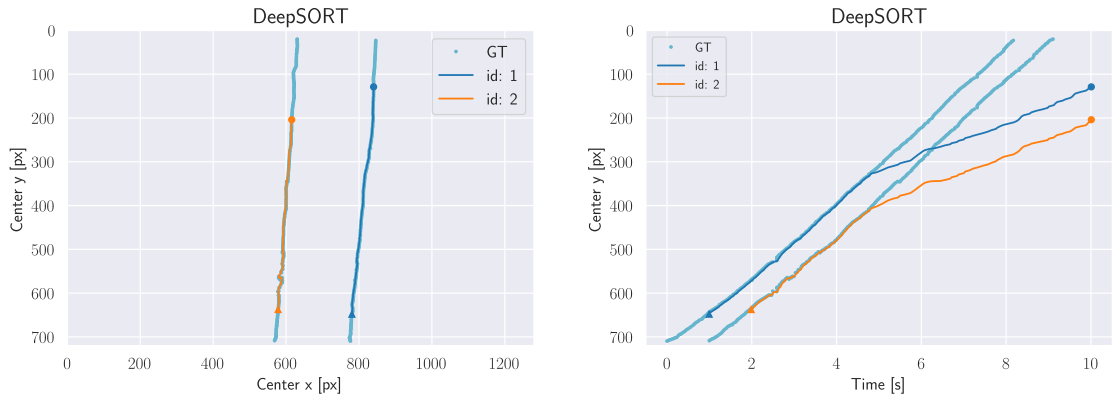
A node that publishes the ground truth bounding boxes from the datasets SeaDroneSea and MOBdrone in Sections 3.3.1 and 3.3.3 respectively, along with the ground truth track ids, was created in order to compare and evaluate the performance of the two tested MOT trackers on idealized detection data. The bounding boxes are published at a frequency of 30 Hz, the same frequency as the bounding boxes from the drone images are published. Figures 5.4 and 5.5 show the tracking performance of the SORT tracker and DeepSORT tracker respectively. The trackers are evaluated on the same dataset, consisting of two human detections which move vertically from the bottom to the top of the image. In the figures, triangles indicate track initiation, meaning the track is confirmed, and the circles indicate track deletion. The left image in both figures shows tracking of the bounding box centers in image coordinates, and the right image are showing the center y positions of the bounding boxes as a function of time. Based on the two figures both trackers manage to only initiate two tracks. However, the tracks in the image coordinate X vs Y plot for the DeepSORT tracker is terminated too early. When looking at the time-dependent plots the DeepSORT tracker is unable to track as a function of time, and will lag behind the ground truth tracks. Both trackers use 1 second to initiate the tracks and mark them as confirmed. This is due to the requirements in the trackers to associate 30 measurements to a track before it is confirmed.

The average runtimes and standard deviation in the runtimes of the update step when a new measurement is presented to the trackers are shown in Table 5.4. The histograms of the runtimes are presented in Figure 5.6. The runtimes are from the same dataset as above, with two human detections with vertical movement. Due to the DeepSORT tracker's requirement to pass the bounding box predictions through a feature embedder, the DeepSORT tracker runs slower than the SORT tracker. By only looking at the average runtimes, both trackers are capable of running at 30Hz, however, problems arise with the DeepSORT tracker's standard deviation. Some iterations have an increased runtime resulting in the tracker "lagging behind" in time. By looking at the histogram of runtimes for the DeepSORT tracker in Figure 5.6b, some of the iterations are seen to use 120 ms.




**(a)** **(b)**

**Figure 5.4:** SORT tracking performance in image coordinates. Vertical movement.

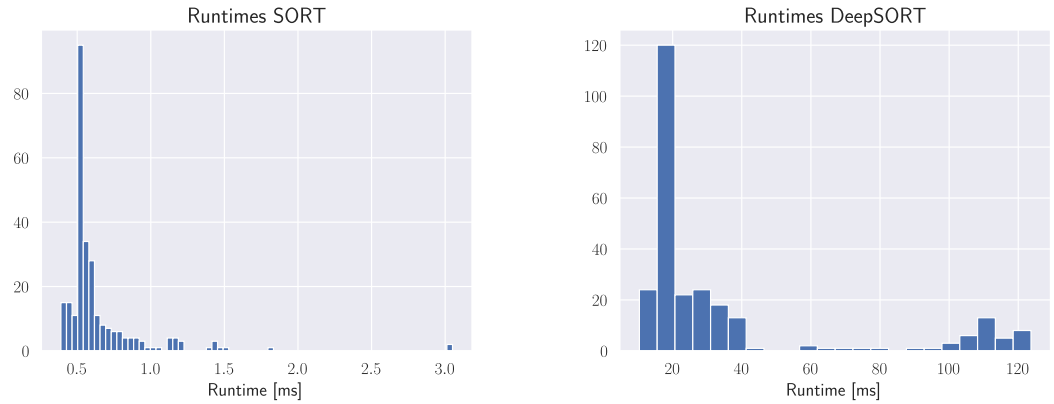



**(a)** **(b)**

**Figure 5.5:** DeepSORT tracking performance in image coordinates. Vertical movement.

| | **SORT** | **DeepSORT** |
|---|---|---|
| Average measurement update runtime | 0.73 ms | 32.45 ms |
| Std. dev in measurement update runtime | 0.41 ms | 33.88 ms |

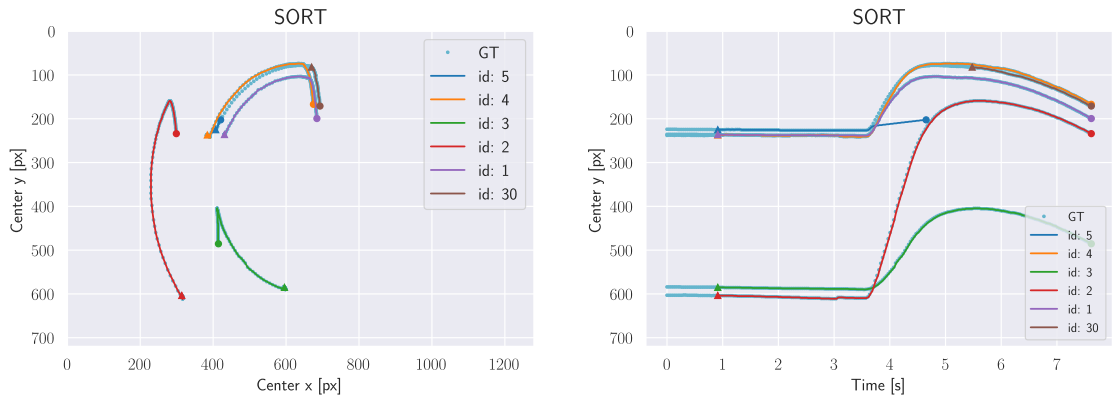
**Table 5.4:** Sort and DeepSORT measurement update runtime metrics.




**(a)** **(b)**

**Figure 5.6:** Histogram of SORT and DeepSORT measurement update runtimes. Note the difference in x-scaling.

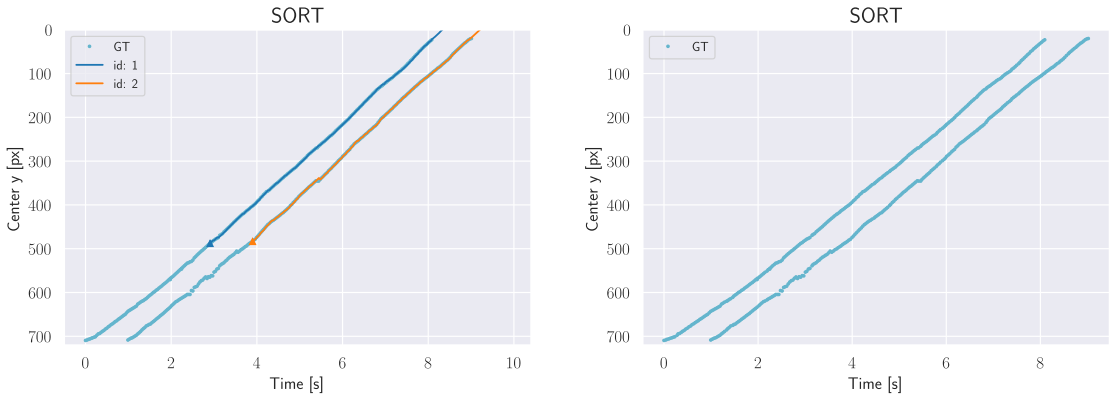
Figure 5.7 shows the SORT tracking performance of both floating objects and humans on a dataset with fast drone yaw motion. In the Figure, the estimated track ids 1, 2, and 3 are floating object detections, while the rest of the tracks are human detections. The tracker is able to track the floating objects and one of the humans correctly throughout the video, however, the other human track is lost and re-initiated under a new track id. This can be seen in Figure 5.7b where the track with id 5 continues with approximately constant y location, even though the ground truth track has y movement upwards in the image. The track is re-initiated with track id 30 once the yaw movement in the camera slows down again.




**(a)** **(b)**

**Figure 5.7:** SORT tracking performance in image coordinates. Fast yaw movement.

Figure 5.8 shows the performance of the SORT tracker when the detection performance is reduced. When the tracker is updated with measurements at a frequency of 10 Hz (every third image contains a detection), the tracking performance is seen in Figure 5.8a. Both humans are tracked correctly throughout the video, however, the tracks uses 3 seconds to be initiated due to the requirement of 30 measurement associations. When the detection performance is reduced even further to 2 Hz in Figure 5.8b, the SORT tracker does not manage to track. None of the two humans are tracked.




**(a)** Measurements at 10Hz

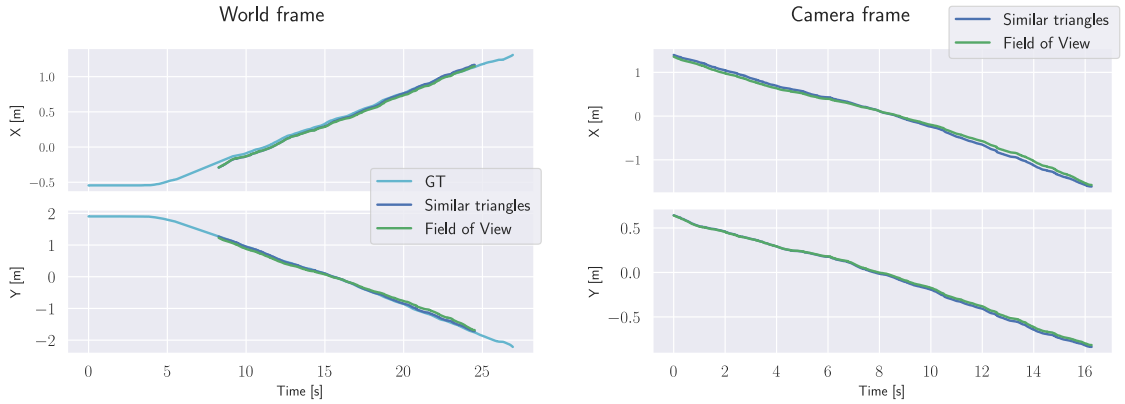
**(b)** Measurements at 2 Hz

**Figure 5.8:** SORT tracking performance in image coordinates under reduced detection frequency.

## 5.4 Pixel-to-coordinate-transformer in the lab

The pixel-to-coordinate-transformer accuracy was evaluated at the drone lab with access to ground truth data. A test node, which detects AprilTags in the image from the drone, and publishes bounding boxes around them was made in order to simulate YOLO bounding box detections. To determine the ground truth location of the AprilTag, three reflective markers were attached to it and the Qualisys Motion Tracker could hence be used to get the ApriTags position. As there are no GNSS signals at the lab, the drone's position reported by Qualisys Motion Tracker was used to simulate GNSS measurements. To assess the performance using ideal measurements, no noise was added to the simulated GNSS measurements. To be able to compare the estimated location of the AprilTag with its ground truth position, the reference frame in Qualisys Motion Tracker was rotated in order for it to be a NED frame with north pointing in the same direction as what the drone reports as north. Finally, a rope was attached to the AprilTag in order to enforce movements.

The performance of the pixel-to-coordinate transformer is shown in Figure 5.9, both in world coordinates with ground truth location and in the camera frame. The accuracy of the two different estimators is comparable, both delivering good estimates. The estimators are generally performing best when the detection is closer to the image center. This is also where the two different estimates are the most equal.




**(a)** World frame **(b)** Camera frame

**Figure 5.9:** Pixel to coordinate transformer with object movement.

In Figure 5.10, the drone is centered above a stationary AprilTag and the drone performs a full yaw rotation. The estimate, from the similar triangles-based estimator, is to some degree influenced by the rotation with a max deviation of approximately 6 cm. This suggests that the drone should ideally be flown with a fixed yaw angle because the transformation depends on the precision of the drone's challenging-to-estimate yaw angle.

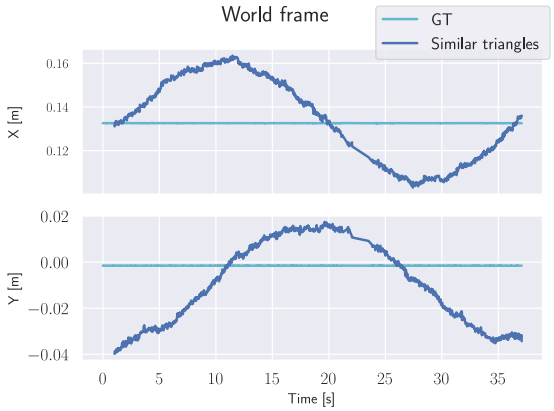



**Figure 5.10:** Pixel to coordinate transformer with yaw motion. Similar triangles-based estimator.

As both two estimators provide estimates with good accuracy, only one of them, the similar-triangle-based estimator, were the one used for outdoor testing.

## 5.5 Helipad tracker subsystem

Figure 5.11 shows the detection performance of the two camera-based estimators under different altitudes ranging from 0 - 35 meters with a vertical drone velocity of 1.5 m/s. In the test, the helipad is static on land, without any rotational motions. The detection performance of the AprilTag-based estimator depends on the detection of the AprilTags and is able to consistently produce estimates of altitudes lower than 14 m. The DNN-CV-based estimator is set to filter out detections of the helipad with a confidence lower than 95%. The DNN-CV-based

estimator produces estimates up to 8 meters. In the figure, it can be seen that the DNN-CV estimates during the final landing phase are unreliable, as the estimated altitude is too large. This is a result of the fact that the entire helipad must be visible in the image in order for the YOLO network to encapsulate the entire helipad with a bounding box. The same problem was discussed in [3] and [4].

