Camera-based perception system for the landing of autonomous multirotor

Simen Stensrød Allum

December 19, 2022

Abstract

Artificial intelligence (AI) has enormous potential and is already being used by humans in various ways every day. One such application is the use of AI-powered autonomous systems in search and rescue operations at sea. These systems, such as small autonomous unmanned aerial vehicles (UAVs), can cover large areas quickly and efficiently, potentially leading to the early detection of victims and reduced fatalities.

However, there are several challenges that arise when using autonomous UAVs in maritime operations, such as the ability of the vehicle to automatically land on a seaborne platform. This thesis aims to robustly estimate the location of a landing platform using the Parrot Anafi drone's onboard sensors in potentially harsh and challenging conditions. It uses the drone's camera as the primary sensor and combines data from the camera with GNSS data and other sensor data from the Anafi drone in a constant velocity Kalman filter. The thesis employs two different camera-based estimators, one based on deep learning and the other on more traditional computer vision techniques, in order to achieve a more robust estimation.

The thesis findings indicate that the full estimation system is reliable enough to track the position of the platform, even when it is not visible to the camera, in both indoor and outdoor environments on land. The system can handle large platform movements, but some minor adjustments to the platform's design may be necessary to make it even more resilient to changing lighting conditions. Overall, this thesis presents a promising approach to using AI-powered autonomous systems for search and rescue operations at sea.

Sammendrag

Kunstig intelligens (KI) har enormt potensiale og blir allerede brukt av mennesker på ulike måter hver dag. Et slikt anvendelse er bruk av AI-drevne autonome systemer i søk og redningsoperasjoner til sjøs. Disse systemene, som små autonome ubemannede luftfartøy (UAVer), kan dekke store områder raskt og effektivt, noe som kan føre til tidlig oppdagelse av ofre og reduserte dødstall.

Det er imidlertid flere utfordringer som oppstår når man bruker autonome UAVer i maritim operasjon, slik som evnen til kjøretøyet til å lande automatisk på en sjøbasert plattform. Denne oppgaven har som mål å robust estimere posisjonen til en landingsplattform ved hjelp av Parrot Anafi drone's onboard sensorer i potensielt krevende og utfordrende forhold. Det bruker drone's kamera som primærsensor og kombinerer data fra kameraet med GNSS-data og annen sensordata fra Anafi dronen i en konstant hastighet Kalman filter. Oppgaven benytter to ulike kamerabaserte estimatorer, en basert på dyp læring og den andre på mer tradisjonelle datasyn-teknikker, for å oppnå en mer robust estimering.

Oppgavens funn indikerer at hele estimeringssystemet er pålitelig nok til å spore plattformens posisjon, selv når den ikke er synlig for kameraet, både i innendørs og utendørs miljøer på land. Systemet kan håndtere store plattformbevegelser, men noen mindre justeringer i plattformens design kan være nødvendig for å gjøre den enda mer motstandsdyktig mot endringer i lysforholdene. Overordnet gir denne oppgaven en lovende tilnærming til bruk av AI-drevne autonome systemer for søk og redningsoperasjoner til sjøs.

Preface

This thesis was completed in the fall of 2022 at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway, as part of the Engineering Cybernetics, Specialization Project course TTK4550. The work done by Martin Falang [1], Peter Bull Hove [2], and Martin Sundvoll [3] in their master's theses is directly continued in the thesis. Section 1.2.1 will describe their contributions to the project.

My academic advisor this semester was Anastasios Lekkas from the Department of Engineering Cybernetics (ITK) at NTNU. He has provided me with ideas and feedback on the work completed for this thesis. He has also been accommodating in reading and offering feedback on my drafts for this thesis.

Paolo De Petris from ITK at NTNU assisted in establishing the fundamental operation of the drone lab at NTNU and its motion capture system.

Øystein Solbø [4] and I collaborated to write the ROS Olympe interface that is discussed in section 4.1.2.

Stefano Brevik Bertelli from ITK at NTNU provided me with access to a laminating machine and laminating sheets, which were used to laminate the AprilTags used in this thesis.

The department has provided a Komplett Khameleon P9 Pro laptop and two Parrot Anafi FPV drones, which are used in conjunction with Parrot's Olympe API 7.3 software and the Parrot-Sphinx 2.9.1 simulation software. The camera on the drones was calibrated using MATLAB r2020B.

In addition, the author made use of various open-source software and libraries in the development of this thesis, including Python 3.8.10, ROS Noetic Ninjemys, darknet_ros from Legged Robotics, OpenCV 4.2.0, BagPy 2.2.5, pandas 1.5.0, NumPy 1.23.3, apriltag 0.0.16, SciPy 1.9.1, seaborn 0.12.0, and matplotlib 3.6.0. All figures in the thesis were created by the author unless otherwise noted in the figure captions.

Contents

Figures

Tables

Acronyms

AI Artificial Intelligence. 1

ANN Artificial Neural Network. 15

API Application Programming Interface. 4, 5, 20, 21, 28, 33

AS Autonomous Systems. 1, 4

CNN Convolution Neural Network. 3, 4, 16

CV Constant Velocity. 2, 38, 40, 42, 43, 70

DLT Direct Linear Transform. 12, 13

DNN Deep Neural Network. 3, 15, 21, 65, 66

DNN-CV Deep Neural Network Computer Vision. x, xi, 2, 3, 30, 35, 37, 39, 42, 43, 48, 52, 56, 59, 65–67, 70, 71

DoF Degrees of Freedom. 3, 12

EKF Extended Kalman Filter. xi–xiii, 3, 5, 17, 18, 20, 37–41, 43, 54–59, 61, 63, 66–70

FPS Frames Per Second. 3, 19–21

GNSS Global Navigation Satellite Systems. 5, 39–43, 56, 59, 62, 67, 68, 70

GPU Graphics Processing Unit. 16, 21

KF Kalman Filter. 2, 3, 16, 17

LM Levenberg-Marquardt. 14, 15

NED North, East, Down. 5, 8, 29–31, 39–41, 70

PnP Perspective-n-Point. 12, 35

RMSE Root Mean Square Error. xiii, 50–52, 54, 65–67

ROS Robot Operating System. x, xiii, 4, 5, 21, 25, 27–29, 31–33, 35, 36, 40–43

SAR Search And Rescue. 1, 4, 8, 28, 71

SIFT Scale-Invariant-Feature-Transform. 11

SVD Singular Value Decomposition. 13

TCV Traditional Computer Vision. 2, 3, 33, 63, 65

UAV Unmanned Aerial Vehicles. ix, 1–4

UKF Uncented Kalman Filter. 17

YOLO You Only Look Once. ix, xi, 16, 21, 35, 36, 52, 53, 64, 66, 70, 71

Chapter 1

Introduction

1.1 Background and Motivation

The increase in the use of Artificial Intelligence (AI) has a large impact on society and leads to the increase in the use of autonomous systems [5]. With the rapid development of complex cybernetics systems, AI is something most people utilize every day, often without them being aware of it [6]. One of the reasons AI and Autonomous Systems (AS) are used is because they can help humans, and in some cases also outperform humans. An example of an AS that uses AI to outperform humans is a robotic manufacturing system. These systems use machine learning and computer vision technologies to automate tasks in a manufacturing environment, such as assembly, inspection, and packaging. In many cases, robotic manufacturing systems are able to work more quickly and accurately than human workers [7]. AI and AS can also be used to perform jobs that are too dangerous for us humans to perform, and computer systems also have the property of not getting exhausted, enabling them to work longer and faster than humans [6].

One example of an AS is Unmanned Aerial Vehicles (UAV). In the later years, there has been a large growth in both the research and use of these systems. This is a result of their low maintenance cost, high mobility and ability to hover [8]. Commercial available UAV systems are often also equipped with high-resolution cameras and hence sophisticated AI algorithms can therefore be used to analyze the video data from the cameras.

One specific area where AI and AS can guide and help humans is in Search And Rescue (SAR) missions. SAR operations need the use of people, boats, helicopters, and aircraft. Adverse sea effects and weather conditions frequently make SAR missions at sea, in particular, worse [9]. In SAR missions, time is a crucial factor to reduce the loss of human life. The authors in [8] point out that using autonomous UAV systems to speed up the search process can potentially save lives. Using UAV systems in such search missions have the benefit of being able to cover a large area in a short time, they are cheap in use compared with traditional aerial systems and they perform a search without putting operators at risk [8].

Whereas autonomous UAVs offer a lot of promise to help in SAR operations, there are also a number of difficulties. The system must be strong enough to allow the UAVs to fly autonomously and only transmit useful data to the rescue team in order for them to be an aid rather than a burden. Therefore, the system must be capable of handling every step of the search by itself, including challenging autonomous actions like landing on a moving target. There exists a lot of research on this particular field of autonomous landing UAVs [10].

When using a drone's camera for detecting purposes in a maritime environment, there are also a lot of issues that come up. When a drone makes a fast movement or is affected by a strong wind, vibrations in the drone may spread into the video stream, resulting in a hazy appearance [11]. Stormy weather will cause waves to partially or completely obscure floating objects, making it impossible for the camera to see them. Additionally, conditions with a calm sea and clear sky will cause water reflections, which will make it challenging to detect floating objects [12]. Finally, by merely employing a regular camera for detecting purposes, the search is limited to well-lit conditions.

The many challenges associated with employing camera vision for detection while deploying autonomous UAVs in marine situations will be examined in this thesis.

1.2 Previous work

1.2.1 Previous project development

This thesis builds upon the research on the same topic conducted in the individual theses of Martin Falang, Peter Bull Hove and Thomas Sundvoll [1–3].

Sundvoll [3] created a drone landing platform that was to be mounted on top of DNV's autonomous research ship ReVolt[1]. The platform looked like a typical helicopter landing platform, but it was built with distinct features that a perception algorithm could identify. In order to estimate the pose of the platform, Traditional Computer Vision (TCV) techniques such as color segmentation, edge detection, and corner detection were used. The perception-based pose estimation system showed potential in simulations, however, the performance was reduced in real experiments due to light conditions and noise in the images.

In order to create the perception system more robust, Hove [2] fused the pose estimates from the approach presented by Sundvoll in [3] with the pose estimates from a deep convolution neural network in a Constant Velocity (CV) Kalman Filter (KF). Even though the output from the algorithm using Deep Neural Network Computer Vision (DNN-CV) gave noisier estimates, the fused output from the KF was more reliable than the system consisting of only the TCV pose estimates.

Hove [2] had problems with communication delay between the physical drone and the control station, and hence Falang [1] upgraded the drone system from the previously used AR.Drone 2.0 to a more modern Anafi FPV both from the French

---

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

drone company Parrot[2]. Falang in [1] wrote a communication wrapper for the Anafi system as well as upgrading the TCV pose estimation algorithm to a new one based on feature detection, feature matching, and homography-based pose estimation. The new TCV pose estimation showed potential, however, it had offsets in all axes and was computationally heavy. The DNN-CV solution gave wrong estimates when the drone was too close to the platform, and Falang suggested that the DNN-CV only should be used for estimation purposes when the drone was sufficiently far away from the platform. The final pose estimation from the KF showed good results in simulations as well as in outdoor experiments when the platform was stationary. However, the estimations proved inaccurate when the platform moved or was introduced to a roll motion.

1.2.2 Perception for autonomous UAVs

There is a lot of research being done on estimating a UAV's pose using visual perception systems. Falanga et al. [13] present a method that is comparable to the one Sundvoll used in [3]. Here, the landing platform is made with distinctive features that make TCV techniques like binary thresholding and feature identification easily deployable. The authors provide an Extended Kalman Filter (EKF) to estimate the platform's location, orientation, and velocity in order to deal with time steps when the platform cannot be observed.

A similar approach is presented in [14] where the AprilTag system [15] is used as the features on the landing platform. Here, AprilTags of different sizes are spread throughout the landing surface. The AprilTags system was selected because it has the property of encoding a significant number of distinct IDs and has a low misdetection rate. Choi employs the same AprilTags method in [16] for detection. Using several AprilTags, where a smaller AprilTag is nested inside a larger AprilTag. An adaptive windowing scheme expedites the detection of AprilTags by creating a binary mask off of the most recent successful detection. The AprilTags have also been improved such that the detection of the AprilTags is unaffected by shadows obscuring half of the AprilTags.

A solution where a Deep Neural Network (DNN) is used to estimate the pose of objects is presented in [17]. Here, a lightweight Convolution Neural Network (CNN) is used to estimate the 4-DoF of gates from an uncalibrated mono-camera. The network is able to run at 10 Frames Per Second (FPS) on an Intel UP board. Another example of applying a DNN methodology to determining the pose of objects is seen in [18]. PoseCNN, a CNN trained to infer 6-DoF object pose estimation, is presented by Xiang et al. The rotation is estimated by regressing to a quaternion representation, and the translation vector is computed by localizing the object center in the image.

---

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

1.2.3 Autonomous UAVs in SAR-missions

Due to their agility, mobility, and aerial accessibility, UAVs have been employed for SAR missions for a number of years [19]. The danger of injury to the rescue team is decreased and the search may be carried out much more quickly and safely by employing autonomous UAVs.

Rudol and Doherty [20] present a method for detecting humans in color and thermal images from a commercially available drone. Geolocating detected human locations enables the creation of a map with various points of interest. Such a map could then be used to schedule the delivery of for instance medical supplies. In [21] the authors explain how they can find people in avalanches faster by using a drone fitted with an avalanche beacon. The drone searches in a grid pattern and then automatically lands in the discovered location of a possible trapped person and relays this location to the rescue crew.

Due to the often very large search areas in SAR missions, it is not easy to detect or find people in images as they are small compared to the size of the area. Persons can often also be partially covered with vegetation, and different light conditions may make it harder to detect persons in images. In [22] the use of deep Convolution Neural Network (CNN) detectors for detecting persons in images from drones are investigated.

1.3 Objectives

The main goal of this thesis is to develop the use of Autonomous Systems (AS), namely autonomous UAVs, in a comprehensive SAR operation at sea. Within this large scope, the goal of this thesis is to address the challenge of *accurately calculating the position of a landing platform*. When the platform is rotating or moving, the perception system ought to be able to determine its location. The system should also be able to know the relative position of the platform even when the platform is not in plain view of the camera.