Based on these findings, the DNN-CV estimates should be used as corrections in the filter when the drone altitude is in the range of 1 meter to 10 meters. The AprilTag-based estimates should be used as corrections in the range of 0 meters to 10 meters. These are the ranges in which the two camera-based estimators are the most likely to produce accurate estimates, and hence used as altitude requirements in the toggle logic for the EKF correction measurements as discussed in Section 4.7.

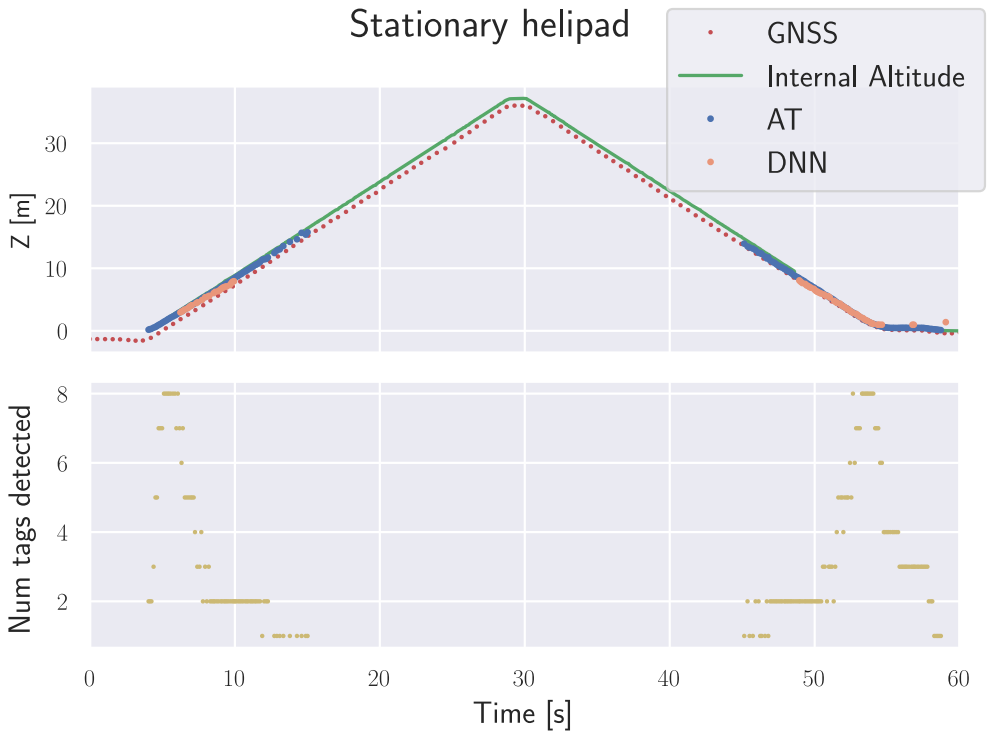
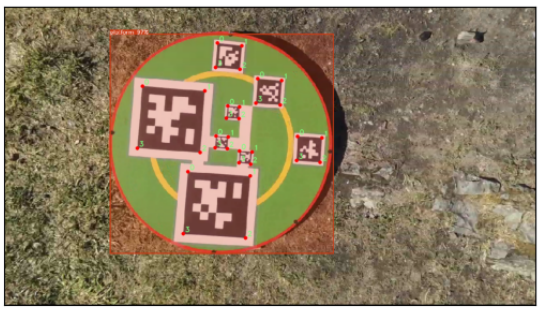



**Figure 5.11:** Detection altitude of helipad

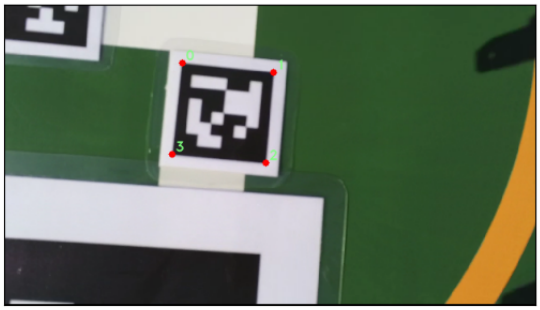
In Figure 5.12a both the bounding box detection of the helipad from the DNN-CV-based estimator and the corners of all 8 Apriltags are detected at an altitude of 1.3 meters. Furthermore, in Figure 5.12b the drone is at an altitude of 8.4 meters, and both estimators are able to detect the helipad and produce estimates. The AprilTag-based estimator can generate an estimate even though only the two larger AprilTags are detected in the image. The minimum detected AprilTags for the estimator to generate estimates is one. In Figure 5.12c, the drone is only 20 centimeters above the helipad. The DNN-CV-based estimator is in this case unable to produce any estimates as the altitude is too low for the full helipad to be visible in the image. However, as one of the three smallest AprilTags is detected in the image, the Apriltag-based estimator is able to produce an estimate. Figure 5.12d shows the detection performance when the helipad is mounted on the ReVolt at sea during the landing phase after a mission. The camera is overexposing the boat and helipad, and hence only the AprilTag-based estimator is

able to produce any estimates.

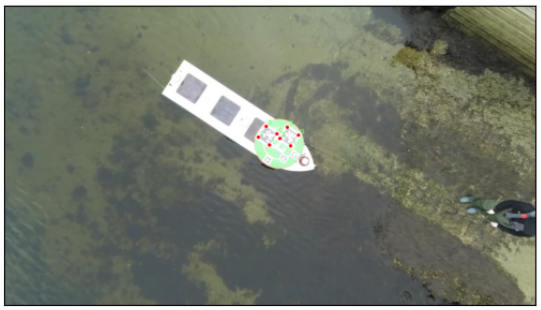


**(a)** 1.3m: Both estimators with detections

**(b)** 8.4m: Both estimators with detections



**(c)** 0.2m: Only AprilTag estimator with detection



**(d)** 6m: Helipad overexposed

**Figure 5.12:** Camera-based helipad tracking from different altitudes

In Figure 5.13 the helipad tracking is shown for a take-off sequence where the helipad is mounted on the ReVolt and exposed to heavy roll motions. As the helipad is mounted on the radar of the ReVolt vessel, the helipad will have a lever arm and hence motions in the XY plane as a result of the roll motion. The DNN-CV measurements are sparser than for a stationary helipad, however as the AprilTags are detected for the whole sequence, the helipad is tracked with high accuracy for the complete take-off sequence.

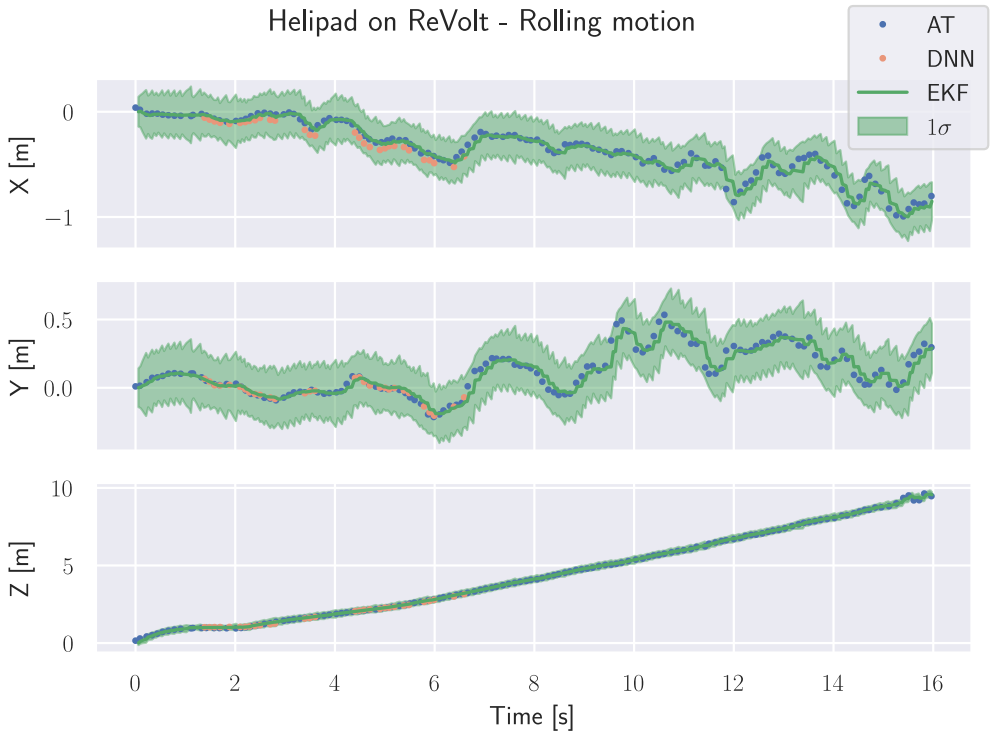
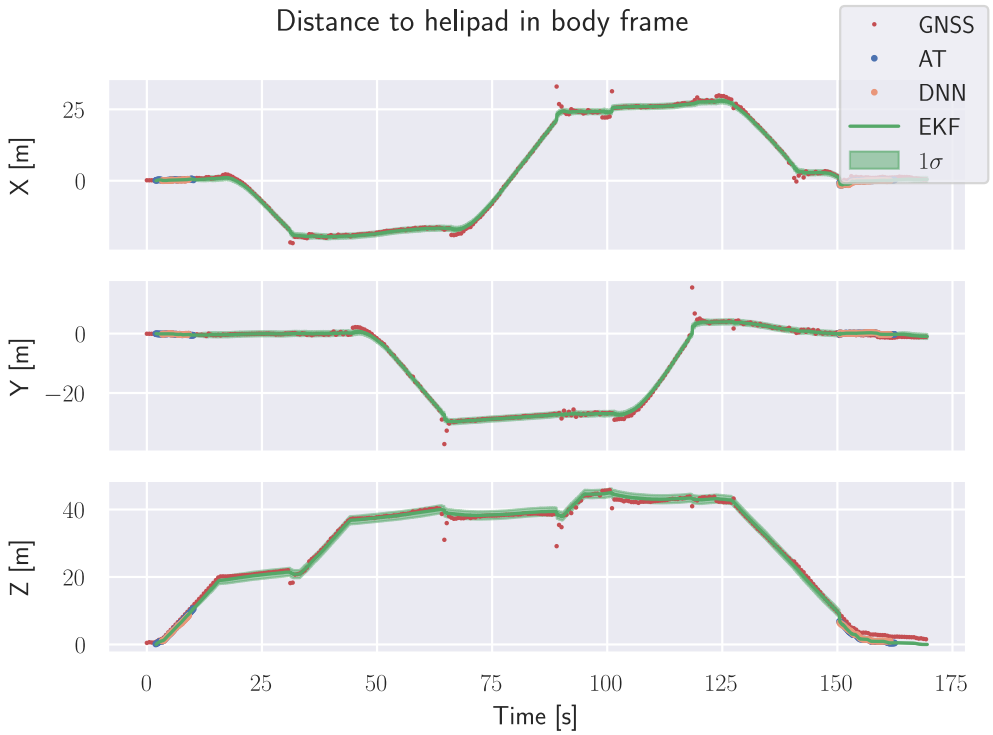
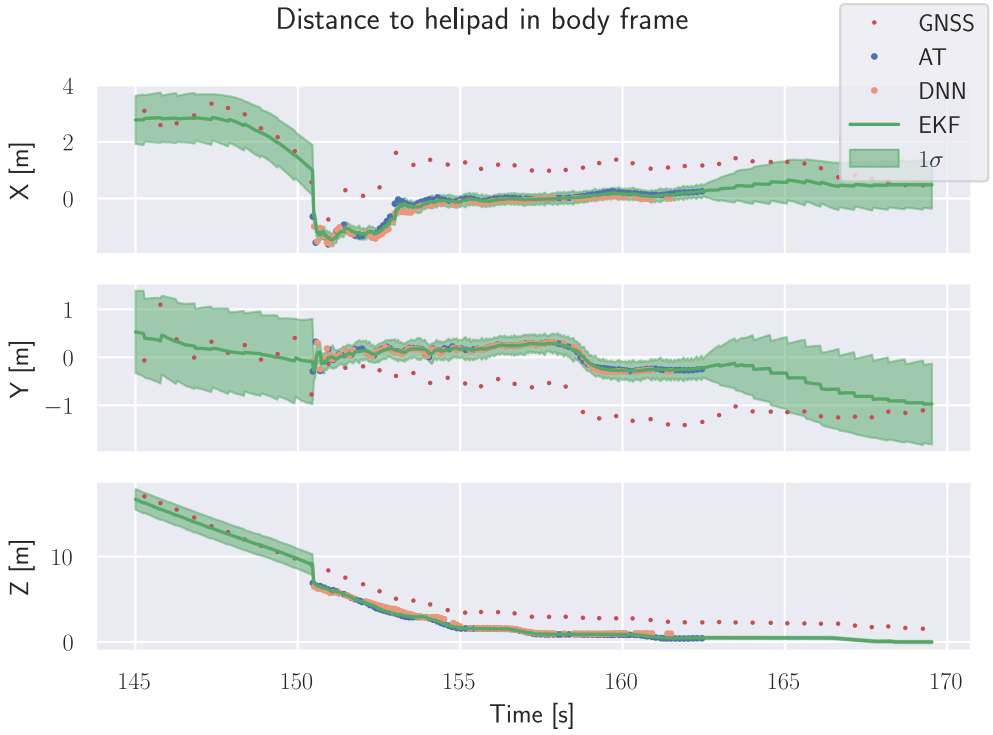



**Figure 5.13:** Helipad tracking with helipad mounted on ReVolt with heavy rolling motions

In Figure 5.14 the helipad tracking is visualized for a drone mission at Korsvika. The drone starts and ends its mission on the helipad, which is stationary at land. Manual control is used to maneuver the drone in a square motion away from the helipad and then back. The toggle logic described in section 4.7 is used to switch between corrections from GNSS and the camera-based estimators. The GNSS measurements are visualized for the full mission length, however, they are only used to correct the filter when the heuristic indicates that it is likely that the helipad is visible in the camera image. The EKF estimate is generally smooth for the whole run, and the tracking seems reasonable. The GNSS measurements can be seen to be far away from the estimate in occasions where the drone comes to a stop. This is due to the drone's need to change its attitude to cancel velocity, and the GNSS feedback is transformed into the body frame using the drone attitude. The covariance of the estimate is low during the full flight. This indicates that the filter is at all times corrected with measurements, and this way does not need to dead recon relying on only the CV filter model. A zoomed-in view of the landing phase can be seen in Figure 5.14b. When the camera-based estimators again are toggled on and are able to produce high accuracy correction estimates, the EKF does not have large step-responses in the estimate. This indicates low filter drift during flight. The covariance in the Z estimate is low in the landing phase due to correction measurements from the internal altitude measurement from the drone.




**(a)** Full mission




**(b)** Zoomed in view of landing phase

**Figure 5.14:** Helipad tracking during a longer flight at Korsvika with stationary helipad on land.

## 5.6 Search subsystem

The search subsystem was tested in Korsvika and Nyhavna in Trondheim. At Korsvika, the drone was flown over a human wearing the immersion suit from Section 3.5.2. Due to the presence of strong currents in the water at Nyhavna and the associated safety concerns, it was not feasible to swim using the immersion suit. Based on the results in Section 5.3, only the SORT tracker was tested on real-world data.

### Search YOLO network detections on drone images

Figure 5.15 shows network predictions from various altitudes. Subfigures 5.15a and 5.15b show correct predictions from the network at drone altitudes of 40 meters and 70 meters respectively. In Figure 5.15c the drone is flown at an altitude of 15 meters over the human and the network wrongly classifies the human detection as a floating object. Figure 5.15d shows a correct human classification from 55 meters, together with false detection of a floating object on land. In addition, in Figure 5.15e the drone is flown at an altitude of 30 meters, and the network predicts two bounding boxes on the human in the water together with a false detection on a rock in the water. Boats are correctly classified as floating objects in Figures 5.15f, 5.15g, and 5.15h from altitudes of 40, 67, and 16 meters respectively. At 67 meters, an orange marker in the water is classified as a human, however, the detection has low detection confidence of 31%. In Figure 5.15i, at an altitude of 9 meters, the network predicts three floating objects at different locations at the boat, rather than correctly predicting one large bounding box.

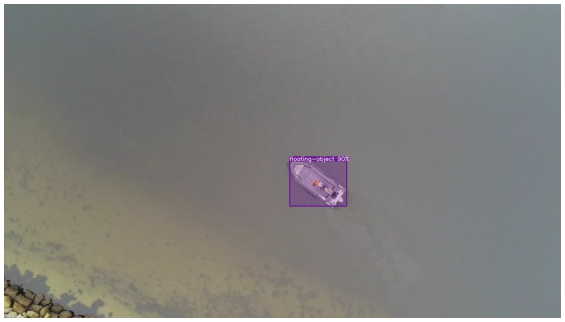
**(a)** 40m: correct human detection

**(b)** 70m: correct human detection

**(c)** 15m: Classifies human as floating object

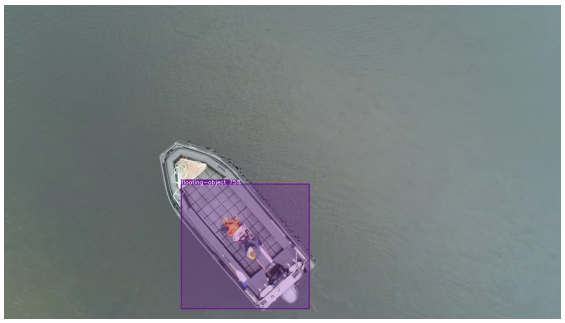
**(d)** 55m: False detection of floating object

**(e)** 30m: False detections of humans



**(f)** 40m: correct detections of floating object

**(g)** 67m: False detection of human



**(h)** 16m: Suboptimal detection of floating object

**(i)** 9m: False detections of floating objects

**Figure 5.15:** Search YOLO network detections from different altitudes. Orange bounding boxes are human detections and purple bounding boxes are floating object detections.

Figures 5.16 and 5.17 shown network predictions processed through the SORT tracker to filter out all detections except for the one for the human. The figures show X and Y given in NED coordinates in the world frame, detection confidence calculated with EWMA-$\alpha$ of 0.5, and drone altitude all as a function of time. When the drone is flown at an altitude between 20 and 70 meters, the network correctly classifies the human with confidence between 26.5% - 81.5%. However, at altitudes between 7 and 20 meters, the human is wrongly classified as a floating object. Below 7 meters, the human is not detected by the network. The testing showed that the confidence of the human detections is generally better for the higher altitude ranges and is highest for the altitude range of 35 to 45 meters.

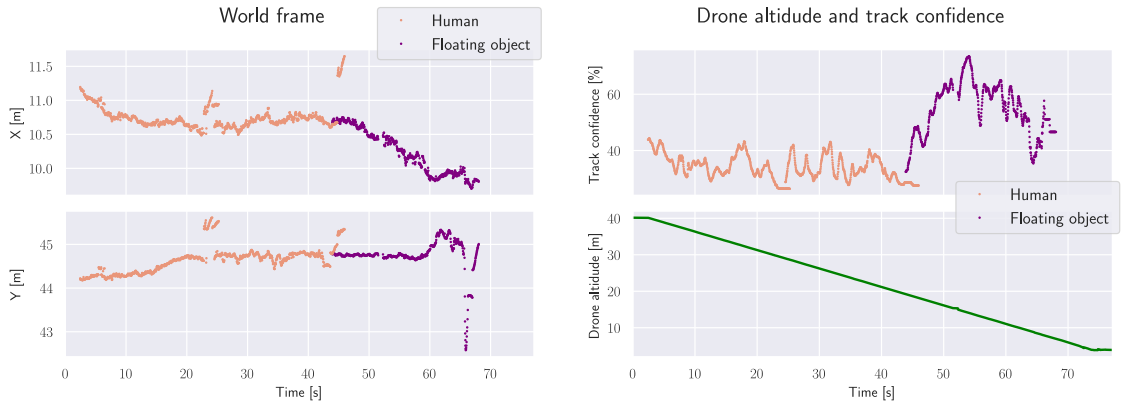


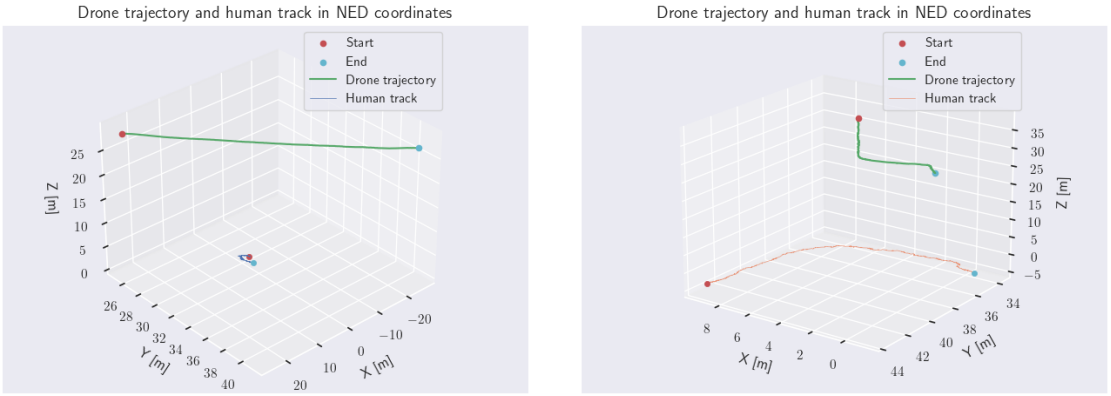

**Figure 5.16:** Search YOLO network detections under slow descent from 40m to 3m with a human in the water


**Figure 5.17:** Search YOLO network detections under ascent from 40m to 70m with a human in the water

## MOT on real-world data

Figure 5.18a show the tracking of a slowly floating human in the water, where the drone slowly flies over the human at a constant altitude with constant speed. The SORT tracker correctly manages to initiate only one track based on the network predictions. Furthermore, the accuracy of the transformation from image coordinates to world coordinates by the pixel-to-coordinate transformer is sufficient as the slowly floating human's location in world coordinates seems reasonable. In Figure 5.18b, the human is slowly swimming, while the drone first descends

from 35 to 25 meters, and then chases after the human in the water. The SORT tracker correctly initiates a track for the human and manages to track the human despite the movements of both the human and the drone.




**(a)** Slowly floating human

**(b)** Swimming human

**Figure 5.18:** Human tracking with drone movement

In Figure 5.19, the drone is flown at a constant altitude of 30 meters with excessive drone movements. The human in the water is floating slowly. When the drone is performing fast movements, the gimbal is not controlled fast enough to compensate for drone rotations by the internal controller. Hence, the gimbal-mounted camera will momentarily follow the drone rotations before the controller compensates for the drone rotation to keep the gimbal pointing straight down in the world frame. This will consequently induce fast movements in the pixel coordinates of detected objects. As a result, the SORT tracker loses track of the human in these cases. However, the track is re-initiated with a new track_id when the gimbal is again stabilized in the world frame.

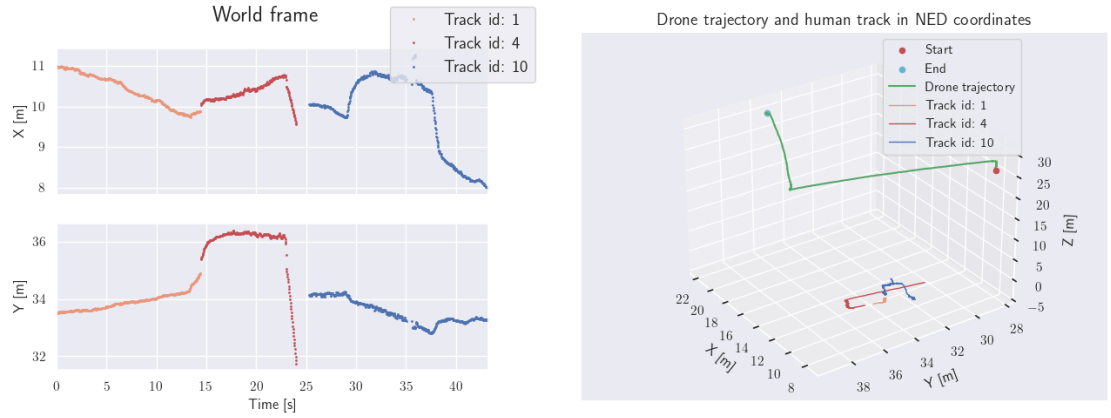
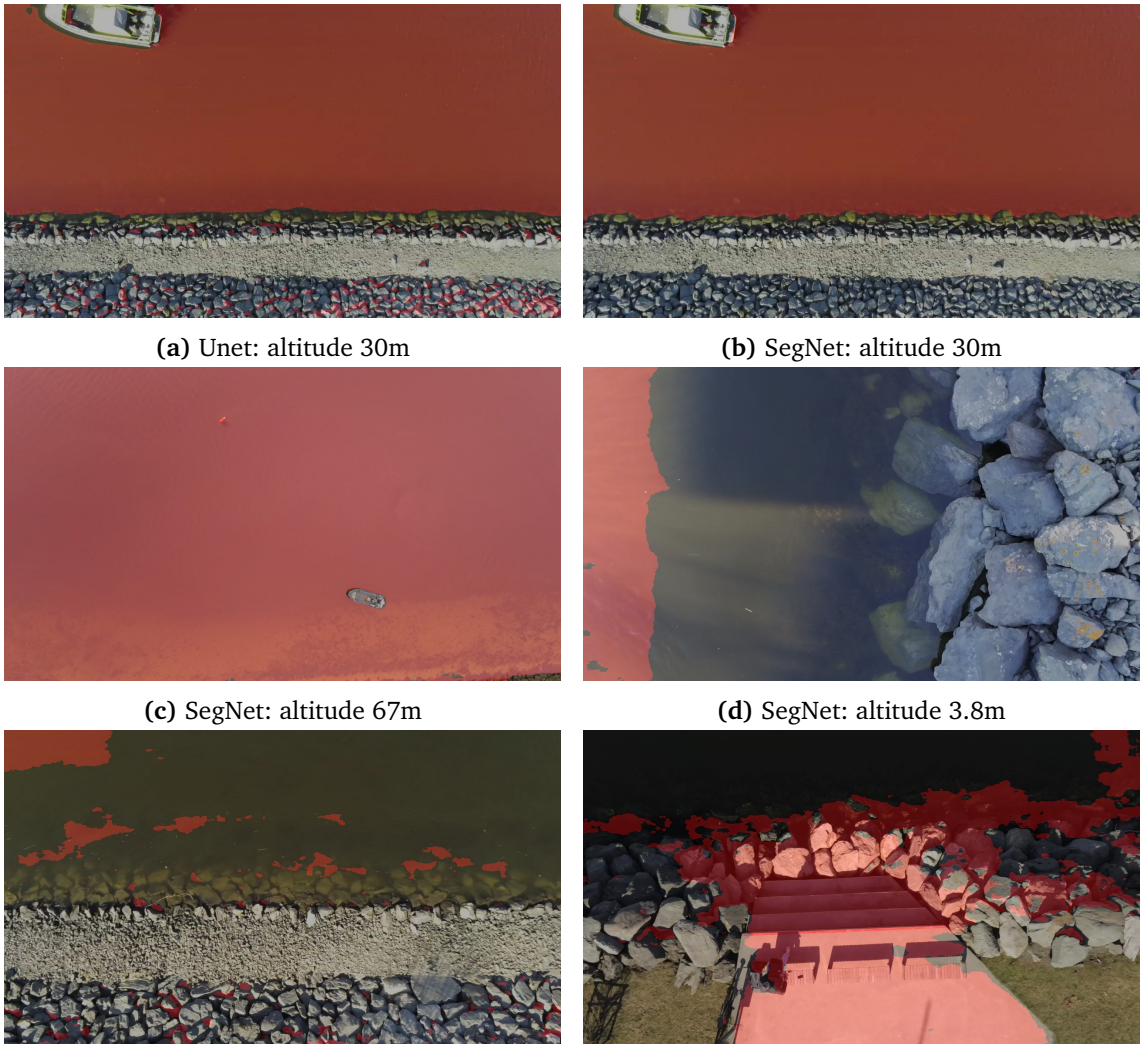



**Figure 5.19:** Human tracking under excessive drone and camera movement

## 5.7 Safe point generator subsystem

In order to evaluate the performance of the safe point generator the drone is flown at varying altitudes and orientations at Nyhavna and Hurtigbåtterminalen in the Trondheim harbor area.