1.4 Contributions

The contributions of this thesis are as follows:

- Co-authoring of a Robot Operating System (ROS) communication wrapper for the new Application Programming Interface (API) version used with the Parrot Anafi drone
- Refactoring and updating of the previously used code base to be compatible with newer ROS versions
- Use of a rotation compensating transformation between frames of reference utilizing the native ROS framework for transformations
- Use of an AprilTag detection-based pose estimation pipeline to produce fast and accurate pose estimates

- Integration of GNSS measurements in a North, East, Down (NED) frame as corrective measurements in a model-based EKF
- Integration of altitude corrective measurements during the landing phase in a model-based EKF

A communication wrapper for the new API version used with the Parrot Anafi drone was developed. This wrapper facilitated communication between the drone and the ROS framework, allowing for more easy control and monitoring of the drone's behavior. The previously used code base was also refactored and updated to be compatible with newer ROS versions. In order to accurately track the drone's position and orientation, a rotation compensating transformation between frames of reference was implemented using the native ROS framework for transformations. Additionally, an AprilTag detection-based pose estimation pipeline was developed to produce fast and accurate pose estimates for the drone. This was critical for ensuring its stable and reliable operation in a variety of scenarios. GNSS measurements in a NED frame were also integrated as corrective measurements in a model-based EKF, improving the accuracy of the pose estimates. Finally, altitude corrective measurements were integrated during the landing phase in a model-based EKF, allowing for safe and controlled landings. Overall, the project significantly improved the performance and reliability of the previously developed perception system in [1].

1.5 Outline

This thesis begins with Chapter 2, which presents the theory and background material for the work done in this thesis. The experimental setup utilized in this thesis is then presented in Chapter 3 before Chapter 4 covers the methods used to build the various subsystems in this thesis. Chapter 5 provides and describes the findings of testing the perception-based posture estimation system both individually and collectively. Chapter 6 examines these findings, and Chapter 7 closes the thesis with some final notes indicating prospective future work on the topic.

Chapter 2

Theory

The guiding theory behind this thesis is presented in this chapter. The chapter begins with a description of a quadcopter's operation and the many frames of reference used in this thesis. The chapter goes on to discuss the fundamental principle of using a camera as a sensor by giving a model of a camera. The topic of pose estimation is then examined utilizing both standard techniques such as feature extraction and processing, homography matrices, and pose extraction. Furthermore, detection in images by utilizing deep neural networks is presented, before the fundamental theory of the Kalman filter utilized in the context of this thesis to fuse the many sensor outputs together is presented.

2.1 Quadcopters

A quadcopter is a type of unmanned aerial vehicle (UAV) with four rotors, each powered by its own motor and propeller. As shown in Figure 2.1, the rotors of a quadcopter are mounted in an X-shape configuration. By applying voltage to the four DC motors, the speed of each rotor can be controlled. It is commonly assumed that the thrust generated by each propeller is proportional to the square of its rotation speed [23]. This means that the propellers are the dominant force acting on the quadcopter. The force generated by each propeller is denoted as F_i in Figure 2.1. In addition to generating force, the propellers also produce angular moments. To cancel out these moments and control the orientation of the quadcopter in the X-Y plane, the non-diagonal propellers rotate in opposite directions, while the diagonal propellers rotate in the same direction. This configuration allows for the cancellation of the generated angular moments. By adjusting the force generated by each of the four motors, the attitude of the quadcopter can be controlled. The quadcopter's attitude, in turn, can be used to control its position.

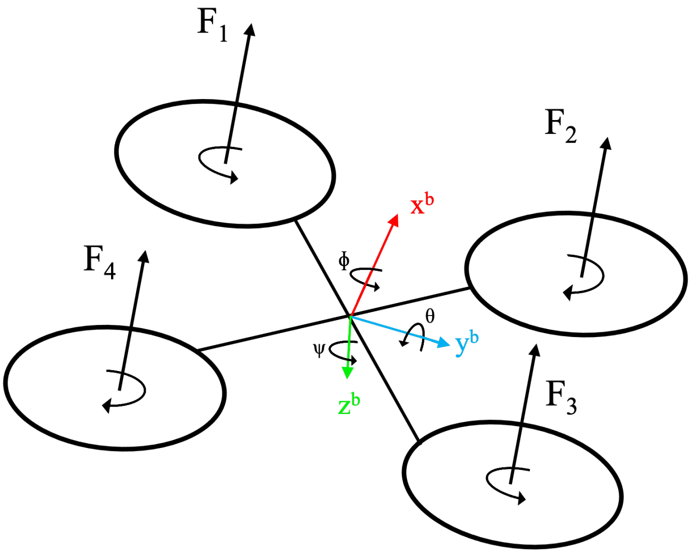


Figure 2.1: Model of the quadcopter. Inspired by [1]

2.1.1 Frames of reference

As this is a direct continuation of the work done in [1–3], all the frames of reference are kept unchanged. The first frame is a world frame that is stationary which is used to describe the quadcopter's position and attitude. This frame is chosen as a North, East, Down (NED) frame, denoted $\{n\}$ and is defined as a tangent plane on a point on Earth's surface. The curvature of the earth can be ignored in the frame for the sake of this thesis and the use case of the drone in SAR mission. The NED frame has the following axis definitions according to the definitions in [24]:

- x^{n} - Points towards true north.
- y^{n} - Points towards true east.
- z^{n} - Pointing downwards normal to Earth's surface.

The second frame of reference is a body-frame attached to the quadcopter as illustrated in figure 2.1, denoted $\{b\}$. This frame is used to describe both linear and angular velocities. The axis of this frame can be defined as in [24] to be:

- x^{b} - Longitude axis, pointing from the back to the front nose of the quadcopter.
- y^{b} - Transversal axis, pointing to the right from the longitude axis.
- z^{b} - Normal axis, directed downwards through the quadcopter.

In addition to the two frames used to describe the motion of the quadcopter, the camera mounted on the drone also has its own reference frame, denoted $\{c\}$. This frame is used to describe the objects seen in the images captured with the camera, and its origin is in the center of the image plane. The camera frame has multiple ways of being defined, and in this thesis, it is defined the same way as in [25] and has the following axis:

- x^c - Right in the image plane.
- y^c - Down in the image plane.
- z^c - Pointing straight out of the camera plane.

It becomes necessary to convert between different frames because some definitions are better expressed in particular frames. The translation vector between the two frames' origins, as well as the rotation between the two frames, are needed to convert between the frames of reference. To transform a vector **v** given in frame $\{1\}$ to frame $\{0\}$ the general expressions is a given in [24] as:

$$\mathbf{v}^0 = \mathbf{R}_1^0\mathbf{v}^1 + \mathbf{t}_1^0. \tag{2.1}$$

In 2.1 the rotation matrix $\mathbf{R}_1^0$ is the rotation from frame $\{1\}$ to frame $\{0\}$, and the translation vector $\mathbf{t}_1^0$ is the translation from frame frame $\{1\}$ to frame $\{0\}$ given in frame $\{0\}$.

2.2 Camera modeling

In order to use the camera as a sensor, a model of how the camera projects the 3D world coordinates into pixel coordinates on the image plane is necessary. One of the more commonly used camera models is the perspective model [25]. In the perspective camera model light passes through an infinite small hole and is projected inverted on the image plane [26]. The projective camera model is also called the pinhole model. This model is an approximation of the imaging process since a pinhole cannot be indefinitely small in reality. However, the model is straightforward and mathematically convenient and offers a good approximation. Figure 2.2 depicts how the projective camera model forms images, as well of some of its parameters.

In order to model the pinhole camera model mathematically the pinhole is defined to be the same as what is called the optical center C. Furthermore, the distance from the optical center to the image plane is defined to be the focal distance f. By introducing *homogeneous coordinates* $\tilde{\mathbf{u}} = \begin{bmatrix}\tilde{u} & \tilde{v} & \tilde{w}\end{bmatrix}^\top$, the relationship between the world 3D coordinates $\mathbf{X} = \begin{bmatrix}X & Y & Z\end{bmatrix}^\top$ and the pixel coordinates u $= \tilde{u}/\tilde{w}$ and v $= \tilde{v}/\tilde{w}$ can be written in a linear manner using matrix notation.

The origin in the image plane is assumed to be at the principal point, being the point on the image plane located in a straight line from the optical center. However, this may not be true in practice and hence the two offset parameters

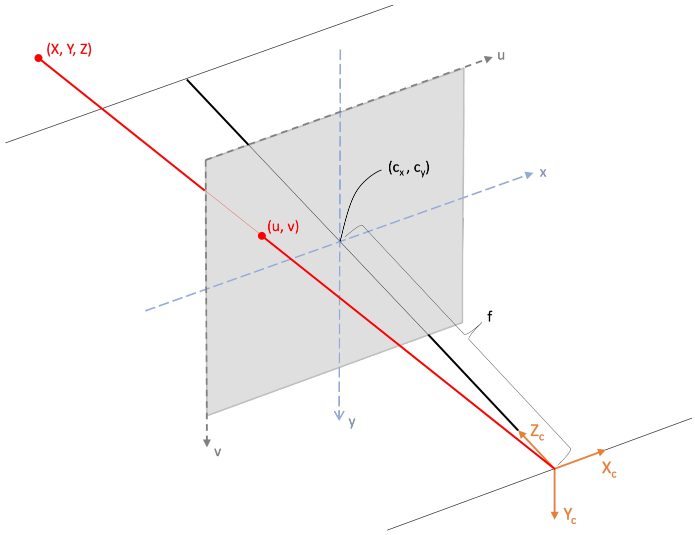


Figure 2.2: The projective camera model

c_x and c_y are introduced. While the focal length f is specified in millimeters, the pixel coordinates u and v are supplied in pixels. Therefore, in order to obtain the pixel coordinates, the s_x and s_y pixel densities are supplied. When employing the whole set of pinhole model parameters, the skew parameter s is the final effect to consider. This will ultimately account for skew sensors and is often set to zero. Ultimately the complete matrix K is reached, the calibration matrix, defining the pinhole camera model in (2.2). The so-called intrinsic of the camera are contained in this matrix [25].

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \underbrace{\begin{bmatrix} s_x f & s & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{K}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.2}$$

The necessity for a link between the world coordinate frame and the camera coordinate frame occurs because points in the 3D world are frequently presented in the world coordinate frame. This is given in terms of a rotation matrix $\mathbf{R}$ and a translations vector $\mathbf{t}$. By defining the world 3D homogeneous coordinates as $\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{X} & \tilde{Y} & \tilde{Z} & \tilde{W} \end{bmatrix}^\top$ implicitly related to $\mathbf{X}$ by $\mathrm{X} = \tilde{X}/\tilde{W}$, $\mathrm{Y} = \tilde{Y}/\tilde{W}$ and $\mathrm{Z} = \tilde{Z}/\tilde{W}$ the general pinhole camera matrix of dimensions 3x4, $\mathbf{P}$, can be represented as

$$\tilde{\mathbf{u}} = \underbrace{\mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}}_{\mathbf{P}} \tilde{\mathbf{X}}. \tag{2.3}$$

2.3 Features in images

The practice of identifying details about a certain area in an image where the picture possesses a particular quality is known as *feature detection*. Numerous applications of computer vision require feature recognition and the matching of these features such as image stitching, 3D model generation, and pose estimation [25]. Such features can for example be church spires, interesting-looking mosaics, or doorways and are also often called key points. The way these key points are described in computer vision algorithms is by describing how the pixels are arranged around them. Other features in images that are easy to detect are *edges*, for example, the transition from a building to the sky behind it. Such edges can be described based on their orientation, curvature or length [25].

A commonly used method for feature detection is the *Harris corner detection* [27]. This method is using corners as features. Corners in images have very large gradients and in a neighborhood of a corner the gradient swing sharply [26]. By looking at the properties of the gradient in windows or sections of the image, areas in the images with corners are found. The Harris corner detector scores point based on both the magnitude and directions of gradients within these windows to find features in terms of corners. Other methods based on the same properties of corners but by using another scoring function exist such as Shi and Tomasi's *good features to track* [28]. One of the weaknesses of harris corner detection is that it is not invariant to scale. Hence, matching features from Harris corner detection from the same image with different scales/zoom levels will most likely not result in any match between the features. Alternatives exist that can address issues, such as Scale-Invariant-Feature-Transform (SIFT). SIFT is invariant to scale, rotations, and partial to illumination changes and hence more robust.

Instead of using corners as features, edges can also be used as features in images. However, as edges are spread around multiple pixels in images they are not suited for the detection of single feature points in images but rather as a feature of the image itself. This can for example be useful for applications where the approach of Bag-of-Visual-Words is used [25]. Canny edge detector or Hough transform are two examples of feature detection methods for finding edges in images [25]. In the same way as edges, circles can be detected in images and used as features. Hough transform can also be used to detect circles in images [29].

Fiducial markers, also known as visual markers or artificial markers, can be added to images to provide distinct, easily recognizable features for image processing algorithms. These markers have highly distinctive patterns and visual characteristics that make them easy to identify even in noisy or cluttered images. They often include a particular encoding or identification number to reduce the likelihood of false positives and improve accuracy [30].

One example of a fiducial marker is the AprilTag, which is a popular choice for robotics applications due to its robustness and reliability [15]. AprilTags use a binary encoding to represent the identification number, which is embedded in a black and white dot pattern that forms the visual marker. This pattern is designed to be easily recognizable by a machine vision system, even when the marker is partially occluded or distorted.

While fiducial markers can provide reliable and accurate results, they are not always feasible to use. In some cases, it may not be possible or practical to add artificial markers to existing images. In these situations, other methods such as feature detection and matching may be used to identify distinct features in the image. These methods may not be as reliable or accurate as using fiducial markers, but they can still provide useful information for image processing tasks.

2.4 Pose estimation

The problem of predicting the pose of a calibrated camera from a set of n 3D points, given in world coordinates and their associated 2D projections in the image, is known as the Perspective-n-Point (PnP) problem. The six DoF in the camera pose are made up of the rotation (roll, pitch, and yaw) and three-dimensional translation of the camera with respect to the world coordinate system. One way of solving this PnP problem is by using Direct Linear Transform (DLT) [25].