**Deep Learning segmentation network predictions on aerial drone images**



**(a)** Unet: altitude 30m

**(b)** SegNet: altitude 30m

**(c)** SegNet: altitude 67m

**(d)** SegNet: altitude 3.8m

**(e)** SegNet: altitude 15.8m

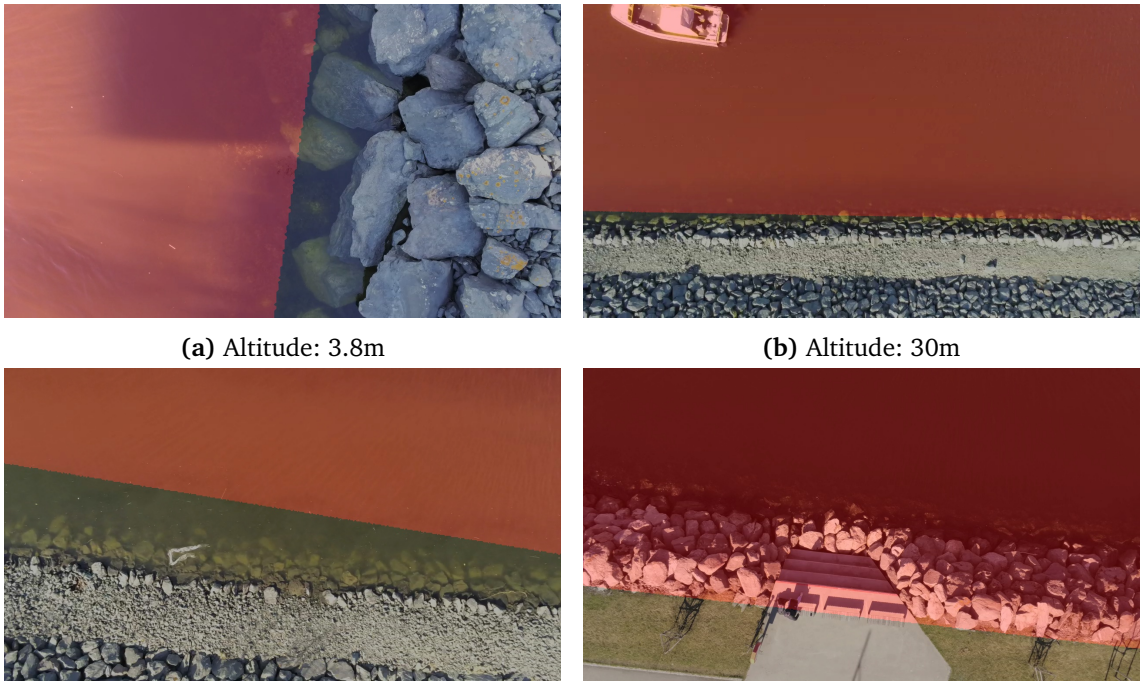
**(f)** SegNet: altitude 14.8m

**Figure 5.20:** Deep learning segmentation masks from images from the Anafi drone. The image mask is marked in transparent red.

The Unet and SegNet models with the best test-mAP (model 4 and 2 respectively) from Section 5.2 are tested on the same image captured at an altitude of 30m. The results can be seen in Figure 5.20a and Figure 5.20b respectively. The water mask is visualized as a transparent red color in the images. Both models manage to mask out the water and leave the boat as non-water, however, the Unet model seems to also mask a lot of shadows caused by the rocks on land as water. As a result, the Unet model was discarded because this result was indicative of multiple detection altitudes. In Figure 5.20c the SegNet model is used to generate masks at 67 meters. The network manages to mask out to the boat and shoreline correctly. Figure 5.20d shows the SegNet model's detection performance at an altitude of 3.8 meters. The model masks some of the water correctly, but not all water is masked out. In Figure 5.20e the drone is flown at an altitude of 15.8 meters and the SegNet model has a hard time masking out the water. Only patches of water are masked out, and the model also wrongly masks out shadows

caused by the rocks on land as water. Lastly, Figure 5.20f shows the SegNet model results when the drone is flown at an altitude of 14.8 meters over a more feature-rich land patch with a uniform-colored concrete area. The model wrongly masks out the gray concrete area as water and misses close to all water in the image.

### Offline map segmentation predictions on aerial drone images



**(a)** Altitude: 3.8m

**(b)** Altitude: 30m

**(c)** Altitude: 15.8m

**(d)** Altitude: 14.8m

**Figure 5.21:** Offline map segmentation output from images from the Anafi drone. The image mask is marked in transparent red

In figures 5.21a and 5.21b the results from the offline map mask are shown for altitudes of 3.8 meters and 30 meters respectively. The mask correctly masks out the water, however, the boat in the image from 30 meters is consequently masked as water. In Figure 5.21c the drone is flown on a day with more cloudy weather conditions and hence with degraded GNSS signal. The consequence is a segmentation mask with non-optimal quality as the offline mask generation ultimately depends on the accuracy of the drone's orientation and GNSS measurements. As the offline map segmentation mask is based on an offline map source, the quality of the mask also directly depends on the correctness of the offline map. In Figure 5.21d the drone is flown over an area where the offline map incorrectly maps the stones as water. This error will consequently be passed on to the segmentation mask.

**Safe point generator**

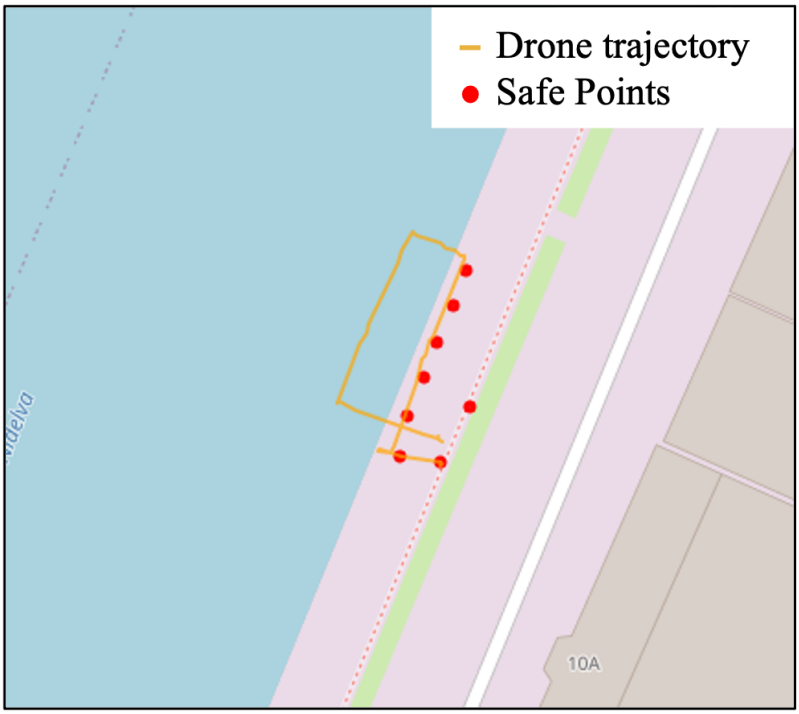



**Figure 5.22:** Map with generated safe points and drone trajectory

Figure 5.22 shows the drone trajectory and the generated safe points from a flight in Nyhavna in Trondheim. Both the radius of acceptance of publishing new safe points and the requirement of sidelengths in the safe landing area are set to 5 meters. The striding in the windowed search is set to 20. Furthermore, the generator is set to use the deep learning-based segmentation mask if the two generated masks match with at least 80%. The drone starts on the ground, flies to an altitude of 15.8 meters, and follows the trajectory in the figure. When the drone flies back to the landing point it flies over water and no safe points should be generated. All safe points can be seen to be generated on land, with a sufficient distance to water. Even though the altitude is sub-optimal for deep learning-based segmentation, the deep learning-based mask was sufficiently equal to the offline map-based mask 47.12% of the time and hence used to find safe points.

## 5.8 Full perception system

Figure 5.23 shows the result of a mission where the drone is slowly flown over a human in the water at an altitude of 40m. All red and green points in the figure show information sent out of the perception system. The threshold for what constitutes a valid human detection is set to 50%, and the radius of acceptance of both new safe points and human detections is set to 5m. The requirement for a safe landing point is set to an area of 5 x 5 meters. The system correctly manages to only send one message about human detection, and all safe points are generated at reasonable locations on land with sufficient far distance to the water. Furthermore, the severity level of the human detection is correctly set to 2, the highest level, as there are no floating objects in close proximity to the human.

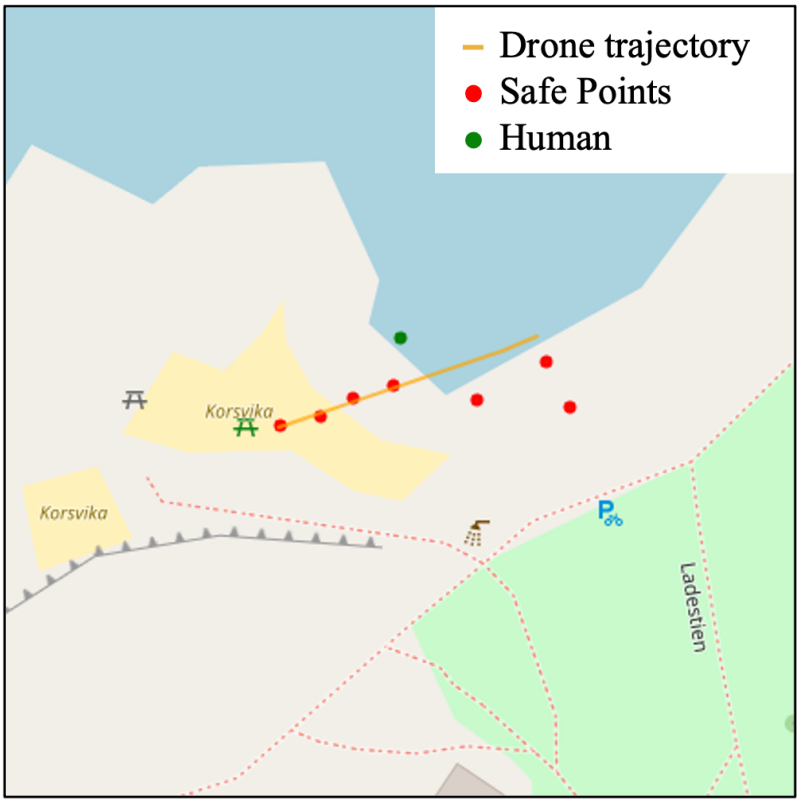
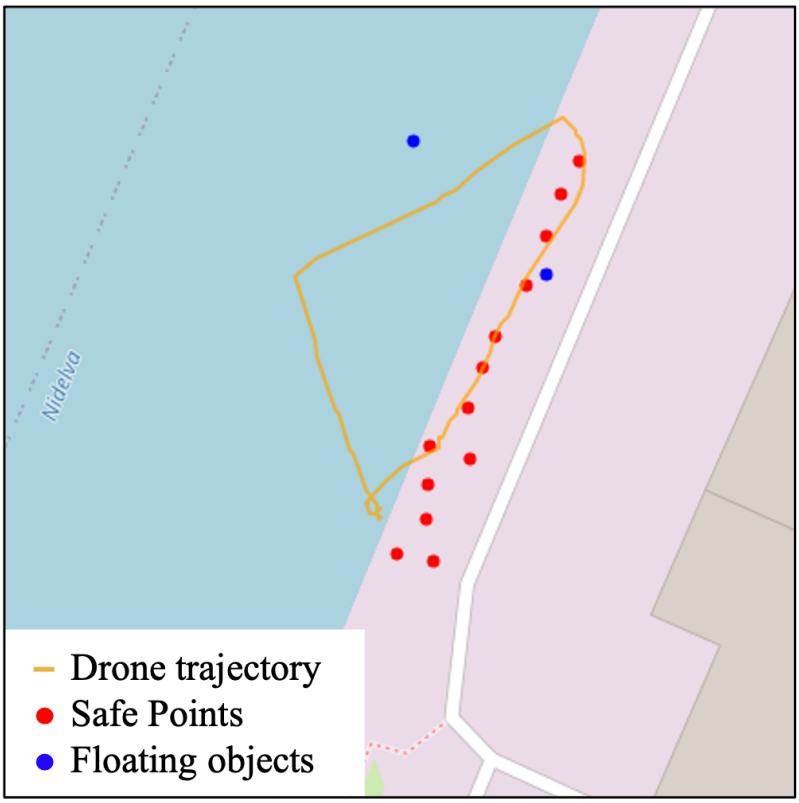
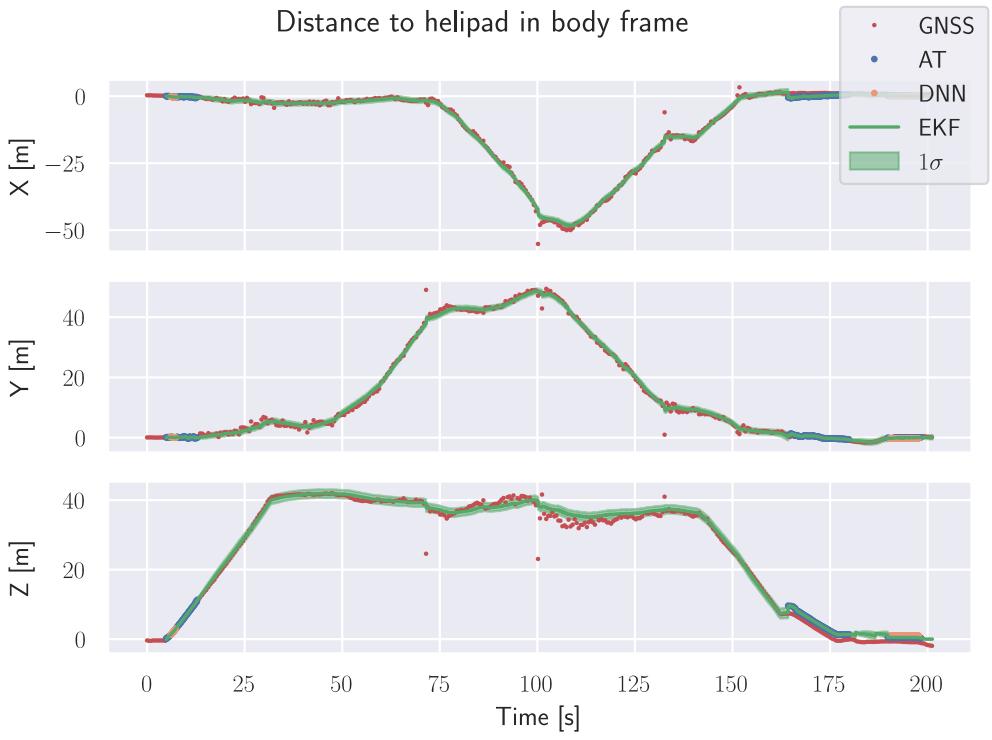



**Figure 5.23:** Mission at Korsvika. Map with generated safe points, human detection, and drone trajectory

Figure 5.24 shows the output from a full mission with all subsystems running at Nyhavna. In the mission, the helipad is mounted on the ReVolt, where the mission starts and ends. Due to extremely low tide on the day of testing with the ReVolt, the minimum requirement for a safe landing point is set to an area of 10 x 10 meters. The threshold for what constitutes a valid floating object detection is set to 80%, and the radius of acceptance of both new safe points and detections is set to 5 meters. On the map in Figure 5.24a two floating objects are marked, whereas only one is correctly a boat in the water. The other floating object is a car parked on land. All safe points are generated on land, and published accordingly. Figure 5.24b shows the helipad tracking during the complete mission.