In the special case where all the world coordinate points are lying on a common plane where the z coordinate is 0, the concept of homography can be introduced, leading to simplifications of the PnP problem. When a planar relationship exists between the points in the image and the points in the real world a homography can be applied to transform between them. The homography is essentially a transformation involving a rotation and a translation between two frames. A homography can be represented using a 3x3 matrix, **H**, transforming 2D points on a plane into image points. Furthermore, the rotation and the translation between the two frames can be extracted from the homography matrix **H**.

To derive the homography matrix **H**, the starting point is as in (2.3) where the relationship between 3D world coordinates given in homogeneous coordinates and the homogeneous image points is stated.

$$\tilde{\mathbf{u}} = \begin{bmatrix} s_x f & s & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ \tilde{W} \end{bmatrix} \tag{2.4}$$

As the homography matrix is to be calculated under the assumption that all the world 3D points lie on the same plane with Z = 0, the homogeneous world 3D coordinate vector becomes $\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{X} & \tilde{Y} & 0 & \tilde{W} \end{bmatrix}^\top$, and hence 2.4 becomes

$$\tilde{\mathbf{u}} = \begin{bmatrix} s_x f & s & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ 0 \\ \tilde{W} \end{bmatrix} \tag{2.5}$$

$$= \begin{bmatrix} s_x f & s & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{W} \end{bmatrix} \tag{2.6}$$

$$= \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{W} \end{bmatrix} \tag{2.7}$$

where the 3x3 matrix **H** is the so called *homography matrix*.

Given planar world points and the corresponding image points, the homography matrix can be calculated by using DLT as in [31]. The system in (2.4) is stated on a linear form explicitly giving the transformation from the world coordinated **X** by $\mathrm{X} = \tilde{X}/\tilde{W}$ and $\mathrm{Y} = \tilde{Y}/\tilde{W}$ to the image coordinates $\mathrm{u} = \tilde{u}/\tilde{w}$ and $\mathrm{v} = \tilde{v}/\tilde{w}$ as follows:

$$\begin{aligned} u &= \frac{h_{11}X + h_{12}Y + h_{13}}{h_{31}X + h_{32}Y + h_{33}} \\ v &= \frac{h_{21}X + h_{22}Y + h_{23}}{h_{31}X + h_{32}Y + h_{33}}. \end{aligned} \tag{2.8}$$

Furthermore, (2.8) can be stated as a linear homogeneous system according to the following

$$\mathbf{Ah} = \mathbf{0} \tag{2.9}$$

where

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{A_1} & \cdots & \mathbf{A_n} \end{bmatrix}^\top \\ \mathbf{A}_i &= \begin{bmatrix} X_i & Y_i & 1 & 0 & 0 & 0 & -X_i u_i & -Y_i u_i & -u_i \\ 0 & 0 & 0 & X_i & Y_i & 1 & -X_i v_i & -Y_i v_i & -v_i \end{bmatrix} \\ \mathbf{h} &= \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}^\top. \end{aligned} \tag{2.10}$$

The column vector **h** now consists of the elements in the homography matrix **H**, and can be found by solving the homogeneous system in (2.9) by Singular Value Decomposition (SVD). According to [31] the solution, **h**, can be found as the the last column of **V**, where $\mathbf{A} = \mathbf{UDV}^\top$.

From (2.9) and (2.10) it is clear that one point correspondence $(u, v) \longleftrightarrow (X, Y, Z)$ gives rise to two independent equations in the matrices $\mathbf{A}_i$. Despite having nine entries, the homography matrix **H** only has eight degrees of freedom since it is specified up to a scaling factor. Therefore, the minimum amount of point correspondences n needed to determine $\mathbf{H}$ is $n = 4$ [31]. To extract the pose from the homography matrix, $\mathbf{H}$ can be premultiplied with the inverse of the intrinsics matrix $\mathbf{K}^{-1}$ resulting in a matrix containing candidates for the rotation and translation:

$$\mathbf{K}^{-1}\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \mathbf{K}^{-1}\mathbf{K}\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix}. \tag{2.11}$$

The columns in the last matrix in (2.11) cannot be interpreted directly as two of the columns in the rotation matrix $\mathbf{R}$ and translation $\mathbf{t}$. This is because of the previously mentioned scale factor. This factor can be found by exploiting the fact that the columns in the rotation matrix $\mathbf{R}$ are of length one. Hence the scale can be found. In order to extract the full rotation matrix $\mathbf{R}$ and not only the two first columns another of the rotations matrix's properties can be used. Namely the fact that all the column vectors in it are orthogonal, and the last column can therefore be found as the cross product of the first and second columns. The translation vector $\mathbf{t}$ can be found by dividing the last column by the previously mentioned scale factor.

2.4.1 Iterative minimization methods

The homography matrix $\mathbf{H}$ can be further optimized to reduce the reprojection error, being the distance error between a projected point using $\mathbf{H}$ and the measured one. Non-linear iterative optimization, such as Gauss-Newton or Levenberg-Marquardt (LM) optimization, are typical methods for doing this [31].

Let this reprojection error for a single world-image correspondence be denoted

$$r(\tilde{\mathbf{u}}, \mathbf{H}\tilde{\mathbf{X}}) = f(\mathbf{H})$$

by summing over all the point correspondences, i, an objective function can be written as a function of the homography matrix $\mathbf{H}$:

$$E(\mathbf{H}) = \sum_i f_i(\mathbf{H})^2. \tag{2.12}$$

The goal is now to iteratively optimize the homography $\mathbf{H}$ in such a way that the total reprojection error in the objective function in (2.12) is minimized, i.e. find a step δ such that the estimated homography matrix $\hat{\mathbf{H}}$ is updated according to

$$\hat{\mathbf{H}} \leftarrow \hat{\mathbf{H}} + \delta.$$

The Gauss-Newton method can be viewed as the classical Newton method with a modification [32]. Instead of solving the standard Newton equations: $\nabla^2 f(\mathbf{H})\delta^N =$

$\nabla f(\mathbf{H})$, the hessian matrix $\Delta^2 f(\mathbf{H})$ in Gauss-Newton is approximated as $\nabla^2 f(\mathbf{H}) \approx \mathbf{J}^\top\mathbf{J}$. Where $\mathbf{J}$ is the Jacobian found as a N x M matrix where n is the number of residuals in the objective functions, and m is the number of parameters in $\mathbf{H}$. Hence to find the update step δ^{GN} in the Gauss-Newton method the following linear system is solved:

$$\mathbf{J}^\top\mathbf{J}\delta^{GN} = -\mathbf{J}^\top \tag{2.13}$$

In the process of solving the linear system in (2.13), the matrix $\mathbf{J}^\top\mathbf{J}$ has to be inverted. It may happen that this matrix is singular and therefore not invertible. Instead of solving the linear system in (2.13), the LM method solves the following system:

$$(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I})\delta^{LM} = -\mathbf{J}^\top, \tag{2.14}$$

where λ is a scalar that assures the matrix on the left side of the equation does not have a rank deficiency. If the system is solved and the error decreases, the increment is accepted, and λ is divided by a factor before the next iteration. If, on the other hand, the value produces an increase in error, λ is increased by the same amount, and the linear system is solved once more, and so on, until a value that causes a decrease in error is identified. Iteration of the LM method refers to the process of repeatedly solving the linear system for different values of λ until an acceptable solution is found [31].

2.5 Deep Neural Networks

Deep learning is a subset of machine learning, based mainly on the concept of Artificial Neural Network (ANN). ANN is a data structure modeled on how the human brain functions [33]. It is made up of multiple tiny computing components that conduct simple operations and interact with one another to get a result. These small processing units are known as "neurons," and they can contain binary inputs and outputs, as well as more sophisticated inputs and outputs such as floating values [33]. When a single neuron gets an input, it multiplies it by a weight determined by the neuron's decision-making. An input layer, one or more hidden layers, and an output layer comprise a general ANN structure. Each layer is made up of many neurons. An ANN is categorized as a Deep Neural Network (DNN) when it consists of two or more such hidden layers [34]. DNNs is often used in the area of computer vision to solve hard problems such as colorization, segmentation, and classification [35].

The network is trained in order to determine the previously stated weights in the network. In a so-called training set, the network is given correct relationships between input and expected output. The network then alters its neuron weights and biases so that the proper neuron provides output while the remainder of the network's decision-making is unaffected. A cost function, also known as a loss function or objective function, is used to determine how efficient the network is learning. This function may, for example, represent the mean square error between the network's input and output. In general, minimizing a function with several variables is a challenging and computationally difficult process. As a result, updating the network weights is done iteratively, for example, by employing gradient descent to minimize the loss function [33]. Because these deep neural networks have the potential to be exceedingly deep, with millions or billions of parameters, the training process becomes highly computationally costly [34]. As a result, this training method could be enhanced to operate on Graphics Processing Unit (GPU) in order to improve performance and reduce training time [35].

2.5.1 Convolution neural networks

Convolution Neural Network (CNN) are a type of network architecture that is particularly well-suited to image classification; they are quick to train and hence aid in the training of deep, multi-layer networks [35]. CNNs employs filters to recognize characteristics in images. A filter is a set of values that are arranged in such a manner that they identify certain traits, such as vertical edges. A convolution operation is performed to generate a value expressing how sure a certain feature is present by computing the dot product of the filter and the input area where the filter is overlapped.

Pooling layers are also included in the CNNs, in addition to the previously discussed convolutional layers. Pooling layers are typically employed just after convolutional layers. The pooling layers reduce the information in the convolutional layer's output by, for example, only maintaining the maximum value in a feature map formed in the convolution layer. This last strategy is known as max pooling [34].

2.5.2 YOLO - Real-Time Object Detection

You Only Look Once (YOLO) is an object recognition method that takes in an image and attempts to discover all occurrences of each category. The YOLO detector uses a single neural network to predict numerous bounding boxes and class probabilities for those boxes at the same time [36]. The YOLO network design has various versions, one of which is YOLOv4. YOLOv4 is designed to be trained and operated on a single conventional GPU, thus it offers both high accuracy and rapid calculations, making it ideal for real-time applications [37].

2.6 Kalman filtering

The Kalman Filter (KF) is an approach for estimating a state and under some assumptions, it is an optimal solution to the estimation problem. Generating an estimate includes measurements of the system together with predictions and previous knowledge of the system. The KF is the optimal filter under the assumption that the system model is perfect, all the system noise is white and the covariances and the noise are known perfectly [38]. There exists several version of the KF including the Extended Kalman Filter (EKF) and Uncented Kalman Filter (UKF).

The standard KF consists of both a model of the system or process model as well as a model of the measurements. The two models can be described on state space form as follows:

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{v}_k, \qquad \mathbf{v}_k \sim \mathcal{N}(0, Q) \tag{2.15}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{w}_k, \qquad \mathbf{w}_k \sim \mathcal{N}(0, R) \tag{2.16}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{P}_0). \tag{2.17}$$

Here the matrices **A** and **B** are the noise-free system transitions matrices, describing the system transition from time *k-1* to time *k*. Furthermore $\mathbf{v}_k$ is the system process noise and is assumed to be zero-mean Gaussian with variance **Q**. The measurement model is described through the measurement matrix **C** as well as the measurement noise $\mathbf{w}_k$ which also is assumed to be zero-mean Gaussian with variance **R**. The measurement model relates the measurement $\mathbf{z}_k$ at time k with the state $\mathbf{x}_k$ at time k. Finally, the initial state $\mathbf{x}_0$ is normally distributed with mean at the initial estimate $\hat{\mathbf{x}}_0$ and covariance $\mathbf{P}_0$. In order for a solution to exist, the Markov model assumption has to hold. This states that the current state $\mathbf{x}_k$ only is dependent on $\mathbf{x}_{k-1}$ as well as the measurement $\mathbf{z}_k$ only is dependent on the current state $\mathbf{x}_k$ [38].

The models provided are linear, therefore the filter won't work if either the model dynamics or the measurement are non-linear. Hence the need of the Extended Kalman Filter (EKF) arises. The EKF is an extension to the normal KF where the process dynamics and/or the measurement dynamics are linearized around each estimate. Now let the transition from the state at time *k-1* → *k* be described by the non-linear function $\mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k)$ and the measurement dynamics in the same manner as the non-linear function $\mathbf{h}(\mathbf{x}_{k-1}, \mathbf{u}_k)$ with the same noise characteristics for both processes in the linear case. Hence 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 according to [38]

$$\mathbf{F}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k) = \mathbf{F}_k = \frac{\partial}{\partial \mathbf{x}_{k-1}} \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) \bigg|_{\mathbf{x}_{k-1} = \hat{\mathbf{x}}_{k-1}} \tag{2.18}$$

$$\mathbf{H}(\hat{\mathbf{x}}_{k|k-1}, \mathbf{u}_k) = \mathbf{H}_k = \frac{\partial}{\partial \mathbf{x}_k} \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k) \bigg|_{\mathbf{x}_k = \hat{\mathbf{x}}_{k|k-1}} . \tag{2.19}$$

The filtering process using the KF consists of two parts: the prediction step and the update step. In the prediction step the state estimate $\hat{\mathbf{x}}_{k|k-1}$ as well as the covariance $\mathbf{P}_{k|k-1}$ is updated according to the following

$$\hat{\mathbf{x}} = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.20}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^\top + \mathbf{Q}. \tag{2.21}$$

In order to do the update step in an optimal way, the predicted measurement, $\hat{\mathbf{z}}_{k|k-1}$, given the state estimate is needed and the *innovation* $\boldsymbol{\nu}_k$ is defined as

$$\boldsymbol{\nu}_k \equiv \mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) = \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1}. \tag{2.22}$$

For the filter to weigh how much it should correct the predicted state estimate after it has received a measurement the concept of a Kalman gain $\mathbf{W}_k$ has to be introduced. In the case of a linear process model as well as a linear measurement model, it can be shown that this Kalman gain $\mathbf{W}_k$ leads to the optimal update. This proof of optimality is lost for the situations where non-linearities are introduced and linearizations are performed. Nevertheless, the EKF has shown to produce very good results [38]. The Kalman gain $\mathbf{W}_k$ is calculated according to the following

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R} \tag{2.23}$$

$$\mathbf{W}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}. \tag{2.24}$$

Finally the state estimate $\hat{\mathbf{x}}_k$ and the covariance $\mathbf{P}_k$ can be updated with the measurement according to

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{W}_k \nu_k \tag{2.25}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{W}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \tag{2.26}$$

Chapter 3

Experimental setup

3.1 Parrot Anafi FPV drone

The drone used for the experiments in this thesis is the Anafi FPV drone from the French drone manufacturer *Parrot*[1]. As seen in figure 3.1 the drone has four propellers mounted at the end of foldable arms. The drone is fitted with a gimbal, camera, and various other sensors, and it weigths 315 grams. The most relevant aspects of the drone will be mentioned in the following sections, however, the full specifications of the drone can be found in [39].

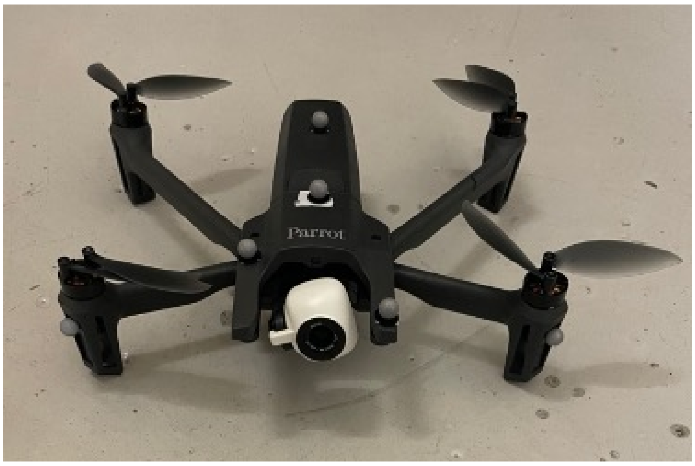


Figure 3.1: Parrot Anafi FPV

3.1.1 Camera and gimbal

The drone is fitted with a 4K camera capable of local storage of 24 Frames Per Second (FPS) at a resolution of 4096 × 2160. As well as local storage the drone also provides live streaming of the video to a phone or a computer. However, the

---

[1] https://www.parrot.com/en

video quality when streaming is lower than the one locally stored. The streaming video is at 30 FPS at a resolution of 1280 × 720, with a latency of 280 ms end-to-end.

The camera is mounted on a gimbal with 2 mechanical axes and 3 electronic axes with electronic image stabilization. The gimbal has a tilt capability of a total of 180 degrees, enabling the camera to be tilted straight up as well as straight down.

3.1.2 Other sensors and internal state estimation

In addition to the camera mounted on the gimbal, the drone is fitted with several other sensors:

- Invensense MPU-6000 **IMU** with:
    - 3-axis gyroscope
    - 3-axis accelerometer
- ST Microelectronics LPS22HB **barometer**
- AKM AK8963 **magnetometer**
- U-BLOX UBX-M8030 **GPS** with 4 constellations:
    - GPS L1
    - Galileo E1
    - Glonass L1
    - BeiDou B1C
- **Ultrasonar** for height measurement. Sensor specifications not identified.
- **Vertical camera** MX388, fixed resolution of 640x480 used for optical flow estimation.

For the purpose of estimating the internal states of the drone, it has a Extended Kalman Filter (EKF) including the following states:

- Body velocities
- Attitude in Euler angles
- Accelerometer biases
- Gyro biases
- Pressure sensor bias
- Position in NED frame
- Wind on x and y axes

3.2 Other software and hardware

For reading sensor data as well as sending commands to the drone, the Parrot Olympe Application Programming Interface (API)[2] provided by parrot was used. The version used for this thesis is version 7.3. The API can be used to control the

---

[2] https://developer.parrot.com/docs/olympe/index.html

drone with custom programming scripts from a ubuntu machine. The API communicates with the drone through a set of custom message types called ARSDK-messages. The API is not meant for closed-loop control, and hence the update frequency of the internal states of the drone is only at 5hz. However, some meta-data is available with the camera feed at 30 FPS.

As the previous work on the project has been developed by the use of Robot Operating System (ROS) with ubuntu 18.04 and ROS-version *Melodic*, it was desirable to continue with this. However as the updated version of the Parrot Olympe required the use of python 3 (that is standard with ubuntu version 20.04), and ROS Melodic is End-of-life in May 2023[3] it was advisable to change the ubuntu version to 20.04 and to use ROS *Noetic Ninjemys* to improve the long time support of the developed code.

In this thesis, the Anafi drone was accessed using the Parrot Skycontroller 3[4]. The Anafi drone can also be accessed directly by connecting to it via WiFi. However, as the author of [1] pointed out, lower signal latencies, as well as longer signal range, can be expected when connecting to the Anafi drone through the SkyController 3. The SkyController 3 was connected to the computer with USB-C. A Komplett Khameleon provided by the institute served as the computer for all of the experiments in this thesis. Its specifications can be found in table 3.1. In order to be able to utilize the computer's Graphics Processing Unit (GPU) in the perception algorithm, the computer was installed with both CUDA for enabling GPU acceleration and CuDNN for GPU accelerated DNN support.

Computer specifications	
Manufacturer	Komplett
Computer type	Laptop
Model name	Khameleon P9 Pro
CPU	Core i7-9750H
GPU	GeForce RTX 2070, 8GB
RAM	32 GB
Software spesifications	
Ubuntu	20.04 Focal Fossa
Python	3.8.10
ROS	Noetic Ninjemys
Parrot Olympe API	7.3
Parrot Sphinx simulator	2.9.1
Nvidia drivers	515.65.01
Cuda	11.7
YOLO	V4
OpenCV	4.2.0

Table 3.1: Computer specifications and software versions used in this thesis.

---

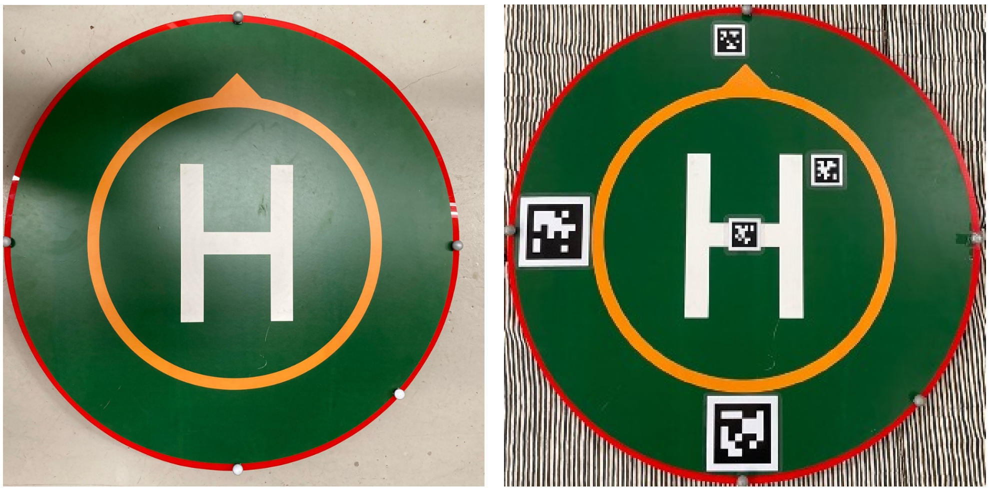
[3] http://wiki.ros.org/Distributions

[4] https://www.parrot.com/us/support/anafi/how-does-the-skycontroller-3-work

3.3 Landing platform

The landing platform utilized in this thesis was designed and built by Sundvoll in [3]. Figure 3.2a depicts the platform. The platform has an overall diameter of 80 cm, with a white "H" in the center covering 1/3 of the entire diameter on the long and 1/4 on the short sides. It also features a yellow circle with a diameter of 50 cm and an arrow indicating the platform's direction. The platform is intended to be fitted aboard a 1:20 model of the autonomous maritime vessel DNV ReVolt[5] as shown in figure 3.3.

In this thesis, the platform was enhanced with the inclusion of AprilTags. The AprilTags were arranged in the patterns depicted in figure 3.2b, with a small AprilTag in the center and larger ones closer to the platform's perimeter. The AprilTags are taken from [40] and belong to the "Tag family" *tag36h11*, with all markers being square with white borders. The AprilTags were laminated and double-sided taped to the platform.



(a) Original. Image from [1]

(b) With AprilTags.

Figure 3.2: Landing platform

---

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

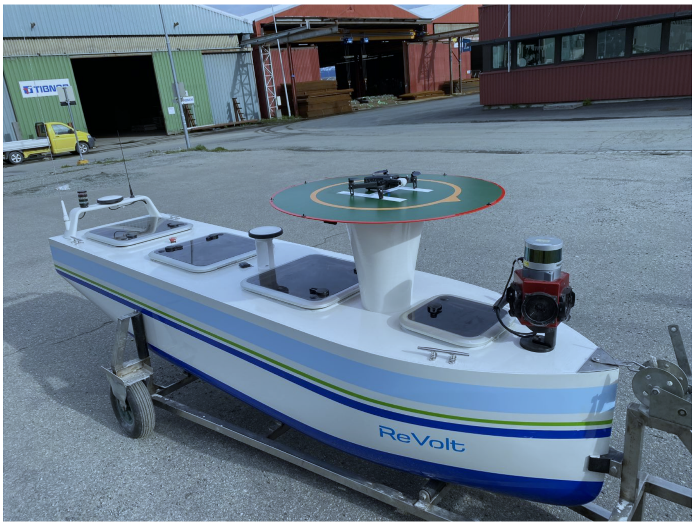


Figure 3.3: The platform mounted on the ReVolt. Image from [1]

3.4 Parrot Sphinx simulator

Parrot Sphinx [41] is a simulator for portraying Parrot drones. It is based on Unreal Engine 4, a gaming engine that can render multiple environments. The simulator includes multiple 3D models of all of Parrot's drones, including the Anafi FPV drone utilized in this thesis. The simulator attempts to make the transition between using a real drone and a simulated one as fluid as possible, with the only difference being the IP address you provide to the drone. This version of Parrot Sphinx is substantially different from the one the author of [1] used in his thesis. The previous version build on the Gazebo simulator allowed for eg. the extraction of positions of objects in the world, as well as moving other objects such as the landing platform. This is no longer supported in the new version of the simulator, and the simulator will not be mentioned and used that much in the development of this thesis, nevertheless, it is still described for the sake of completeness. Figure 3.4 shows the Anafi drone standing on the platform in Parrot Sphinx simulator.

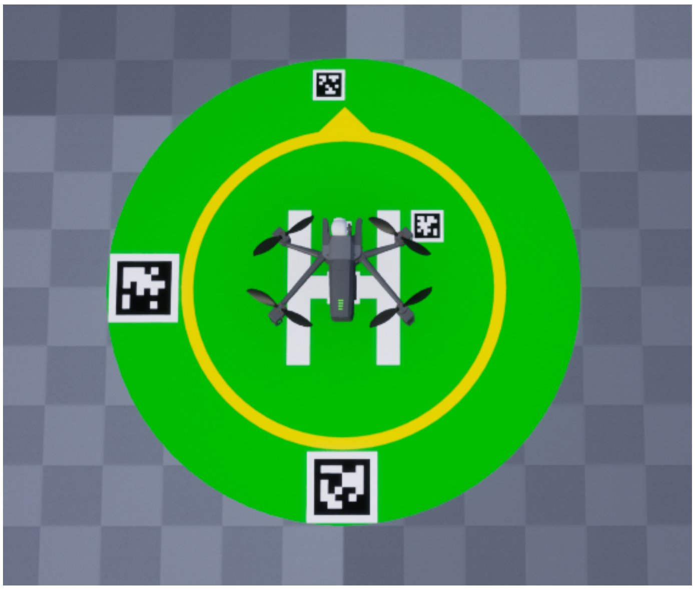


Figure 3.4: Anafi drone and platform portrayed in Parrot Shinx

3.5 Drone lab

The drone lab at NTNU was utilized as the testing location for the work done in this thesis. The NTNU Drone Lab is situated on the second basement level of Elektrobygget at NTNU Gløshaugen. The drone lab is equipped with foam mattresses on the floor and a net that surrounds the flight zone, giving it a secure environment for testing as seen in figure 3.5.

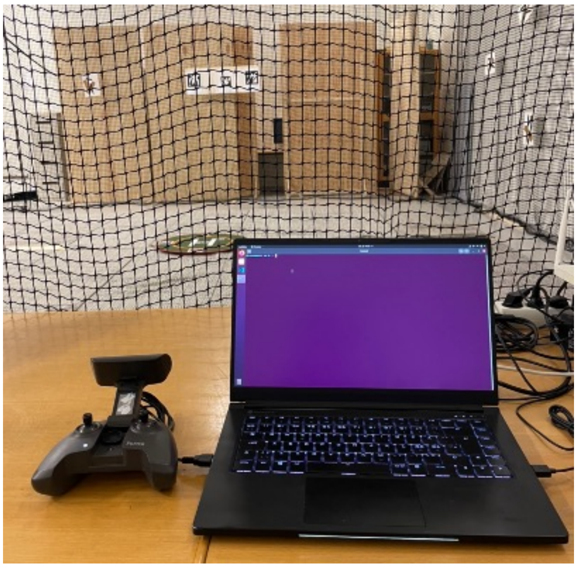


Figure 3.5: Set up at the drone lab at NTNU

3.5.1 Qualisys motion capture system

The Qualisys Motion Capture system [42] is installed at the drone lab. This system is a precision motion capture and 3D positioning tracking system that uses multiple cameras to detect reflective markers in a scene to determine the precise locations and orientations of objects in it. Qualisys claims a location precision of 1mm and an orientation accuracy of 0.1 degrees [43]. It is possible to define objects with multiple reflecting markers and retrieve their pose in the 3D scene relative to a calibrated world origin by using Qualisys Track Manager[6]. This system is utilized in this thesis to obtain the ground truth location and orientation of objects in the scene in order to assess the performance of the algorithms developed.