**(a)** Map with generated safe points, Floating object detections, and drone trajectory




**(b)** Helipad tracking for the extent of the mission

**Figure 5.24:** Mission at Nyhavna. Helipad mounted on ReVolt.

Figure 5.25 shows the helipad tracking during the landing phase for the Nyhavna mission. The camera-based estimators are toggled on correctly when the altitude is below 10 meters. The X and Z EKF estimates can be seen to have small step-responses when the camera-based estimators are toggled on and can produce high-accuracy correction measurements. The step responses can, however, be seen to be very close to being within one standard deviation. As the drone is manually controlled in the mission, the operator did not manage to control the drone in a straight path such that the helipad was visible in the camera during the complete landing phase. Between 180 and 190 seconds in the figure, the GNSS feedback is again toggled on as corrections due to the helipad not being visible in the camera.

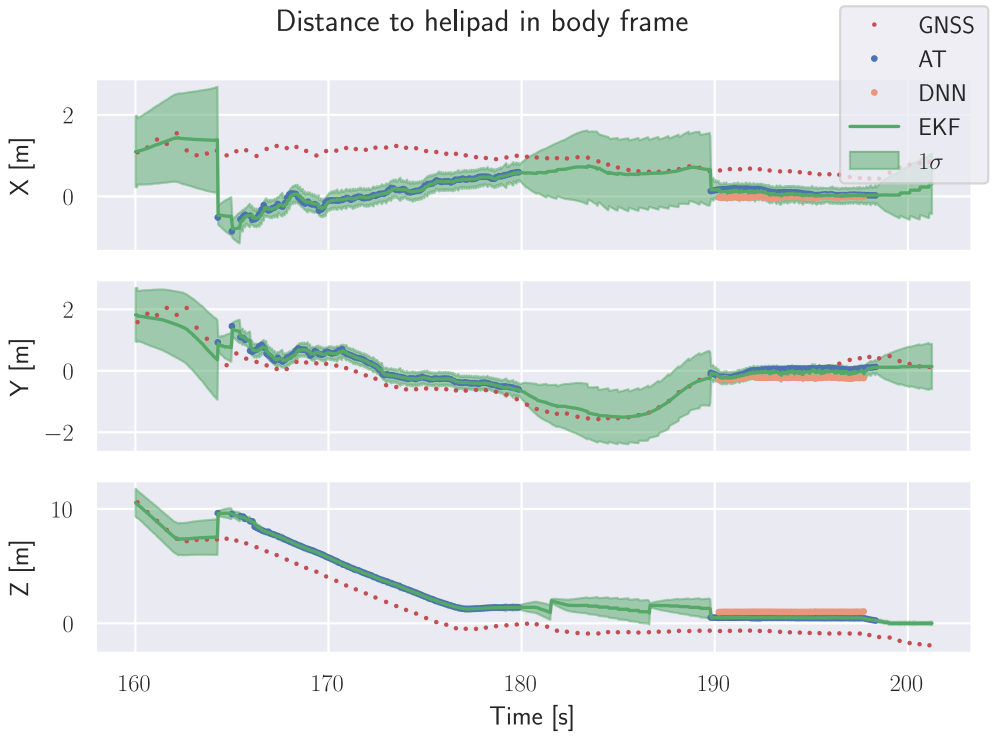
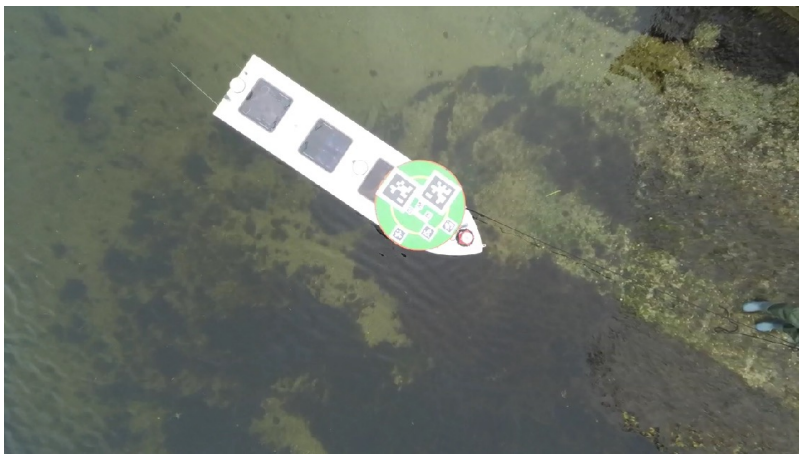



**Figure 5.25:** Mission at Nyhavna. Helipad tracking during the landing phase

During the landing in Figure 5.25 only the AprilTag-based estimator can be seen to produce corrective measurements in the filter between 165 seconds and 190 seconds. The DNN-CV-based estimator is not able to produce any estimates. The reason is that the camera is overexposed during the landing phase, making the helipad appear whiter than it really is. The helipad mounted on the ReVolt can be seen during the takeoff phase in Figure 5.26a and during the landing phase in Figure 5.26b. It is evident that the camera is overexposed to light during the landing phase, something that makes the YOLOv8 network used to detect the helipad unable to detect the helipad.

**(a)** Takeoff



**(b)** Landing

**Figure 5.26:** Camera overexposure. Takeoff vs landing.

# Chapter 6

# Discussion

This section begins by discussing the results of testing the various submodules developed for this thesis, followed by an assessment of the overall system's performance.

## 6.1 Helipad tracker subsystem

As the functionality of this module is essentially unchanged from the project thesis [4], this section will only shortly discuss what the changes to the system in this work have contributed to the subsystem's performance.

### 6.1.1 AprilTag estimator

The new AprilTag design on the helipad over doubles the detection range from 6 meters in the project thesis [4], to 14 meters. The addition of the new two large AprilTags with size 28 x 28 cm is responsible for the detection in the range of 6 to 14 meters. In addition, as the number of AprilTags is increased from 5 to 8, the system has more redundancy, especially in the lower altitude range. The estimator's accuracy could be improved further by reprinting the helipad overlay with AprilTags included in the overlay. This way the Apriltags' positions on the helipad could be measured with an even higher accuracy.

### 6.1.2 DNN-CV estimator

The new YOLOv8 network used for helipad detection increased the detection altitude of the helipad from 5 meters in the project thesis [4], to surpassing 8 meters. Furthermore, the offsets in the estimates from the DNN-CV module described by Falang in [3] and in the project thesis [4] are removed with the new YOLOv8 network used to detect the helipad. The labeling procedure used for the old data was thought to be the cause of the offsets' presence. And as a result, the old network predicted too large bounding boxes, giving the impression that the helipad was closer than it actually was. The new dataset contains images captured from the Anafi drone with a resolution of 1280 x 720, and the ground truth bounding boxes in the dataset are manually hand-annotated with high precision. Hence, the network predicts bounding boxes that encapsulate the helipad with higher accuracy.

During the testing with the helipad mounted on the ReVolt, the DNN-CV-based estimator had problems detecting the helipad in the landing phase. The reason was due to the helipad and ReVolt being overexposed. The Anafi drone automatically adjusts the white balance of its

camera, which creates problems when the drone transitions from flying over the dark sea to descending straight towards the helipad. When the drone is flown over the sea, the camera's white balance is increased to better capture dark details. However, when the drone is then flown back again to the helipad, the increased white balance leads to the ReVolt and helipad being overexposed and the network is unable to detect the helipad. A possible solution to address this problem is to try to disable Anafi's automatically adjusted white balance and keep it at a fixed value. This approach would ensure consistent helipad visibility during both takeoff and landing phases.

The new YOLOv8 architecture is also more lightweight than the old YOLOv4 architecture. Whereas the v4 required over 4GB VRAM when running, the v8 requires only 1.1GB with the nano models used in this thesis.

### 6.1.3 Extended Kalman filter

The new toggle logic, controlled by the perception master, works well. It heuristic manages to estimate when the helipad is in view in the camera, and the camera-based estimators are able to predict the helipad position. On the other hand, when the helipad isn't visible in the camera, the GNSS feedback makes the system more resilient to both drifting and noise. Furthermore, when the camera-based estimators provide the accurate position measurement, then less accurate GNSS measurements will not influence the estimate.

The Apriltag-based estimator provides the most accurate and reliable correction measurements in the filter. This was also a result of the project thesis in [4], however, the DNN-CV-based estimators performance is increased in this work. Even though the DNN-CV-based estimator's performance is inferior to the Apriltag-based estimator, it provides corrective measurements to the filter with high accuracy when the helipad is detected by the network. Redundant correction measurements in the EKF can be beneficial for several reasons, mentioning robustness, accuracy, and increased reliability. The EKF can reduce the impact of individual measurement errors or outliers and it may also combine the information from both estimators to produce a more accurate and precise estimation of the drone's position in relation to the helipad.

A small step-response was observed when the camera-based estimators were turned on during the landing phase after a mission. However, this step response was very close to the expected value based on the estimate, considering the level of uncertainty. When the camera-based estimators are unavailable, the filter relies on GNSS measurements as correction measurements in the EKF. This is done to avoid relying solely on the underlying CV model and dead reckoning. However, using GNSS measurements introduces higher uncertainty into the estimation process because GNSS measurements are less accurate compared to the camera-based estimators. The observed behavior, with the fast changes in the estimated position, can be explained by the increased uncertainty introduced by the GNSS measurements. The step-responses are a result of the underlying variance associated with these corrective GNSS measurements.

Due to the underlying CV model and the GNSS feedback, the EKF can only estimate the position of a location-fixed helipad. The EKF are able to handle different helipad rotations. The filter could be made more comprehensive by keeping track of the helipad's NED world position and by including some way of correcting these estimates from outside sources.

## 6.2 Search subsystem

### 6.2.1 Search YOLO network

The search YOLOv8 network has been shown to provide correct detections at altitudes larger than 20 meters. The detections were with the highest confidence in the altitude range of 35 to 45 meters. Furthermore, the YOLO network provides detections in real-time at 30 Hz. There were, however, some limiting factors to the total performance of the detector, the main ones being

- Wrongly classifies humans as floating objects at altitudes between 7 and 20 meters.
- There are very few, if any, detections at altitudes below 7 meters.
- False detections of humans and floating objects, especially on land.
- Different light conditions influences detection confidence.

The main underlying reason for the network's predicative abilities in the lower altitude ranges was due to biases in the training data. The SeaDroneSea dataset, which makes up the main part of the dataset created for the search YOLO network, has the most images from an altitude of 40 meters. The AFO dataset, which is the other dataset used in YOLO training does not have metadata of image capture altitude, however, the altitudes in this dataset are also large. The network is therefore biased toward an altitude of 40 meters which corresponds to the altitude where the network seems to be performing the best. The wrongly classified humans between 7 and 20 meters may be a result of how large the humans appear in the images. As the network is biased towards larger altitudes, where humans usually are small in the images, the network may be confused at lower altitudes where the human appears larger, and hence more likely to classify the detection as a floating object. Moreover, since the human in the tests wears an immersion suit, the human may appear bulkier than the humans in the training data, particularly for the lower altitude ranges.

The lack of training data also contributed to the network's limited or nonexistent detections in the lower ranges below 7 meters. The image captured from the lowest altitude in the training data were captured from 8 meters above the sea. Furthermore, there are very few images of the coastline in the dataset used for training. Therefore, when the network is used on images where land is visible, it makes false detections as the environment is to some degree unknown. During testing, these false detections were for the most floating object detections, with occasional human detections. This is not a very large problem in the use case of this system, where the search is meant to be performed in the open sea. The search YOLO network is toggled on and off with a service call and can consequently be turned off before the drone has arrived at an area of interest. Furthermore, as the network predictions are further used in the SORT tracker, a small number of false detections, or multiple detections with low detection confidence, do not influence the detection pipe as the SORT tracker requires at least 30 detections and association to the same track and a minimal detection confidence before a track is confirmed and published.

Finally, variations in light conditions make the detection confidence variate. Objects in the scene may be overexposed or underexposed, making it harder to detect the objects in the scene. This will ultimately make the confidence of a detection highly variate, and dependent on factors such as detection altitude and light conditions.

In order to reduce the problems mentioned above while at the same time using a normal

RGB camera, the YOLOv8 network should be retrained. Ideally, the dataset should contain images from a large specter of altitudes, where no altitudes are more present than others. In addition, it would be beneficial to include more images in the dataset from a coastline environment in order to be able to use the system closer to the shore. However, research should be done on whether this compromises the detection capabilities of the network at open sea. The dataset and images could also be preprocessed before being used in the YOLO network in order to increase detection performance. An example of a preprocessing technique usable is, for example, the edge detection algorithm presented in [74].

Another option to increase the detection performance of the system is to use a thermal camera instead of the RGB camera. The thermal camera may be used in both day and night scenes, making it a better option for SAR missions at sea. Furthermore, the images from a thermal camera may either be processed by for example using edge detection as in [75] or they may be used directly in a YOLO network as in [76] to detect objects in the scene.

### 6.2.2 Multi-object trackers

The SORT tracker generally worked well and provided good tracking of objects when the detections from the YOLO network were of sufficient quality. Nevertheless, some comments on the performance of the SORT tracker and DeepSORT tracker are needed.

The SORT tracker performs data association by measuring IOU overlap between predicted bounding boxes and measured bounding boxes. Hence, its performance may be degraded in very cluttered scenes. It can be argued that this is not a large problem in the use case of SAR missions at sea due to the vast search area. In this thesis, it is not tested on a real-world cluttered scene with multiple objects to track. However, the SORT tracker works well on idealized detection data with multiple objects to track. It is based on this, assumable that the SORT tracker will be able to track multiple objects from real-world data as long as the bounding box detections are of sufficient quality.

As the SORT tracker is using a constant velocity KF to model tracks without any feature information, it has no way of recovering when a track drifts off. This was the clearest for very fast drone yaw movements, where the CV model no longer is sufficient to describe the bounding box motions in the image. The SORT tracker also lost tracks when the drone performed excessive motions that the gimbal was not able to keep up with. This issue can be avoided by controlling the drone via a closed loop rather than an open loop with manual control. Closed loop control can operate the drone with smoother movements than a human operator, thus eliminating the problem.