The Qualisys Motion Tracker software is installed on a separate computer from the one that is connected to the drone. The Qualisys Motion Tracker computer publishes the position, orientation, and speed of objects in the scene on predefined ROS topics. The computer donnected to the Anafi drone must be able to interact with the other computer in order to access these ROS topics. This is accomplished through the use of a multimachine configuration in ROS, in which both machines are connected to the same network. A WiFi router exclusively connected to the two machines was set up to decrease latency in the communication channel. Furthermore, the computer connected to the Anafi drone launches a roscore in one terminal, while the computer running the Qualisys Motion Tracker is configured to use this roscore on the other computer. As a result, the topics comprising the pose and velocity data of the objects in the scene will be accessible on the computer connected to the Anafi drone.

3.5.2 Marker placement and objects

To avoid unclear poses that might result in inaccurate system outputs from the Qualisys Motion Tracker, the markers were set asynchronously around the axes of the objects that were tracked. 5 markers were set around the border of the platform, and 9 smaller markers were set on the Anafi drone body. The markers on the Anafi drone were smaller due to weight and space limitations. They were located on the top, back, batteries, and at the end of each leg. Figure 3.6 shows the platform and the drone with the markers on them.

---

[6] https://www.qualisys.com/software/qualisys-track-manager/

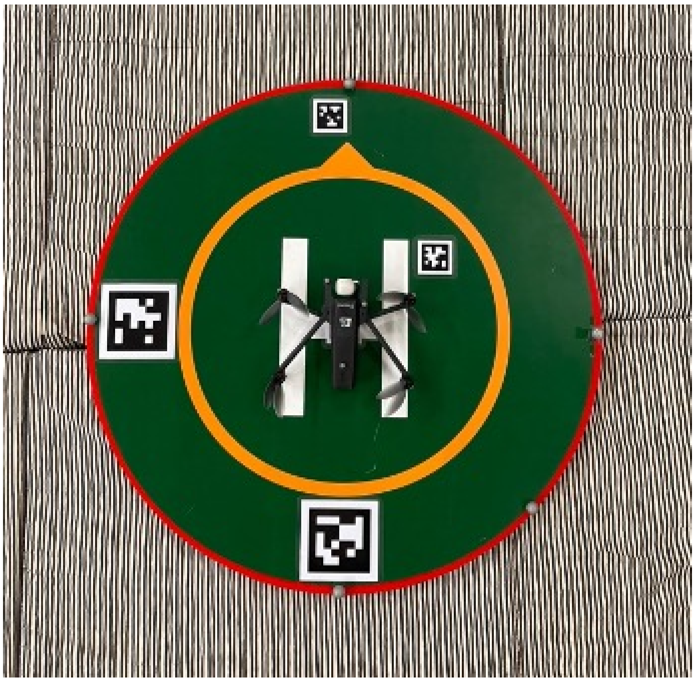


Figure 3.6: Anafi drone and platform with markers

3.5.3 Outdoor testing

The Civil Aviation Authority sets Norway's drone usage regulations. Drone flying outside requires adherence to these regulations, and every drone that weighs more than 250 grams or includes a camera must be registered and insured. Additionally, in order to be authorized to operate drones in classes A1 and A3, the drone operator must be registered and have successfully completed an online course to get a certificate. The author of this thesis is a registered private drone operator with completed A1 and A3 certifications and has drone insurance through home contents insurance.

Chapter 4

Methodology

4.1 Development

This section covers the development approach and provides an overview of the system architecture.

4.1.1 Design

The starting point of the software development process for this thesis was a sizable codebase because it is a direct descendent of the work done in [1]. Three design concepts were specified by the author of [1]: modularity, platform independence, and testability. The same ideas will be used to guide further development.

Modularity

The capacity of a software code base to be separated into distinct parts with standardized inputs and outputs is referred to as modularity. By following this principle, the dependencies between various code components may be minimized, making the code simple to maintain and fix without breaking. Every code module is divided into its own ROS package, and the inputs and outputs are based on standardized ROS topics, making ROS an excellent tool for this process. The modularity principle could not be stated to have been preserved in the code base from [1] without the need for extensive restructuring. In order to try to fix this every module was first placed into its own package, which would eventually result in the code base being modularized and manageable. The code base also heavily relied on customized ROS messages when it could have utilized preset messages that were well-known to the ROS community. In order to make the code more intelligible for engineers who are not familiar with the code base, several of the message types were altered to utilize standard ROS messages rather than specialized messages.

Platform independence

In the context of software development, platform independence refers to the ability of the code base to be quickly migrated to a different hardware platform with little to no coding modifications required. This is particularly significant in this project since the work completed is a scaled-down representation of a real-world SAR operation at sea. The project has to be simple to distribute to additional computers and drones. By putting the drone-specific features in their own interface and publishing all data from and to the drone on ROS-topics, the creator of [1] upholds this idea. Additionally, a configuration file maintained in the code, makes the change of external parameters of significance easy. The software created for this thesis will be developed using the same approach.

Testability

The code must be able to be tested in a lab or simulation environment without requiring changes to the algorithms themselves in order to assess the performance of the work. Before testing the code in a real-world experiment, it is crucial to filter out any potential bugs in the code in simulations or in the lab. Whether the drone is a real one, one that is flown in a lab, or one that is simulated, the drone interface package in this thesis is the same. Furthermore, regardless of the environment, in which the drone is operated, all algorithms are the same. The only modification is the addition of the Qualisys ground truth data. When the drone is in flight, all data is simply recorded into rosbags[1], and all assessment and transformation of the ground truth data is done in post-process. Because the entire computational pipeline may be heavy, this avoids using essential CPU capacity to handle this ground truth data. In this manner, the performance of the algorithms may be quickly assessed in the post-processing stage without significantly degrading the computer performance.

4.1.2 Olympe ROS interface

As mentioned by Falang in [1], the Parrot Olympe API received a substantial update from the version he used. The ROS interface developed by Falang was therefore not applicable. Additionally, as it had never been done before by Falang, it was desirable to utilize the metadata supplied along with the camera stream. Hence, the need of an updated version of the interface was needed. The new interface is based on a GitHub repository previously available by the Github user *andriyukr*[2]. This interface was originally written for a Parrot Olympe API version that was much closer to version 7.3 than the one used in [1] for version 3.0.0. The Parrot Olympe API unfortunately does not has a changelog, and it was, therefore, easier to make the GitHub version compatible with Parrot Olympe API version 7.3. The

---

[1] `http://wiki.ros.org/rosbag`

[2] https://github.com/andriyukr

interface was revised with new functionalities listed in table 4.1.

The Olympe ROS bridge consists of multiple input command topics that send commands to the drone, and output topics containing sensor data from the drone. The different topics are listed in table 4.2.

Revision	Description
General version update	Update all API-commands to match the ones used in Olympe version 7.3
Frame transformations	Changing the frame convention from ENU to NED
Remove most of the custom ROS messages	Changing from custom messages to standard ROS messages where possible
Add simulated GNSS-data	Pose estimates from the Qualisys system acts as GNSS-data at the drone lab
Correcting attitude estimates	Remove the attitude measurement bias from the Anafi drone
Extract metadata	Extract the metadata that comes along with the camera stream and publish it on ROS topics
Utilize EventListeners dataclass	Instead of polling the sensors to get data, utilize a new data class in the API to get sensor readings at 5Hz.

Table 4.1: Revisions on the ROS olympe bridge

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/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	SkyControllerCommand *	If the SkyController3 is used to control the drone, it publishes the commands from the controller

Table 4.2: Command topics and output topics from the Olympe ROS bridge. Message types marked with '*' are custom messages. Table is a cooperation with the author of [4].

4.2 Coordinate frames and convertions

This section will specify which frame of reference the following measurements and estimations will be provided since the thesis uses many frames of reference. It will also explain how to transition between the most crucial frames. The different frames are shown in figure 4.1 and the different estimations and data frames are summarized in table 4.3.

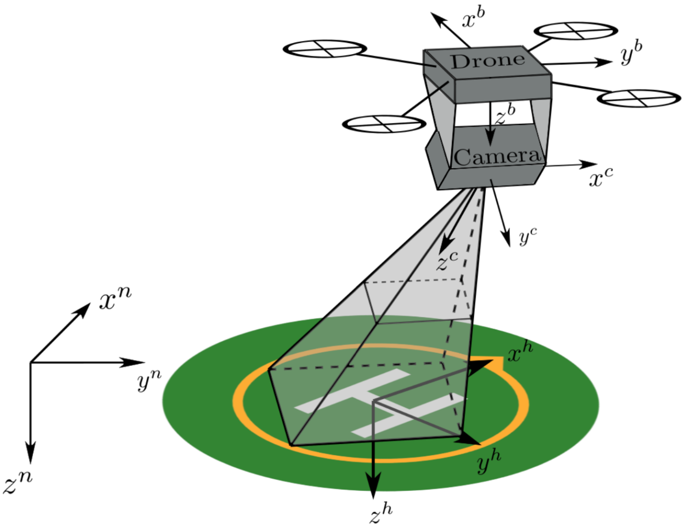


Figure 4.1: Coordinate frames NED {n}, platform {h}, drone body {b}, and camera frame {c}. Image from [1]

System	Output reference frame
AprilTags pose estimate	camera {c}
DNN-CV position estimate	camera {c}
Kalman filter	body {b}
Qualisys Motion Tracker - ground truth data	NED {n}

Table 4.3: Systems and output frames of reference

4.2.1 Convertions

Camera to drone body frame

It is required to transform the estimates into the body frame before applying them in the Kalman filter since both the DNN-CV position estimator and the Apriltags pose estimator provide estimates in the camera frame of reference. At first, it was attempted to position the camera on the gimbal such that it was facing directly down within the body frame. If so, the transition from the camera to the body frame would require a 90-degree rotation around the z-axis. However, since the Anafi drone's internal controller controls the gimbal in relation to the world frame and the gimbal stabilization algorithms maintain the gimbal's stability in the world frame, this was not possible to do. Initially, it was attempted to compensate for whatever rotations the Anafi drone was going through by sending commands to the gimbal at a rapid frequency, such as ensuring that the camera was always pointed directly down in the body frame. The Anafi drone did not appear to acknowledge the use of such quick orders, and as a result, the gimbal was unable to keep up with the drone's rotations.

Instead, the issue was resolved by configuring the Anafi drone's camera to constantly aim downward in the world frame. Hence, the internal controller inside the Anafi drone operating at 200hz is responsible for stabilizing the camera gimbal at a specific angle. Additionally, it is possible to use the transform when necessary by creating a ROS transform broadcaster[3] that publishes the transform from the camera frame to the body frame. The transform involves first a rotation of the camera frame of 90 degrees around the z-axis of the camera frame and a translation to compensate for the offset between camera frame origin and body frame origin. The offset is estimated to be 70mm on the x-axis and zero on the z- and y-axes. Thereafter it involves the rotation from the NED frame to the body, only including the roll and pitch angles of the Anafi drone body. The gimbal can only do pitch and roll motions; a yaw motion is not possible, hence the yaw angles are not compensated for. The full transformation will therefore be as follows

$$\mathbf{p}^b = \mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi}(\mathbf{R}_{z,90}\mathbf{p}^c + \mathbf{t}_c^b). \tag{4.1}$$

NED to drone body frame

The ground truth data from the Qualisys Motion Tracker program must be transformed from the NED frame to the body frame in order to be able to assess how well the algorithms performed. A complete rotation around the drone attitude as described in the NED frame and a translation between the NED frame's origin and the drone frame's origin are considered to constitute the NED to body transformation. The transformation will be as follows

$$\mathbf{p}^b = \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi}\mathbf{p}^n + \mathbf{t}_n^b. \tag{4.2}$$

---

[3] `http://wiki.ros.org/tf2`

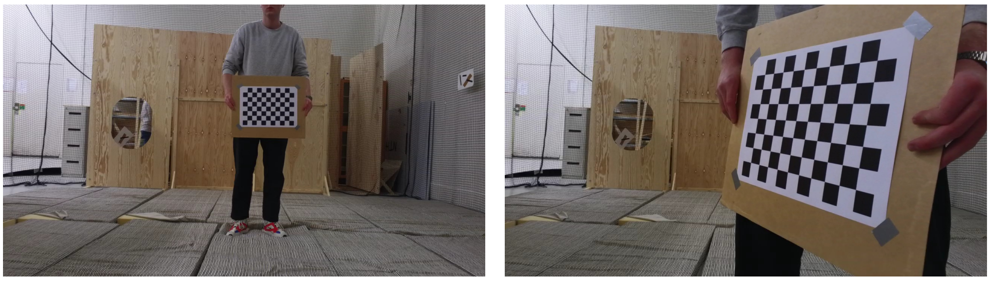


Figure 4.2: Checkerboards used in camera calibration

4.3 Camera calibration

The camera was recalibrated to remove any issues with the previous initial calibration performed by Falang in [1]. The procedure of determining the **K**-matrix in (2.2) from section 4.3 is referred to as calibrating a camera. The real-world coordinates of objects within it are required in order to find the intrinsic matrice **K**. One of the more popular patterns to employ when calibrating a camera is the checkerboard pattern. The reason being them having sharp gradients in two directions and the corners are always on the intersection between two checkerboard lines. The corners of the pattern do also have stable and well-known world coordinates. According to the directions found on [44], a 10x7 checkerboard pattern offered online by Mark Hedley Jones was printed out in A3 format. The side length of the individual squares was 34mm. The checkerboard was printed this large in order to be able to detect the checkerboard at a great distance from the camera. As seen in figure 4.2, the pattern was taped on a flat surface.