The DeepSORT tracker was only tested on idealized detection data with no false detection nor any miss detections and with only two objects in the scene. The tracker was not tested further due to it not being able to guarantee real-time performance on this data. It was tested with the intent of enabling re-identifications of detections and improving performance in cluttered scenes. However, the runtime of the feature embedder made real-time tracking at 30Hz impossible. A possible solution for DeepSORT is to downsample the detections and this way compensate for the possible slow feature embedder speed. However, as there are no guarantees on the max time of the embedder, especially in cases with very many detections in one image, this may be an infeasible approach. It is also beneficial for the performance of the total system

to not load the already heavily used GPU any more than necessary. A better solution may instead be to make a deeper integration between the YOLO network and the DeepSORT tracker by extracting the feature vector from the YOLO network for each bounding box directly from the feature embedder in the YOLO network. This way, the tracker is given access to feature vectors of objects in the scene without ever needing to use a separate feature embedder. After completing the experiments, it has been noted that a newer version of the YOLOv8 framework maintained by Ultralytics has been updated with a tracker module. The new tracking module supports Bot-SORT [77] and ByteTrack [78]. This tracker module could improve tracking, however, as the Ultralytics YOLOv8 framework is a pip-installed package, the flexibility and adaptability of the module may be limited. Furthermore, the two available trackers do not, at the time of completion of this thesis, have any re-identification properties.

As a final note on tracking, it may also be possible to transform all detections into world coordinates with class and confidence meta-data and do Multi Object Tracking in NED world coordinates by for example the Joint Probabilistic Data Association Filter (JPDAF) [40].

## 6.3 Safe point generator subsystem

### 6.3.1 Deep learning-based segmentation

The deep learning-based segmentation module has been shown to perform at a limiting degree for real-world experiments as it manages to segment out water for larger altitudes. However, the main problems with the module were

- Erroneous segmentation masks for low altitudes.
- Classifies shadows caused by land objects as water.
- Classifies uniform colored land patches as water.

It has been demonstrated that the deep learning-based segmentation does not produce trustworthy masks at low altitudes. The reason is that the segmentation models are trained on satellite images with a resolution of 8-10 meters per pixel. At 30 meters, the horizontal ground coverage of the drone's camera is approximately 40 meters given the camera's FOV. With the downsampling of the image before inference to a horizontal resolution of 256, this gives approximately 0.15 meters per pixel. Hence, it is assumed that the limiting segmentation performance in the lower altitudes is due to the fact that the segmentation network is strongly biased towards detections at large altitudes. This claim is also supported by the fact that the network was able to produce good segmentation masks for large altitudes during testing.

The segmentation network also often predicts shadows caused by land objects such as rocks to be water. This was much more evident for the Unet structure than for the SegNet structure. This is not surprising as the Unet structure was developed for the segmentation of medical images and the skip connections contain the entire feature map. The segmentation mask is therefore more detailed by design. The segmentation of shadows as water occurred also the most for the lower altitudes and is therefore also believed to be an effect of biased training data.

The last observed problem with the network predicting uniform colored land patched as water is believed to be for the same reason as for the shadow problem mentioned above. As the networks are trained on images with low resolution per pixel the water will appear more uniform colored and the network is likely to classify uniform patches as water.

The best solution to the problems mentioned above is to retrain the SegNet segmentation network on a new segmentation dataset. Even though the dataset used to train the networks in this thesis is from an environment similar to what the drone will be in in the use case of a SAR operation at sea, the altitude and the resolution of the images in the dataset are not optimal and hence the networks are not able to generalize well in the lower range of altitudes. Although the performance in the lower altitude range is not ideal, it works well at higher altitudes, and the predictions provide useful information when there are boats, for example, in the sea.

### 6.3.2 Offline map-based segmentation

The offline map segmentation module has been shown to be working well. It manages to produce correct segmentation masks in a great altitude range. Nevertheless, some problems have to be addressed

- Produces sub-optimal segmentation masks in the case of erroneous map data.
- No real-time analysis of the drone environment.
- Ultimately dependent on the accuracy of the GNSS and yaw measurement.

Errors in the offline map data will be reflected in the segmentation mask generated. This can lead to two scenarios; true water is falsely classified as land or true land is falsely classified as water. The former, where there is a chance of marking water as a safe landing point, is the most crucial error of the two. Nevertheless, unless the offline map data contains errors of several meters this problem will be mitigated by the requirement of a minimum size of what is considered a safe landing point. The offline map data in this thesis is downloaded from OpenStreetMap, which is an open-source map. A higher-quality more regularly updated licensed map could be used. The same thing will be a problem in the case of extremely high or low tides, however, again as long as the minimum safe distance is large enough this problem will be reduced to a minimum.

Boats or other floating objects that might be a safe landing point for the drone in an emergency are missed because of the way the module is constructed using offline map data. Some of the real-time aspects of the system would therefore be lost if the safe point generator only consisted of the offline map segmentation module. Furthermore, the module is dependent on the accuracy of the GNSS and yaw measurement. Small errors in either of them will be reflected in the segmentation mask. A possible solution in order to increase the accuracy of the GNSS measurement is to use a drone that supports a real-time kinematic (RTK) positioning system. According to [35] the accuracy of a local area RTK with a single base station will be in the centimeter range of up to 20 km. This will be far sufficient for an autonomous drone given the expected flight time of about 25 minutes. Finally, the node requires that the camera's FOV is fairly precisely known in order to calculate the ground coverage of the camera.

### 6.3.3 Safe point generator

The safe point generator manages to produce safe points based on the two segmentation masks. The design of the total safe point generator subsystem works as intended, and combines both robust and real-time point generation. Nevertheless, some minor problems and considerations have to be discussed.

- May miss safe points in the mask due to the striding in the search.

- May discard good deep learning-based segmentation masks in favor of offline map segmentation masks.
- Does not consider the environment around the safe point, as long as it is non-water.

Due to the way, the search for safe points in the masks is performed by using striding, potential safe points may be missed. The striding is needed in the search to limit the time it takes to analyze the masks. In the testing, the striding in the windowed search is set to 20 pixels, and hence 2304 points in each image are checked for being a safe point. The windowed search uses in this case between 0.05 - 0.12 seconds. However, if the striding is decreased to 5 pixels, this would require 36 864 points to be checked in each image, and a runtime of 0.8 - 1.8 seconds. The striding of 20 pixels, is thus chosen as a compromise between speed and accuracy. Nevertheless, other options may exist to perform the search in the mask faster such as an outwards spiral search from the image center with early stopping.

A good deep learning-based mask might be overlooked in favor of a potentially worse offline map mask if they differ pixel-wise by more than 80%. A solution to this may be to use a more complex method to compare the two masks, however, the author did not find this to be a very large problem. This is because the masks from the offline map segmentation turned out to be of good quality. Thus, it will often be a good benchmark for whether the deep learning-based mask is good enough. This is especially true for high altitudes, where the deep learning mask has shown to be the best. One downside to comparing pixel-to-pixel is that potential large floating objects that the deep learning mask manages to mask out will potentially not be used if the pixel-to-pixel mask similarity is less than 80%. This will be much more applicable if the network is retrained and its performance is increased at lower altitudes. Other methods than the pixel-to-pixel comparison may in this case be explored to determine which mask to use.

The only requirement for a point to be considered a safe point is that it should not be located in the water. As a result, a safe point may actually be at an unsuitable landing location where the drone may not be able to safely land. The subsystem could therefore be expanded to do a more complex analysis of the surrounding environment. Furthermore, the safe points altitude is set to the same altitude as what the drone had when it detected the point. If, for example, the drone flies over the water at an altitude of 30 meters, and there is a jetty at the edge of the image which is only 20 meters below the drone, the system will mark the altitude of the jetty to be 30 meters below the drone. The system which controls the drone to one of these safe points will therefore need to first fly to the correct longitude and latitude, and then decrease the altitude until the drone is landed. This is comparable to how a "return to home" drone feature works.

## 6.4 Pixel to coordinate transformer

Due to the unavailability of ground truth data for evaluating the system's performance outdoors, this section primarily focuses on assessing its performance in a lab setting where ground truth data is available. Nevertheless, the transferability to outdoor flying will be discussed towards the end.

### 6.4.1 Image to camera coordinates

The transformation from image to camera coordinates was tested by the use of two different algorithms; one based on similar triangles and one based on FOV. They performed similarly and functioned well overall.

The two algorithms can be seen to be deviating the most from the ground truth when the point is far away from the image center. This is also where the two algorithms deviate the most. This can point to the fact that there are some unmodelled distortions in the camera closer to the image borders. For the FOV-based estimator, it points to the possibility that the field-of-view degrees are not distributed linearly throughout the camera's FOV. However, the deviation from the ground truth is in the centimeter range, and hence negligible for the use case of SAR missions at sea.

Both algorithms are dependent on the accuracy of the altitude measurement coming from the drone. As the altitude measurement is made available in the tf-tree from the transform publisher node the processing time of points will be compensated for. Hence, the drone altitude of when the point was actually detected will be the altitude used in the transformation. Furthermore, the altitude measurement is used as a proxy for the altitude of the point that should be transformed. Hence, if the point is very much higher in the scene than what the drone reports as its altitude the accuracy of the transformation will be affected by this.

The choice of the preferred image-to-camera coordinates algorithm depends on the quality and accuracy of the camera information available. If the camera matrix, used in the similar-triangle-based algorithm, is estimated with high accuracy this is the preferred algorithm to use. On the other hand, if the camera FOV is available with high accuracy and is linearly distributed throughout the cameras FOV this is the preferred algorithm to use.

### 6.4.2 Camera to world coordinates

The transformation from camera to world coordinates is also made available in the tf-tree by the transform publisher. Hence, in a similar way as for the altitude, the processing time of the algorithm will be compensated for in the transformation. Nevertheless, the accuracy of the camera to world transformation is dependent on the accuracy of the GNSS measurement and the yaw measurement of the drone. In order to increase the accuracy of the transformation, a previously mentioned RTK system could be used.

The full transformation pipe from image-to-world coordinates also provided reasonable results when tested outdoors. For the safe point generator, the node transformed all safe points to land points when visualized on a map. When transforming track coordinates from the search subsystem the node is able to provide reasonable transformations, both when the drone is moving or when it is stationary.

## 6.5 Full perception system

Despite the issues mentioned in the different subsystems, the total perception system worked as intended and the perception master is able to provide an eventual planning system with valuable information during a SAR mission at sea. The system is capable of tracking the drone's

position in relation to a helipad, and in the landing phase, it provides high-accuracy position estimates. Furthermore, it manages to locate safe landing points for the drone in case of emergencies. Finally, humans in the water are detected and tracked given a sufficiently large flight altitude. Nevertheless, some aspects of the system need some comments.

The inconsistency in the confidence thresholds from the search YOLO network detections makes it hard to choose the threshold in the perception master for what constitutes a valid detection of either a human or a floating object. The confidence levels for the tracks are calculated as an Exponential Weighted Moving Average to put the most emphasis on the latest associated track measurement. However, there may be beneficial to use another way of calculating the confidence levels for the tracks. The outside tests showed that the confidence levels were dependent on the detection altitude, and hence it may be possible to compensate for detection altitude in the confidence calculations of the tracks. This problem may also be mitigated by re-training the search YOLO network.

The severity level calculations of detections are only based on if there are floating objects present in close proximity to the detected humans. However, other factors are also important in the calculations of such severity levels. The detection system could be updated to also detect if the human is wearing an immersion suit / a life jacket. Furthermore, the distance to the shore could also potentially be a criterion.

As the SORT tracker does not have any re-identification properties, single human detections may be sent out of the system multiple times if the tracks are lost and re-initiated under new track ids. The perception master will only update the position of detections if the track_id is the same. Hence, if the human detection is re-initiated with a new track_id, and the human's position is further away from all other tracks than the threshold, the detection will be published as a new detection with a new track_id. The best solution to this problem is to use a tracker with re-identification properties such as the DeepSORT tracker. Another option could be to assume that a re-initiation has taken place if the position of a new track_id is very close to that of an old track_id. However, this will make tracking in cluttered scenes impossible.

Due to deep learning-based segmentation, safe points could be marked on boats in open water. The perception master, however, does not distinguish between fixed safe landing points on land and potentially dynamic safe landing points, such as those on boats. Hence, if the boat moves, the point is no longer safe, something that should be taken into consideration by the perception master. Checking whether the safe point is on land or possibly on a moving object in open water by comparing the safe point location with offline map data may be a solution. If the point is on land, it is always a safe place to land; however, if the point is on a moving object, it needs to have a window of time during which it is still valid.

**Towards SAR operations at sea**

In light of the underlying theme of SAR operations at sea, it is important that the drone is well suited for the task. The drone used has to handle harsh weather conditions such as wind and rain as SAR operations at sea often occur in these conditions. The Anafi drone used in this work is not suited. The Anafi drone is not waterproof, and the drone has problems with wind gusts of 11-12 m/s. In addition, to extend the drone's use case in SAR missions at sea, the drone could also be used to carry a payload in the form of a life jacket or lifebuoy. This is also

not possible with the Anafi drone. There do not exist many commercially available drones with such features, however, one possible choice is the *SplashDrone 4* available from *SwellPro*[1].

### The road to edge computing

As a last discussion point, the transferability of the system to edge computing on a drone will be discussed shortly. In this work, the drone is controlled in open-loop with manual control, and all data is processed by an external computer. However, for closed-loop control, the processing should beneficially be done locally on the drone. This is in order to reduce potential delay times in data transmission. The author believes that a split design, with some edge computing and some external computing, is the best architecture for this system. Everything relating to real-time data such as the detection and tracking of humans, floating objects, and the helipad should be done locally on the drone. However, since it is not as time-critical, the process of locating safe points should be handled externally and sent back to the drone. The choice of YOLO network architectures was specifically chosen with the low processing requirements in mind. Both networks are YOLOv8 nano network, and they both require about 1.1 GB VRAM each. Hence running these networks on an edge device is very much feasible. Furthermore, the two networks will never run on the GPU at the same time as one is used during takeoff and landing, and the other during the search. On the other hand, the deep learning segmentation network requires about 2.5 GB VRAM and should be processed on the external computer. The analysis of the segmentation masks also requires a significant amount of CPU power and should be carried out externally. The full split-design system could use a multi-machine ROS configuration with a roscore running on the drone.

[1] `https://store.swellpro.com/products/splash-drone-4?variant=40008611102800`

# Chapter 7

# Conclusion and Future work

This chapter will first summarize the findings and results from this thesis. On the basis of the findings in this thesis, the following section will outline potential future research areas.

## 7.1 Conclusion

This thesis has investigated a camera-based situational awareness perception system for autonomous drones with the end goal of providing a possible mission planner with valuable information. The experiments were conducted indoors, in a lab environment, and outdoors in three different locations in the Trondheim harbor area using a Parrot Anafi FPV drone.

A search subsystem was created in order to detect and track humans and floating objects at sea. The subsystem consisted of a trained YOLOv8 network and a MOT tracker. For the purpose of training the YOLOv8 search network, a dataset was created by combining two online available datasets. Furthermore, the YOLOv8 framework was wrapped into ROS. The network's detection performance was good for large altitudes, however, it suffered from false detection and misclassifications at lower altitudes due to biased training data. The network was suggested retrained with unbiased training data in order to further increase its detection performance for a larger altitude range. Two different MOT trackers, the SORT and DeepSORT, were wrapped into ROS with track logic and confidence calculation using EWMA. The SORT tracker provided good tracking when the detections were of sufficient quality. It had the hardest time tracking objects during excessive drone motions, where tracks could be lost and re-initiated under a new track_id. However, as long as the drone movements were smooth, tracking was upheld with consistent track_ids. The DeepSORT tracker was only tested on idealized detection data due to it not being able to operate in real time. I was suggested to make a deeper integration between the detector and the DeepSORT tracker to reduce the problem of track re-initiations.

A safe point generator subsystem was developed to generate safe landing points for the drone in case of an emergency. The safe point generator system was based on sea-land segmentation, and in order to achieve and combine both robust and real-time segmentation, the segmentation was based on information from two different segmentation methods. The first segmentation method was a deep-learning encoder-decoder structure, which segmented images from the drone in real time. Two different network architectures were tested; Unet and SegNet. The two networks were trained on a dataset that combines data from two different publicly available datasets. The networks managed to create good segmentation masks when the drone altitude was sufficiently large, and the SegNet network gave the best segmentation

mask of the two. For lower drone altitudes, the network performance was poorer due to bias in the training data towards higher altitudes. In order to increase the network performance in a wider altitude range, the network was suggested retrained on a new dataset with labeled drone-captured images. The other segmentation method was based on offline map data, drone GNSS positions, altitude measurements, and yaw measurements. This module worked well and provided segmentation masks with good accuracy. The module gave suboptimal segmentation masks in the case of erroneous map data and as a consequence of the offline map design, it was not able to mask out possible floating objects in the water such as boats. The segmentation masks based on the two segmentation methods were pixel-wise compared in the segmentation master module. The offline map segmentation masks were used as a benchmark for whether the deep learning masks seemed reasonable. The deep learning-based masks were used for further processing if they were good enough, and discarded in favor of the offline map-based masks otherwise. Safe landing points were found in the mask by locating the area closest to the drone of a given size where the drone could land. The search for such points was conducted by applying a sliding window approach with striding. The full safe point generator found safe landing points for the drone, however, the module was suggested expanded in order to do a more complex analysis of possible safe landing points.

In order to georeference points, by transforming detections of humans, floating objects, and safe landing points from image coordinates to world coordinates the pixel-to-coordinate module was developed and used. Two different image-to-camera coordinate algorithms were tested. One based on similar triangles proposed by Sundvoll in [1], and one new algorithm based on FOV proposed by the author. The performance of the two algorithms was comparably good when tested in the drone lab with access to ground truth data. All transformations in the system used the native ROS framework for transformations between frames. This way, the processing time of the algorithm was compensated for in the transformations. When tested outside, the pixel-to-coordinate module provided reasonable transformations given the accuracy of the used GNSS measurement, both with and without drone motion.

The helipad tracker system developed in the author's project thesis [4] was integrated into the work in this thesis. The helipad tracker system employed a CV EKF, with measurements from two different camera-based position estimators, GNSS data, velocity measurements, and in the landing phase an internal drone altitude estimate. The first of the two camera-based position estimators was an estimator based on the detection of fiducial markers, AprilTags [15], on the helipad. In this work, the helipad was attached with more and larger AprilTags to improve the detection performance at larger altitudes. The detection altitude of the Apriltags was increased, from 6 meters to 14 meters. It was suggested to reprint the helipad overlay with the AprilTags printed on it to further increase the estimator's performance. The AprilTag-based estimator was also expanded with toggle logic for turning the processing of the node on and off. The other camera-based position estimator, the DNN-CV-estimator, was based on bounding box detections from a YOLO network trained for detecting the helipad. A dataset consisting of 2014 manually hand-annotated images of the helipad was made. Furthermore, a lightweight YOLOv8 nano network was trained on the dataset for detecting the helipad. The detection altitude of the DNN-CV-based estimator was increased from 5 meters to 8 meters. The DNN-CV-based estimator had problems detecting the helipad, when overexposed in the image, during landing phases in outside tests where the helipad was mounted on the ReVolt. The camera's automatic white balance adjustment was suggested disabled in order to ensure consistent helipad visibility during both takeoff and landing phases. The Kalman Filter was ex-

tended with toggle logic for turning the GNSS and the two camera-based position correction measurements on and off in the filter. Based on a heuristic, the two camera-based position estimators were toggled on and used as corrections when they were likely to produce estimates. On the other hand, when it was likely that they are unable to produce estimates, the GNSS measurements were used as corrections in the filter to prevent filter drift and make the filter more resilient to noise. Overall, the helipad tracking was good, however, it was suggested to estimate the NED position of the helipad in the filter to allow for a moving helipad.

All information from the different subsystems created for the thesis was gathered, processed, and distributed from the perception system by the perception master node. The perception master correctly published new safe points when they were sufficiently far away from old safe points. Suggestions were made on how to classify safe points as either dynamic or static. Furthermore, the perception master was able to keep track of human and floating object detections given sufficient large drone altitude and published human detections with severity levels at adequate times. The perception master had issues with setting a fixed confidence threshold for when to publish human detections due to variate confidence caused by altitude and light conditions. Suggestions were made to enhance the system's capabilities in addressing varying confidence levels and advancing the accuracy of severity level calculations for human detections.

## 7.2 Future work

This section will briefly discuss possible areas of future research that the author has identified.

The areas of improvement are summarized in Table 7.1. For the helipad tracker subsystem, the helipad design should be reprinted as a new overlay with the AprilTags printed on it. The camera's automatic white balance adjustment is reccomended disabled in order to ensure consistent helipad visibility during both takeoff and landing phases. Furthermore, the EKF states could be expanded to also track the helipad's position in the NED world frame. This could potentially be a solution to allow for a moving helipad, although alternative approaches may exist. For the search subsystem, the YOLOv8 network is suggested retrained with unbiased altitude data. Image preprocessing could also be investigated further to increase detection performance. A deeper integration between the YOLO network and the DeepSORT tracker is proposed in order to allow for track re-identification. Another possibility is to look more into the integrated tracker module in the YOLOv8 framework. To improve the safe point generator, retraining the deep learning-based segmentation with drone-captured labeled images is advised. Additionally, expanding the safe point generation to perform a more comprehensive analysis beyond just differentiating between sea and land could be explored. Enhancing the overall complexity of the perception master is suggested, and specific suggestions can be found in table 7.1. The drone should be upgraded to a new more robust drone system, and the code should be updated accordingly. Considering that ROS1 Noetic reaches its end of life in May 2025, porting the code-base to ROS2 is suggested. Furthermore, if a potential new drone supports some edge computing, the idea of a split design should be tested out to improve the system's real-time performance. Finally, the perception system developed should be tested together with a mission planner in closed-loop control.

| | |
|---|---|
| **Helipad tracker** | |
| AprilTag estimator | • Reprint the helipad overlay with the AprilTags printed on it. |
| DNN-CV estimator | • Turn off automatic white balance adjustment in the camera. |
| EKF | • Expand the filter to accommodate a moving helipad. |
| **Search module** | |
| YOLO network | • Retrain the network using a dataset of images from a high-altitude specter.<br>• Investigate image preprocessing. |
| MOT | • Extract features from the YOLOv8 network and use these in the DeepSORT framework to allow for re-identification.<br>• Investigate the integrated tracker module in Ultralytics YOLOv8 framework. |
| **Safe point generator** | |
| DL segmentation | • Retrain the network using a dataset of images from a drone. |
| Segmentation master | • Expand the module to do a more complex analysis than just separating sea/land. |
| **Perception master** | |
| General | • Investigate heuristics for confidence level calculations and track confirmation.<br>• Expand severity level calculations of detections.<br>• Separate between static/dynamic safe points. |
| **Other** | |
| Search-and-rescue | • Change the drone to a more robust waterproof and windproof drone, ideally with both a thermal and RGB camera. |
| Sofware | • Port code-base to ROS2. ROS1 Noetic end of life May 2025.<br>• If the new drone supports edge computing, test out the split design. |
| Experiments | • Test the perception system together with a mission planner system in closed-loop. |

**Table 7.1:** Possible areas of future research

## Final Note

The author is excited about the fact that the project will be continued as a master's thesis under the guidance of Anastasios Lekkas from ITK at NTNU. The author envisions the developed system as a springboard for advancing the ultimate objective of employing autonomous drones in Search And Rescue missions at sea.

# Bibliography


[1] T. Sunvoll, "A camera-based perception system for autonomous quadcopter landing on a marine vessel," Master's thesis, Norwegian University of Science and Technology, O. S. Bragstads Plass 2D, 7034 Trondheim, 2020.

[2] P. B. Hove, "Perception and high-level control for autonomous drone missions," Master's thesis, Norwegian University of Science and Technology, O. S. Bragstads Plass 2D, 7034 Trondheim, 2021.

[3] M. Falang, "Autonomous uav landing on a boat - perception, control and mission planning," Master's thesis, Norwegian University of Science and Technology, O. S. Bragstads Plass 2D, 7034 Trondheim, 2022.

[4] S. S. Allum, "Camera-based perception system for the landing of autonomous multirotor," Project report in TTK4550, O.S. Bragstads Plass 2D, 7034 Trondheim, 2022.

[5] Ø. Solbø, "Guidance and control for landing of autonomous multirotor," Project report in TTK4550, O.S. Bragstads Plass 2D, 7034 Trondheim, 2022.

[6] G.-R. Shih, P.-H. Tsai, and C.-L. Lin, "A speed up approach for search and rescue," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 4178–4183. DOI: 10.1109/SMC.2018.00708.

[7] S. Aronica, F. Benvegna, M. Cossentino, S. Gaglio, A. Langiu, C. Lodato, S. Lopes, U. Maniscalco, and P. Sangiorgi, "An agent-based system for maritime search and rescue operations," in *CEUR Workshop Proceedings*, vol. 621, Sep. 2010.

[8] J. P. Queralta, J. Raitoharju, T. N. Gia, N. Passalis, and T. Westerlund, *Autosos: Towards multi-uav systems supporting maritime search and rescue with lightweight ai and edge computing*, 2020. arXiv: 2005.03409 [cs.RO].

[9] R. Glomseth, F. Gulbrandsen, and K. Fredriksen, "Ambulance helicopter contribution to search and rescue in north norway," *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, vol. 24, Dec. 2016. DOI: 10.1186/s13049-016-0302-8.

[10] S. P. Yeong, L. M. King, and S. S. Dol, "A review on marine search and rescue operations using unmanned aerial vehicles," in *International Journal of Marine and Environmental Sciences*, vol. 9, 2015, pp. 396–399.

[11] K. Jayalath and S. R. Munasinghe, "Drone-based autonomous human identification for search and rescue missions in real-time," in *2021 10th International Conference on Information and Automation for Sustainability (ICIAfS)*, 2021, pp. 518–523. DOI: 10.1109/ICIAfS52090.2021.9606048.

[12] J. Sharafaldeen, M. Rizk, D. Heller, A. Baghdadi, and J. .-. Diguet, "Marine object detection based on top-view scenes using deep learning on edge devices," in *2022 International Conference on Smart Systems and Power Management (IC2SPM)*, 2022, pp. 35–40. DOI: 10.1109/IC2SPM56638.2022.9988928.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "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.

[14] M. Marti, A. Barekatain, H.-F. Shih, S. Murray, Y. Matsuo, and H. Prendinger, "Situation awareness for uavs using deep learning techniques," *JSAI Technical Report, Type 2 SIG*, vol. 2017, no. AGI-005, p. 04, 2017. DOI: 10.11517/jsaisigtwo.2017.AGI-005_04.

[15] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[16] M. Rizk, F. Slim, and J. Charara, "Toward ai-assisted uav for human detection in search and rescue missions," in *2021 International Conference on Decision Aid Sciences and Application (DASA)*, 2021, pp. 781–786. DOI: 10.1109/DASA53625.2021.9682412.

[17] M. S. Sruthi, M. J. Poovathingal, V. N. Nandana, S. Lakshmi, M. Samshad, and V. S. Sudeesh, "Yolov5 based open-source uav for human detection during search and rescue (sar)," in *2021 International Conference on Advances in Computing and Communications (ICACC)*, 2021, pp. 1–6. DOI: 10.1109/ICACC-202152719.2021.9708269.

[18] E. Lygouras, N. Santavas, A. Taitzoglou, K. Tarchanidis, A. Mitropoulos, and A. Gasteratos, "Unsupervised human detection with an embedded vision system on a fully autonomous uav for search and rescue operations," *Sensors*, vol. 19, no. 16, 2019, ISSN: 1424-8220. DOI: 10.3390/s19163542. [Online]. Available: https://www.mdpi.com/1424-8220/19/16/3542.

[19] A. Al-Naji, A. G. Perera, S. L. Mohammed, and J. Chahl, "Life signs detector using a drone in disaster zones," *Remote Sensing*, vol. 11, no. 20, 2019, ISSN: 2072-4292. DOI: 10.3390/rs11202441. [Online]. Available: https://www.mdpi.com/2072-4292/11/20/2441.

[20] V. A. Feraru, R. E. Andersen, and E. Boukas, "Towards an autonomous uav-based system to assist search and rescue operations in man overboard incidents," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 57–64. DOI: 10.1109/SSRR50563.2020.9292632.

[21] R. Geraldes, A. Gonçalves, T. Lai, M. Villerabel, W. Deng, A. Salta, K. Nakayama, Y. Matsuo, and H. Prendinger, "Uav-based situational awareness system using deep learning," *IEEE Access*, vol. 7, pp. 122 583–122 594, 2019. DOI: 10.1109/ACCESS.2019.2938249.