The camera on the Anafi drone's gimbal was used to take several pictures of the checkerboard. This was done by writing a ROS node that saved individual images from the video stream as images in .jpeg format in 1-second intervals. To capture the checkerboard from various angles, it was moved in 3D space. It was first attempted to use the calibration pipeline developed by Falang in [1], however, this pipeline contained multiple errors. Instead, a new pipeline was developed consisting of functions from the OpenCV python library. The pipeline creates world coordinates of the checkerboard in an equal distance grid pattern with sidelengths given by the real-world size of the checkerboard squares. Furthermore, it detects the corners in the checkerboard in the image and refines the pixel positions before it uses an OpenCV function to estimate the full camera matrix. Due to the limited amount of documentation using OpenCV with python it was not clear in which order the corners of the checkerboard were detected. In order to cross-evaluate the camera matrix, the camera calibration app in MATLAB[4] was used. This application allows for a more visual approach to the calibration pipeline, where the image corners can be visualized as well with the reprojections and the estimated pose of

---

[4] `https://se.mathworks.com/help/vision/ug/using-the-single-camera-calibrator-app.html`

the camera making it easier to verify the quality of the calibration.

4.4 Latency issues with the Anafi drone

Some serious latency issues were discovered during the initial testing of the codebase developed Falang in [1] together with the new Parrot Olympe API. The problems were found to be caused by updates to the API, which delayed messages coming from the drone when it also received command messages. In his thesis, Falang claimed that he too experienced latency problems, but with the new API, the delays ranged from a few milliseconds to several seconds, making them impossible to forecast. The API is not intended for closed-loop command control, according to Parrot, who notes this in one of their forum postings[5]. The latency issue is not a major issue in the creation of this thesis. This is due to the fact that all of the algorithms created for this work only read data from the Anafi drone. The control algorithms created by Solbø in [4] will be used in parallel with the perception-based pose estimation algorithm established in this thesis. The issue will consequently have a significant impact on the functionality of the entire system composed of the methods created for this and Solbø's thesis [4]. For a more in-depth analysis of the latency issues with the Anafi drone, the reader is directed to Solbø's thesis in [4].

Actions have been taken to lower the processing requirements of the algorithms created in this thesis as well as the internal latency of ROS. Before it is employed in the subsequent algorithms, the image stream from the Anafi drone is republished at a lower frequency of 15Hz. This will eventually result in the algorithms continuing to provide pose estimations at a high rate, but the overall performance of the system, including the work done by Solbø in [4], will be improved and the required processing power will be decreased.

4.5 AprilTag Detection and pose estimation

By identifying AprilTags on the platform and subsequently solving the PnP problem outlined in section 2.4, the AprilTag detection and pose estimation node attempts to estimate the pose of the Anafi drone. This module replaces the TCV module currently available in [1] to address its limitations. The primary problems with the TCV module, according to the author of [1], are the high computational costs, the sparse feature recognition while testing outdoors, and the offsets in every axis. Further tests on it in the lab setting revealed that the TCV solution also had a relatively poor recognition performance while the Anafi drone was going vertically and that the output frequency was generally low.

---

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

To avoid potentially obscuring the detection capabilities of other modules, two larger Apriltags of size 11.7 cm were put on the platform's edge rather than shadowing its inner orange border. In addition, three smaller Apritags of size 5.4 cm were positioned at the platform's center, to the right of the upper corner of the "H", and slightly to the left of the orange arrow of the platform. The larger AprilTags were placed on the outermost border so they could be seen from a greater distance; in contrast, the smaller AprilTags can be seen at a lesser distance when the Anafi drone gets closer to the platform. Additionally, the placement of the AprilTags avoided obtaining coplanar features, which results in a less accurate pose estimate.

Every corner of the AprilTags' physical distance from the platform's origin, the "H"'s center, was measured in the platform coordinate system. The Qualisys Motion Tracker program stated in 3.5.1 was utilized to get accurate measurements of the AprilTags' corners. On each corner of the AprilTags, reflective markers were placed, and their positions were measured. Due to the excellent precision of the Qualisys system, this strategy will be accurate, and more precise than manual measurement. Every marker contained four measurements, i.e., every corner, sorted from corners 0-3. The corner distances were saved to a .txt-file. The distances to the center of the different AprilTags, measured from the center of the platform, can be seen in figure 4.3.

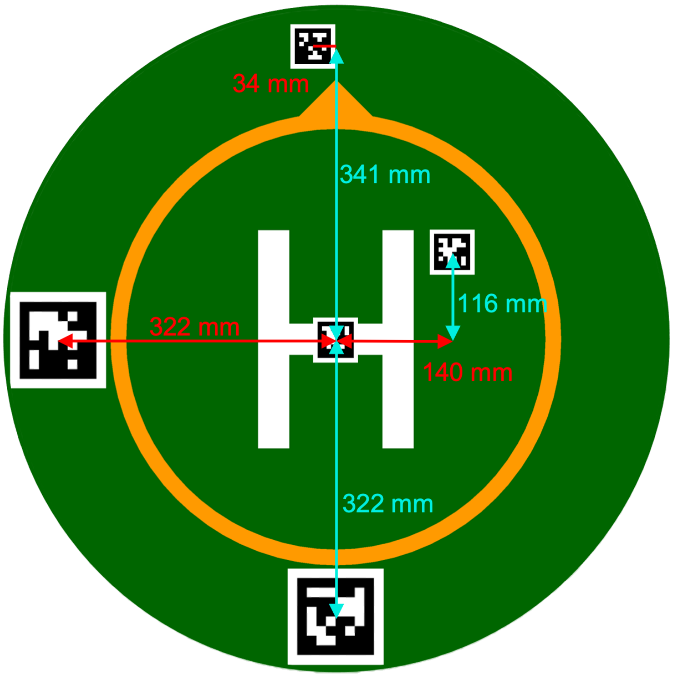


Figure 4.3: Platform with physical distances

The python package available on GitHub from *swatbotics*[6] allows users to detect and identify AprilTags in images. The process begins by converting the images to grayscale, then using the package's detection.detect() object to identify the AprilTags. The output of this process is a dictionary containing information about each AprilTag detected in the image, including its ID, the location of its corners in pixel coordinates, and other relevant details. The next step is to match the corners of the AprilTag in the image with the physical corners of the platform in the reference frame. This involves creating two ordered lists, with the first list containing the corner positions in pixel coordinates and the second list containing the corresponding corner positions in the platform reference frame. This matching process allows the pose of the camera to be estimated using the known 3D positions of the corners in the reference frame and their corresponding 2D positions in the image.

By solving the PnP problem, the camera's pose with respect to the platform may be calculated using the point correspondences between the pixel coordinates and 3D distances of the corners of the Apriltags. The improved OpenCV implementation decreases the pose estimation problem's runtime. The "IPPE" flag may be set when calling the OpenCV function *solvePnP*[7] to use the "Inverse Perspective Point Error" method to solve the PnP problem. This method uses an iterative approach to solve the PnP problem by first estimating the pose using the perspective-n-point algorithm, then refining the pose estimate using a Levenberg-Marquardt optimization. The IPPE method does not use homography estimation itself, but rather uses the initial pose estimate obtained from the PnP algorithm as a starting point for the iterative optimization process. The initial pose estimate obtained from the PnP algorithm may be calculated using the homography-based solution to the PnP problem when the 3D points are planar.

4.6 DNN-CV based position estimation

This section is based on the work done by Hove in [2]. Hove trained a Deep Neural Network in the form of a You Only Look Once (YOLO) detector to detect the three following classes: "Helipad", "Arrow" and "H". The three different classes are shown in figure 4.4. The output from the network is thereafter used to estimate the Anafi drone's position relative to the detected platform. The work done by Hove is adapted to be able to run in ROS Noetic and python 3. However, most of the algorithms are left unchanged.

---

[6] `https://github.com/swatbotics/apriltag`

[7] `https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html`

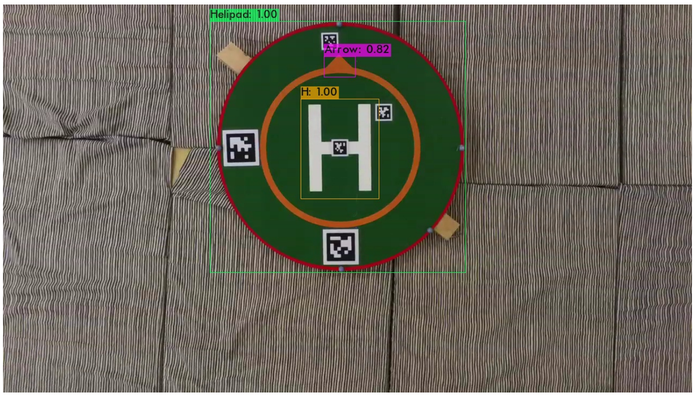


Figure 4.4: Detection output from trained YOLO network showing the three classes: "Helipad", "Arrow", and "H".

Object detection

The detection of objects in the images from the Anafi drone is done by using YOLO v4 as in [2]. The weights used in the network as well as the general structure of the Deep Neural Network are kept the same. The author of [2] provided two sets of weights; one for images coming from the real drone and one set for images coming from the simulator. The network was trained on images coming from the drone used in [2]; the Parrot AR2.0 drone. The Darknet neural network framework [45] serves as the YOLOv4 detector's core. The Darknet ROS bridge at [46] allows the integration of the output from the darknet into ROS. However, ROS Noetic did not automatically fully accept the code in the darknet ROS bridge GitHub repository. This is because the darknet ROS bridge in the GitHub repository can't be used with OpenCV 4.2.0, which is the standard OpenCV version in ROS Noetic. The ROS package with the darknet ROS bridge used in this thesis is a fork of [46], modified to support ROS noetic and openCV 4.2.0. The fork's author is the GitHub user *Ar-Ray-code* and is available on Github[8].

Position estimate

In order to estimate the position from the bounding boxes returned from the darknet ROS bridge, an algorithm originally proposed in [3] is used. This algorithm was the one used in [1, 2]. The algorithm can be seen in algorithm 1. Knowing the center of the platform found from the "Helipad" bounding box, the pixel length and metric length of the platform's radius as well as the camera intrinsics from the new camera matrix **K** will make it possible to estimate the position of the platform using similar triangles. Algorithm 1 returns the position of the platform given in the camera's frame of reference {c}. Hence in order to transform it to the body frame the new transformation from the camera frame of reference {c} to the

---

[8] `https://github.com/Ar-Ray-code/darknet_ros_fp16/tree/noetic-fp16`

body frame of reference {b} was used. This transformation compensates for the rotation of the Anafi drone as described in section 4.2.1.

Algorithm 1 Estimate position from DNN-CV detections.

```
Let $c_x$, $c_y$ be the pixel coordinate of the platform center
Let r, R be the platform pixel radius platform metric radius respectively
procedure EstimatePosition($c_x, c_y, r, R, f$)
    Let f be the camera focal length given in pixels
    Let $o_x, o_y$ be the pixel coordinates of the image center
    $\hat{z}^c \leftarrow \frac{Rf}{r}$                          ▷ Similar triangles
    $d_x \leftarrow o_x - c_x$
    $d_y \leftarrow o_y - c_y$
    $\hat{x}^c \leftarrow \frac{\hat{z}^c d_x}{f}$                ▷ 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
```

4.7 Kalman filter estimation

A Extended Kalman Filter is used to combine the pose estimates from the AprilTag pose estimation module and the DNN-CV-based position estimate together with sensor output from the Anafi and a dynamical model. The EKF used in this thesis is largely based on EKF developed by Falang in [1]. However, it is updated with new measurements, sensor data, and different measurements than the ones Falang used in [1]. As the output from this EKF is meant to be used in the control algorithm developed by Solbø in [4] the output of the Extended Kalman Filter is the distance from the Anafi drone to the platform, given in the body frame. Doing this enables the EKF estimate to be used directly as the error in the control system. Another key factor of using a EKF to estimate the drone-to-platform distance, is the EKFs ability to produce estimates at a much higher rate than what the estimators can provide as stand-alone estimators. The EKF can produce estimates in between every correction measurement using its prediction model. The rate of the EKF is set to 25 Hz to make sure it provides the control algorithm with estimates at a fast enough rate.

The state of the Extended Kalman Filter should be kept as simple as possible according to [38], and hence it contains only three positional states and three velocity states. The states are the position of the platform given in the body frame and the velocity of the Anafi drone given in the body frame. The full state vector is given as the following

$$\mathbf{x} = \begin{bmatrix} x_h^b & y_h^b & z_h^b & v_{d,x}^b & v_{d,y}^b & v_{d,z}^b \end{bmatrix}^\top \tag{4.3}$$

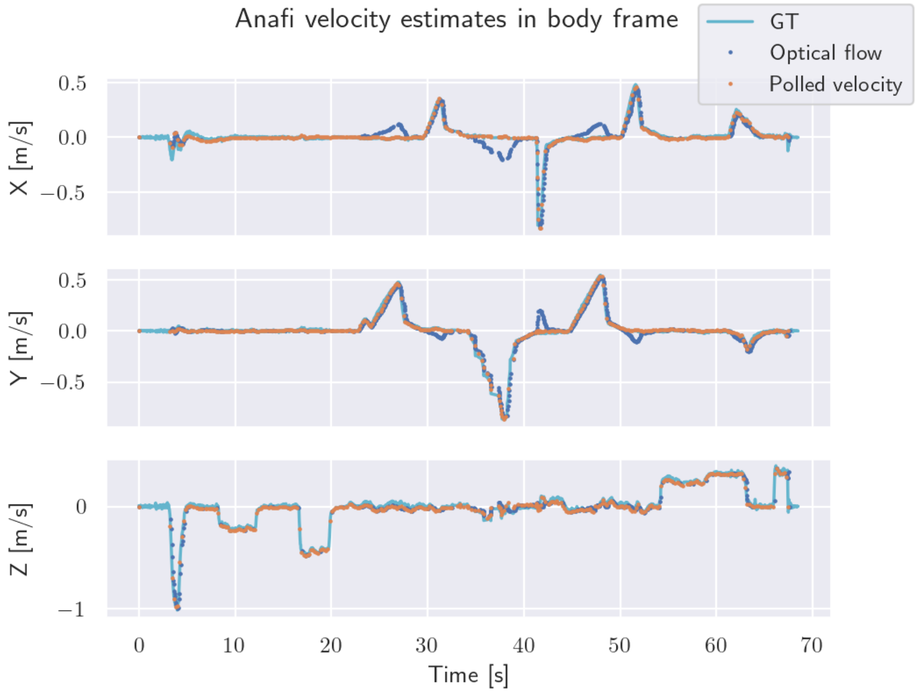


Figure 4.5: Velocity estimates from Anafi drone

Furthermore, the dynamical model in the EKF is a Constant Velocity (CV) model. The model is chosen due to its simplicity, but also its capability to provide good predictions given the assumption that the velocity is constant between samples. This pose estimation system is meant to offer pose estimates to the drone in a landing scenario at sea. When landing the drone on a boat at sea slow speeds for the drone are a necessity. Furthermore, the Anafi provides two different velocity measurements at two different frequencies. It provides a velocity estimate at 30Hz based on optical flow as well as a velocity estimate at 5Hz polled from the internal controller of the Anafi drone. The Optical flow measurement is provided with the metadata coming from the video stream. The author of [1] did not access this metadata in his work. Figure 4.5 shows the two velocity estimates from the Anafi drone plotted against the ground truth velocity. The plot shows that the optical flow estimate has a higher rate, but a change in velocity on either the x or y-axis propagates to the other axis. This velocity estimate is therefore not desirable to use. The estimate at 5Hz polled from the internal EKF of the Anafi drone is therefore the measurement used in the EKF.

According to [38] the CV model can be described in continuous time with the following model

$$\mathbf{x} \;= \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{n}, \tag{4.4}$$

where the matrices **A** and **G** is as following

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{3x3} & \mathbf{I}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \quad \text{and} \quad \mathbf{G} = \begin{bmatrix} \mathbf{0}_{3x3} \\ \mathbf{I}_{3x3} \end{bmatrix}. \tag{4.5}$$

Furthermore, the process noise, **n**, is assumed to be white with diagonal covariance according to

$$\mathbf{n} \sim \mathcal{N}(0, \mathbf{D}\delta(t-\tau)) \quad \text{where} \quad \mathbf{D} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}. \tag{4.6}$$

The σ on the diagonal of the **D** matrix is a measure of how much acceleration the target is expected to undergo [38]. Hence, with knowledge about the likely acceleration in the system, the σ can be estimated roughly.

In order to correct the prediction in the EKF by the constant-velocity model, correcting measurements are needed. The EKF in this thesis is corrected with four different measurements in addition to the previously mentioned velocity estimate. All five correction measurements are given in table 4.4. The AprilTag pose estimate and DNN-CV position measurement are described in section 4.5 and 4.6 respectively. The two last correction measurements are described in the two following subsections; altitude measurement and GNSS-measurement.

Measurement	Description
AprilTag pose estimate	Pose estimate from the AprilTag pose estimation module
DNN-CV position estimate	Pose estimate from the DNN-CV position estimation module
Velocity estimate	Velocity estimate from the Anafi drone.
Altitude measurement	Distance to ground estimate from the internal EKF of the Anafi drone.
GNSS measurement	GNSS measurement in NED converted to body frame. Relative to init position of Anafi drone.

Table 4.4: Correction measurements in the Kalman filter

Altitude measurement

The altitude measurement comes from the Anafi drone's internal EKF. The measurement is shown to be the most accurate at lower altitudes in the descending phase when the Anafi drone is moving straight vertically downwards. This can be seen in figure 4.6. It can also be noted that the estimate is unusable in the initial startup-phase, and hence cannot be used in this case. However, this is not a problem as the internal controller of the drone is responsible for the take-off of the drone. In figure 4.6, the Anafi drone is flown to an altitude of 3 meters, and then subjected to horizontal and vertical velocities, something that makes the internal altitude estimate untrustable. However, when the drone is moving vertically downwards the estimate is reliable with little uncertainty and noise. This measurement is therefore only used to correct the EKF predictions when the predicted altitude in the filter is below 1 m in the descending phase. When the Anafi drone is below this altitude, it is in most situations in a landing scenario and the precision has to be better than when flying at higher altitudes.

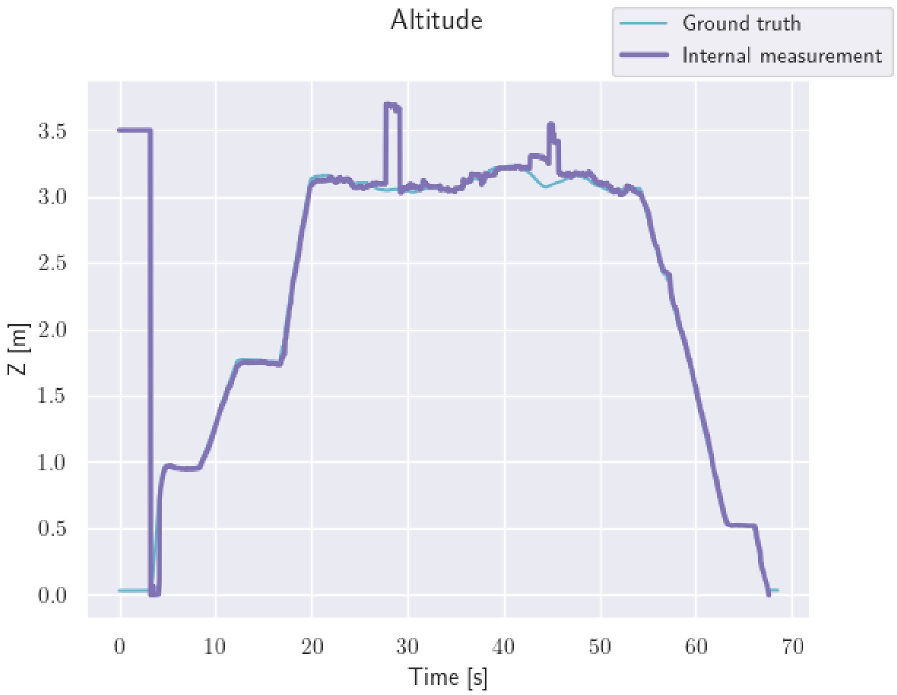


Figure 4.6: Altitude estimate from Anafi drone's internal EKF

GNSS measurement

Without any correction measurements when the platform is not in view in the camera the EKF would discretely integrate the velocity estimates in the Constant Velocity model. This is known as dead reckoning and will make the uncertainty in the filter grow [24]. In order to reduce the uncertainty when the platform is not in view, GNSS measurements in the body frame are therefore incorporated in the filter. The ROS Olympe interface described in section 4.1.2 publishes the GNSS-locations in a NED frame relative to the initial position of the Anafi drone. Under the assumption that the platform is stationary in the water, by that the ReVolt vessel has some form of dynamical positioning system [24], and that the Anafi drone is initiated at the platform, this GNSS-position can be used as an estimate of the platform position. The first presumption, that the platform is not exactly where it was when it was first seen, can be solved by conducting a search for the platform using a search pattern centered on the initial location of the Anafi drone or by accessing GNSS information from the ReVolt. However, this is outside the scope of this thesis.

The GNSS-position of the Anafi drone, given in a NED-frame, have to be converted to a vector describing the position of the platform given in the body frame to be able to use directly in the EKF. The transform publisher ROS node does this transformation and publishes it on another topic. The transformation from the NED-to-body vector, $\mathbf{t}^n_{n\rightarrow b}$, given in the NED frame to the body-to-NED vector, $\mathbf{t}^b_{b\rightarrow n}$, given in the body frame is as follows

$$\mathbf{t}^b_{b\rightarrow n} = -(\mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi}\mathbf{t}^n_{n\rightarrow b}), \tag{4.7}$$

where the $\theta, \phi, and \psi$ describe the rotation of the Anafi drone around the x, y, and z-axis of the world frame respectively.

To be able to test this at the drone lab at NTNU the Qualisys Motion Tracker output is used to simulate GNSS-measurements. The measurement of the drone position is converted to longitude, latitude and altitude, downsampled to 1Hz, and added with white Gaussian noise. Furthermore, the ground truth NED frame at the drone lab is rotated to have the north direction in the same direction as what the Anafi drone reports as north.

Measurement matrices

The correction measurements come into the EKF at different times, and hence they need to have individual measurement matrices **H** to tell which of the states they should correct. The measurement matrices are as follows

$$\mathbf{H}_{AprilTags} = \mathbf{H}_{DNN} = \mathbf{H}_{GNSS} = \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \tag{4.8}$$

$$\mathbf{H}_{Velocity} = \begin{bmatrix} \mathbf{0}_{3x3} & \mathbf{I}_{3x3} \end{bmatrix} \tag{4.9}$$

$$\mathbf{H}_{altitude} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.10}$$

Tuning parameters

The tuning parameters of the EKF are displayed in table 4.5. The tuning is based on empirical testing, and the measurement variances are based on their accuracy during testing.

	Initial values					
	x	y	z	v_x	v_y	v_z
x_0	0.0	0.0	0.0	0.0	0.0	0.0
	Initial variance					
	σ_x	σ_y	σ_z	$\sigma_{v,x}$	$\sigma_{v,y}$	$\sigma_{v,z}$
P_0	0.01	0.01	0.01	0.01	0.01	0.01
	Model uncertainty					
	σ_x	σ_y	σ_z	$\sigma_{v,x}$	$\sigma_{v,y}$	$\sigma_{v,z}$
CV-model	0.1	0.1	0.1	0.1	0.1	0.1
	Measurement uncertainty					
	σ_x	σ_y	σ_z	$\sigma_{v,x}$	$\sigma_{v,y}$	$\sigma_{v,z}$
Velocity	-	-	-	0.01	0.01	0.01
GNSS	2	2	2 / 4	-	-	-
Altitude	-	-	0.1	-	-	-
AprilTag	0.2	0.2	0.2	-	-	-
DNN-CV	0.3	0.3	0.7	-	-	-

Table 4.5: Kalman filter tuning. Note that the z-component for the GNSS-measurement is for lab / outside respectively.

Overall system architecture

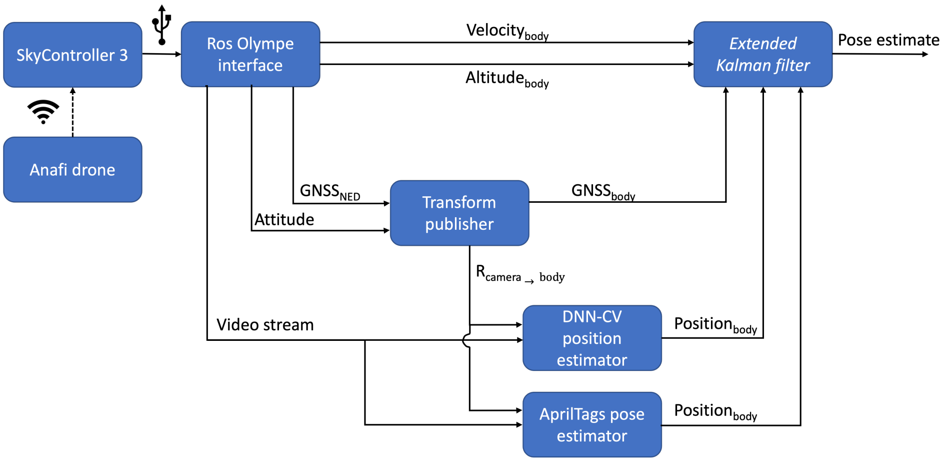


Figure 4.7: Overall system architecture

The overall system architecture is displayed in figure 4.7. In order to receive a stronger signal, the drone is communicating with the ROS Olympe interface via the SkyController 3 as described in section 3.2. Additionally, the ROS software system has four main nodes. The GNSS measurement is published in the body frame, together with all other transformations by the transform publisher node. When the platform is visible, the AprilTag pose estimator and the DNN-CV position estimator both publish their estimates. Finally, using a CV-model, the EKF node fuses the two estimates with the GNSS signal, the velocity estimate, and, during the landing phase, the altitude measurement.

4.8 Post-process

All data processing is carried out in post-processing, as stated in 4.1.1. The author of [1] offered a bash script for adding certain ROS topics to rosbags. Furthermore, using the *bagpy*[9] python library, the topics were extracted, and the various signals were compared and evaluated in post-processing. The *bagpy* package allows users to extract every topic from a rosbag and save them as timestamped.csv files. The data series underwent additional analysis using pandas dataframes[10], and visualization was carried out using the Python tools matplotlib[11] and seaborn[12].

---

[9] `https://jmscslgroup.github.io/bagpy/index.html`

[10] `https://pandas.pydata.org/`

[11] `https://matplotlib.org/`

[12] `https://seaborn.pydata.org/`

Chapter 5

Results

This section contains the result from the different pose estimation algorithms developed as well as the result from the camera calibration. When evaluating the performance of the pose estimation, every module will be presented in its own section and the performance is evaluated on the same dataset. Finally, a test of the system in an outdoor environment on land will be presented.

5.1 Camera calibration

The result of the camera calibration is the camera matrix **K** as shown in (5.1). In comparison to the matrix Falang used in his work, this one is significantly different. The pipeline that the Falang employed had a number of flaws, including failing to provide the dimensions of the checkerboard squares and interchanging the image's width and height. The matrix **K** in 5.1 was generated by the MATLAB camera calibrator program. Visual confirmation reveals that the camera's focal length is close to accurate by analyzing the camera pose image supplied by the application in figure 5.1.

$$\mathbf{K} = \begin{bmatrix} 928.4323 & 0 & 645.9899 \\ 0 & 937.3226 & 364.8657 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

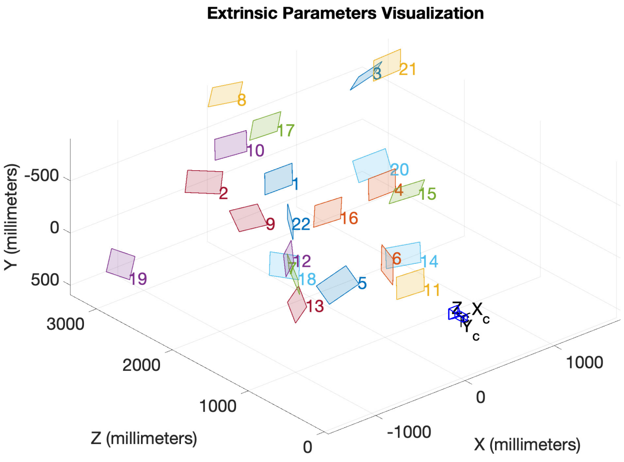


Figure 5.1: Camera pose in the camera calibration for a subset of the calibration images

Furthermore, the reprojection errors of the calibration phase have a mean reprojection error of 0.25 pixels as shown in figure 5.2. This reprojection error points to the fact that the calibration is sufficiently accurate given the sensor size of 1280x720 pixels. An image of the detected points with the reprojected points can be seen in figure 5.3.