[22] E. Vasilopoulos, G. Vosinakis, M. Krommyda, L. Karagiannidis, E. Ouzounoglou, and A. Amditis, "Autonomous object detection using a uav platform in the maritime environment," in *Research Challenges in Information Science*, R. Guizzardi, J. Ralyté, and X. Franch, Eds., Cham: Springer International Publishing, 2022, pp. 567–579, ISBN: 978-3-031-05760-1.

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

[24] S. Zhang, Q. Zhang, Y. Yang, X. Wei, P. Wang, B. Jiao, and Y. Zhang, "Person re-identification in aerial imagery," *IEEE Transactions on Multimedia*, vol. 23, pp. 281–291, 2021. DOI: 10.1109/tmm.2020.2977528. [Online]. Available: https://doi.org/10.1109%2Ftmm.2020.2977528.

[25] R. Jadhav, R. Patil, A. Diwan, S. M. Rathod, and M. Inamdar, "Aerial object detection and tracking using yolov4 and deepsort," in *2022 International Conference on Industry 4.0 Technology (I4Tech)*, 2022, pp. 1–6. DOI: 10.1109/I4Tech55392.2022.9952705.

[26] A. Sivakumar, M. Shahid, M. S, and A. Balaji, "Deeptracking from aerial platforms," in *2021 Asian Conference on Innovation in Technology (ASIANCON)*, 2021, pp. 1–6. DOI: 10.1109/ASIANCON51346.2021.9544679.

[27] A. Jadhav, P. Mukherjee, V. Kaushik, and B. Lall, *Aerial multi-object tracking by detection using deep association networks*, 2019. arXiv: 1909.01547 [cs.CV].

[28] S. Lenka, B. Vidyarthi, N. Sequeira, and U. Verma, "Texture aware unsupervised segmentation for assessment of flood severity in uav aerial images," in *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, 2022, pp. 7815–7818. DOI: 10.1109/IGARSS46834.2022.9883678.

[29] M. Zaffaroni, C. Rossi, *et al.*, "Water segmentation with deep learning models for flood detection and monitoring," *ISCRAM 2020 Conference Proceedings – 17th International Conference on Information Systems for Crisis Response and Management*, pp. 24–27, May 2020.

[30] T. Chowdhury and M. Rahnemoonfar, *Attention based semantic segmentation on uav dataset for natural disaster damage assessment*, 2021. DOI: 10.48550/ARXIV.2105.14540. [Online]. Available: https://arxiv.org/abs/2105.14540.

[31] Y. Zuo, J. Yang, Z. Zhu, R. Li, Y. Zhou, and Y. Zheng, "Real-time semantic segmentation of aerial videos based on bilateral segmentation network," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, 2021, pp. 2763–2766. DOI: 10.1109/IGARSS47720.2021.9554952.

[32] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, *Bisenet: Bilateral segmentation network for real-time semantic segmentation*, 2018. DOI: 10.48550/ARXIV.1808.00897. [Online]. Available: https://arxiv.org/abs/1808.00897.

[33] O. Tatale, N. Anekar, S. Phatak, and S. Sarkale, "Quadcopter: Design, construction and testing," *International Journal for Research in Engineering Application & Management*, vol. 4, no. AMET 2018, pp. 1–7, Mar. 2019, ISSN: 2454-9150.

[34] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, *Model predictive control for micro aerial vehicles: A survey*, 2020. DOI: 10.48550/ARXIV.2011.11104.

[35] P. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Artech House, 2013, 2nd ed., ISBN: 9781608070060.

[36] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. West Sussex, United Kingdom: John Wiley & Sons, 2021, 2nd ed., ISBN: 9781119991496.

[37] R. Szeliski, *Computer Vision - Algorithms and Applications, Second Edition* (Texts in Computer Science). Springer, 2022.

[38] D. A. Forsyth and J. Ponce, *Computer vision - a modern approach, Second edition*. Boston: Pearson, 2012, ISBN: 9780136085928.

[39] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2003, ISBN: 0521540518.

[40] E. Brekke, *Fundamentals of Sensor Fusion - Target tracking, navigation and SLAM, third edition*. 2022.

[41] N. Buduma and N. Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, 1st. O'Reilly Media, Inc., 2017, ISBN: 1491925612.

[42] T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1, ISBN: 9780070428072.

[43] N. O' Mahony, T. Murphy, K. Panduru, D. Riordan, and J. Walsh, "Adaptive process control and sensor fusion for process analytical technology," in *2016 27th Irish Signals and Systems Conference (ISSC)*, 2016, pp. 1–6. DOI: 10.1109/ISSC.2016.7528449.

[44] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Advances in Computer Vision*, K. Arai and S. Kapoor, Eds., Cham: Springer International Publishing, 2020, pp. 128–144, ISBN: 978-3-030-17795-9.

[45] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. DOI: 10.48550/ARXIV.1511.08458. [Online]. Available: https://arxiv.org/abs/1511.08458.

[46] G. Jocher, A. Chaurasia, and J. Qiu, *YOLO by Ultralytics*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[47] S. Patel, "Deep learning models for image segmentation," *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 149–154, 2021.

[48] J. Wei, *Sea-land segmentation dataset*, 2020. DOI: 10.21227/fgcf-cz74. [Online]. Available: https://dx.doi.org/10.21227/fgcf-cz74.

[49] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. DOI: 10.48550/ARXIV.1505.04597. [Online]. Available: https://arxiv.org/abs/1505.04597.

[50] V. Badrinarayanan, A. Kendall, and R. Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*, 2015. DOI: 10.48550/ARXIV.1511.00561. [Online]. Available: https://arxiv.org/abs/1511.00561.

[51] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artificial Intelligence*, vol. 293, p. 103 448, Apr. 2021. DOI: 10.1016/j.artint.2020.103448. [Online]. Available: https://doi.org/10.1016%2Fj.artint.2020.103448.

[52] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[53] Parrot, *Anafi white paper*, version v1.4, Accessed: 2023-02-20. [Online]. Available: https://www.parrot.com/assets/s3fs-public/2020-07/white-paper_anafi-v1.4-en.pdf.

[54] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*, 1st. O'Reilly Media, Inc., 2015, ISBN: 1449323898.

[55] L. A. Varga, B. Kiefer, M. Messmer, and A. Zell, "Seadronessee: A maritime benchmark for detecting humans in open water," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2260–2270.

[56] J. Gąsienica-Józkowy, M. Knapik, and B. Cyganek, "An ensemble deep learning method with optimized weights for drone-based water rescue and surveillance," *Integrated Computer-Aided Engineering*, pp. 1–15, Jan. 2021. DOI: 10.3233/ICA-210649.

[57] D. Cafarelli, L. Ciampi, L. Vadicamo, C. Gennaro, A. Berton, M. Paterni, C. Benvenuti, M. Passera, and F. Falchi, "Mobdrone: A drone video dataset for man overboard rescue," in *Image Analysis and Processing – ICIAP 2022*, Cham: Springer International Publishing, 2022, pp. 633–644.

[58] C. Seale, T. Redfern, and P. Chatfield, *Sentinel-2 water edges dataset (swed)*, Downloaded: 2023-03-03, 2021. [Online]. Available: https://openmldata.ukho.gov.uk/.

[59] *Sentinel-2 mission guide*, https://sentinel.esa.int/web/sentinel/missions/sentinel-2, Accessed: 2023-03-13.

[60] *Qualisys motion capture system*, https://www.qualisys.com/, Accessed: 2023-02-21.

[61] *Qualisys - cybernetics*, https://home.hvl.no/ansatte/gste/ftp/MarinLab_files/Manualer_utstyr/QTM-usermanual.pdf, Accessed: 2023-02-21.

[62] *Kartverket*, visited on 2023-05-02. [Online]. Available: https://kartverket.no/.

[63] *Civil aviation authority – norway - drones*, https://luftfartstilsynet.no/en/drones/open-category/, Accessed: 2023-05-23.

[64] Ø. Solbø, "Towards mission planning for search and rescue at sea," Master's thesis, O.S. Bragstads Plass 2D, 7034 Trondheim, 2023.

[65] G. Jocher, "Ultralytics/coco2yolo: Improvements," May 2019. DOI: 10.5281/zenodo.2738323.

[66] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, *On large-batch training for deep learning: Generalization gap and sharp minima*, 2016. DOI: 10.48550/ARXIV.1609.04836. [Online]. Available: https://arxiv.org/abs/1609.04836.

[67] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[68] D. Sarkar and R. Bali, *Transfer learning in action*, https://livebook.manning.com/book/transfer-learning-in-action/chapter-1/v-1/60, visited on 2023-05-23, 2021. [Online]. Available: https://livebook.manning.com/book/transfer-learning-in-action/chapter-1/v-1/60.

[69] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, *Learning transferable visual models from natural language supervision*, 2021. DOI: 10.48550/ARXIV.2103.00020. [Online]. Available: https://arxiv.org/abs/2103.00020.

[70] V. Rajput, *Robustness of different loss functions and their impact on networks learning capability*, 2021. DOI: 10.48550/ARXIV.2110.08322. [Online]. Available: https://arxiv.org/abs/2110.08322.

[71] D. Masters and C. Luschi, *Revisiting small batch training for deep neural networks*, 2018. DOI: 10.48550/ARXIV.1804.07612. [Online]. Available: https://arxiv.org/abs/1804.07612.

[72] B. Dwyer, J. Nelson, and J. Solawetz, *Roboflow*, version 1.0, 2022. [Online]. Available: https://roboflow.com.

[73] S. S. Allum, *Helipad dataset*, https://universe.roboflow.com/mscthesis/platform-uyw39, Open Source Dataset, visited on 2023-03-28, Mar. 2023. [Online]. Available: https://universe.roboflow.com/mscthesis/platform-uyw39.

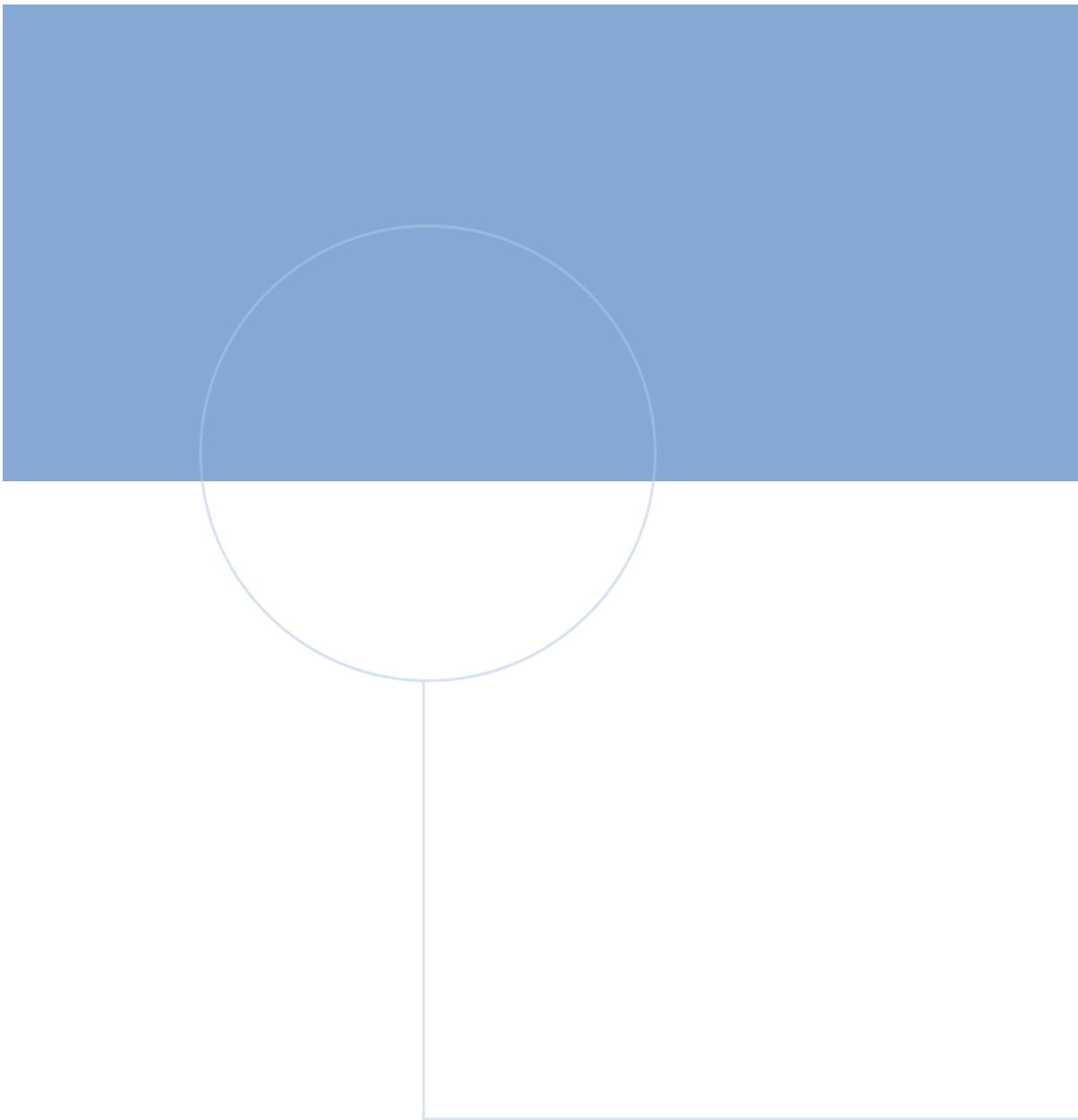
[74] C. D. Rodin and T. A. Johansen, "Detectability of objects at the sea surface in visible light and thermal camera images," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, 2018, pp. 1–10. DOI: 10.1109/OCEANSKOBE.2018.8559310.

[75] F. S. Leira, H. H. Helgesen, T. A. Johansen, and T. I. Fossen, "Object detection, recognition, and tracking from uavs using a thermal camera," *Journal of Field Robotics*, vol. 38, no. 2, pp. 242–267, 2021. DOI: https://doi.org/10.1002/rob.21985. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21985. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21985.

[76] C. Jiang, H. Ren, X. Ye, J. Zhu, H. Zeng, Y. Nan, M. Sun, X. Ren, and H. Huo, "Object detection from uav thermal infrared images and videos using yolo models," *International Journal of Applied Earth Observation and Geoinformation*, vol. 112, p. 102 912, 2022, ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2022.102912. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569843222001145.

[77] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, *Bot-sort: Robust associations multi-pedestrian tracking*, 2022. arXiv: 2206.14651 [cs.CV].

[78] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, *Bytetrack: Multi-object tracking by associating every detection box*, 2022. arXiv: 2110.06864 [cs.CV].



NTNU

Norwegian University of
Science and Technology