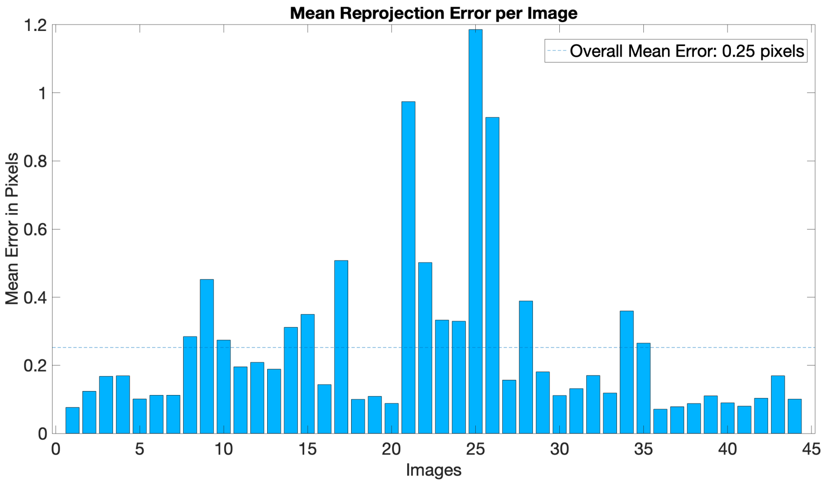


Figure 5.2: Camera calibration reprojection errors

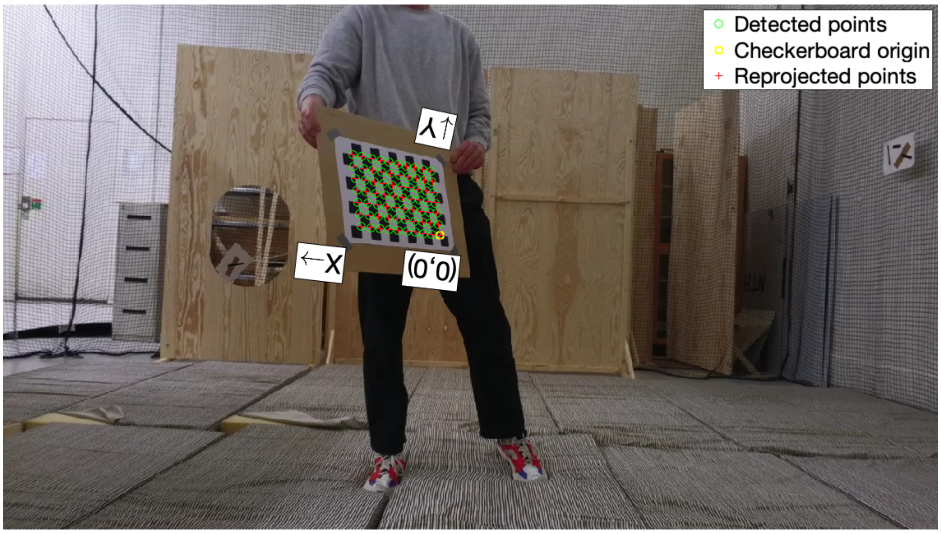


Figure 5.3: Detections and reprojected points from camera calibration

5.2 AprilTag pose estimator

Detection altitude of the AprilTags

Figure 5.4 shows the detection performance and the number of AprilTags detected at different altitudes. The Anafi drone is flown outside at a slowly varying altitude between 0m and 14m, with only vertical movement. The figure shows that the AprilTag-based pose estimator is able to detect 1-5 tags at altitudes in the interval [0.18m, 6.3m]. At higher altitudes, it detects the two larger AprilTags close to the border of the platform. The detection of the two larger AprilTags when the drone is at 6.3m is shown in figure 5.5.

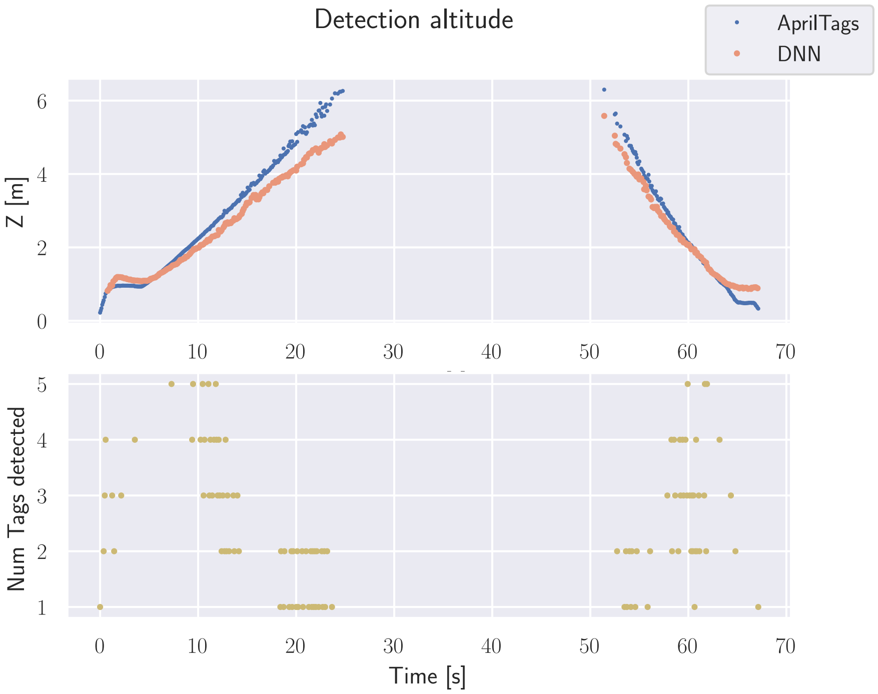


Figure 5.4: Detection altitudes of the AprilTags and DNN-CV estimator

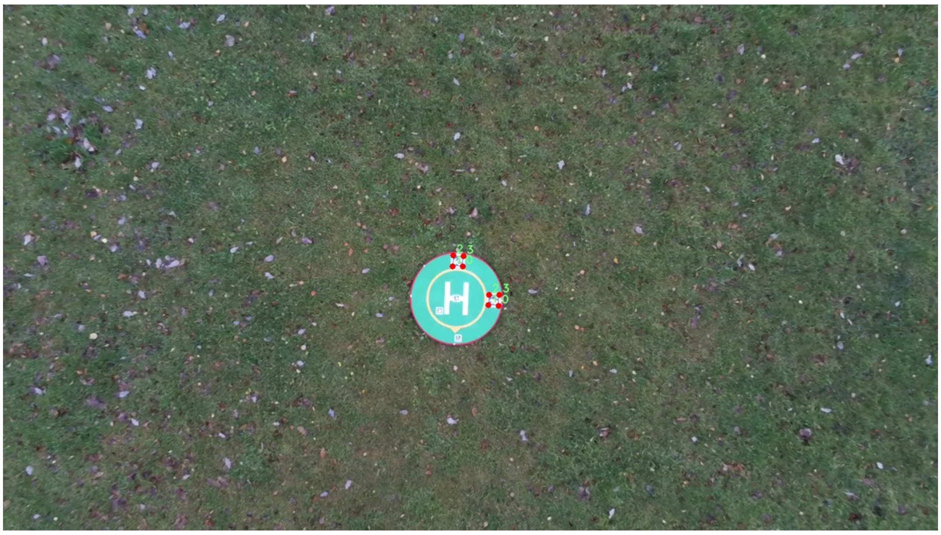


Figure 5.5: Detection of the apriltag corners when far away from the platform. The red dots indicate corner detections.

On the other hand, when the Anafi drone is close to the platform, it estimates the pose down to an altitude of 18cm. This is depicted in figure 5.6. The corresponding output from the camera when the drone is close to the platform at 18cm is shown in figure 5.7. It is now only the center AprilTag that is detected.

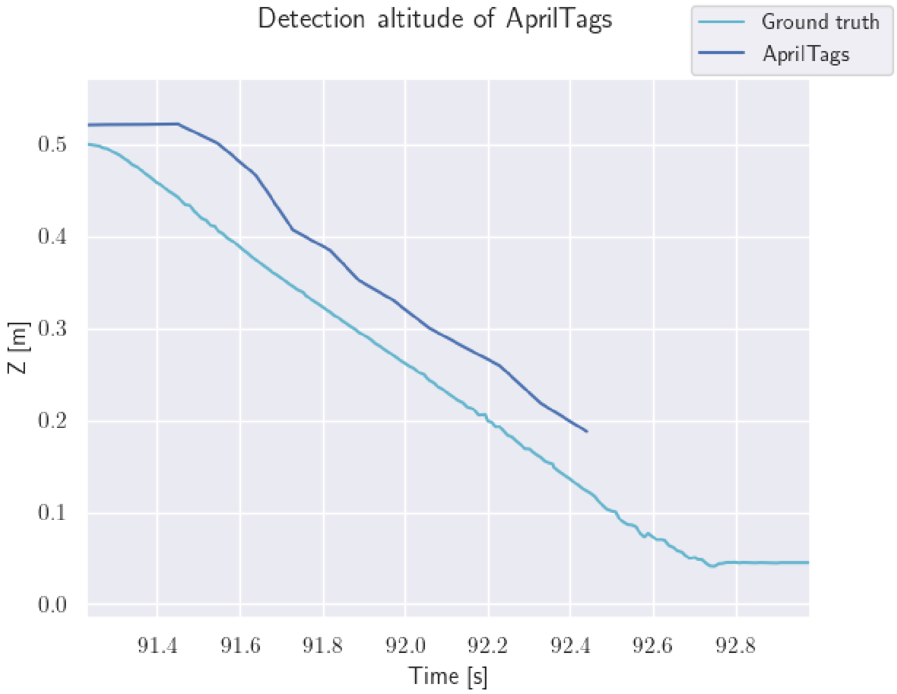


Figure 5.6: Detection of the apriltag corners when close to the platform

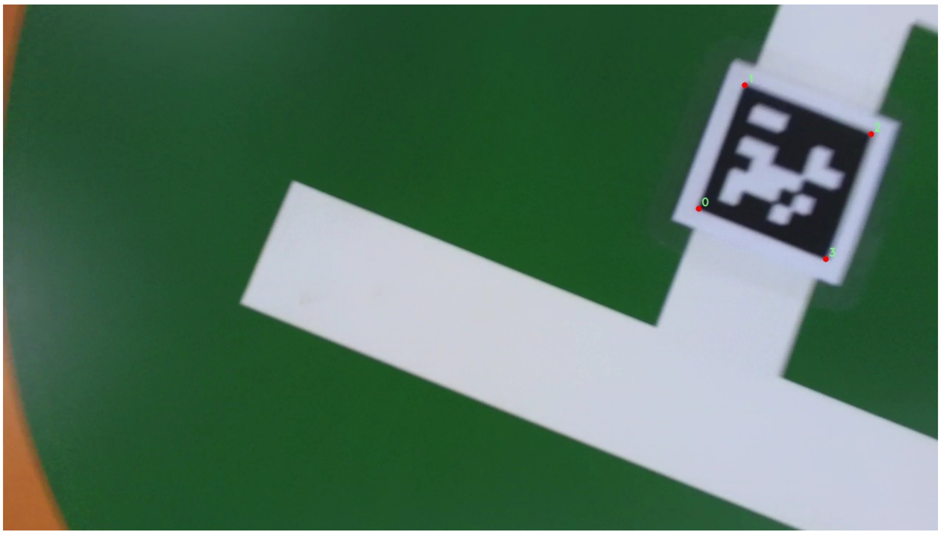


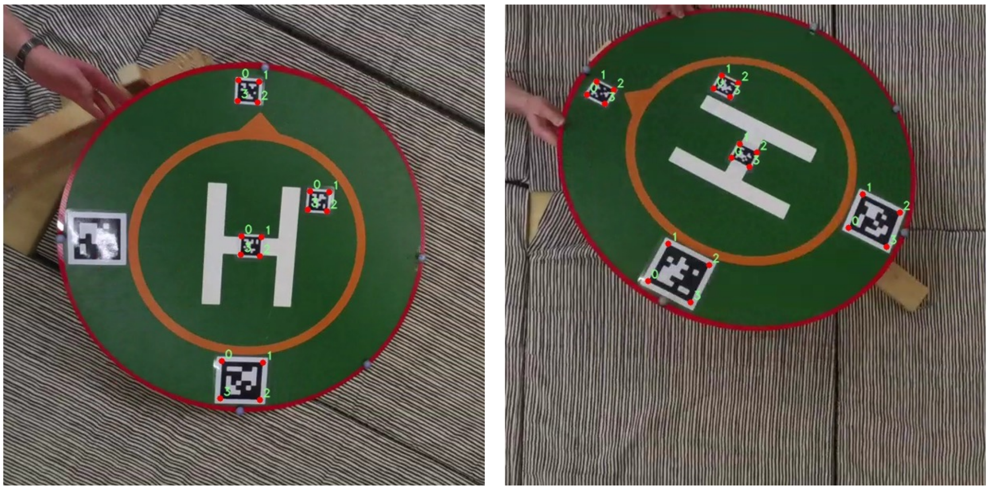
Figure 5.7: Detection performance of the AprilTags at low altitudes. The red dots indicate corner detections.

Detection robustness

Figure 5.8a shows that the detection of one of the AprilTags is lost when a lot of light hits the AprilTag directly from the lamps in the ceiling of the room. The

AprilTags are laminated with glossy plastic, eventually reflecting all light from the ceiling lamps into the camera when the platform is rotated at such an angle. Even though one of the Apriltags is not detected, all other AprilTags are detected, and pose estimation is not lost.

In figure 5.8b, the platform is proposed for manual movement to simulate waves in the ocean. The platform in the figure is oriented at an extreme angle of ≈ 45 degrees around the x-axis of the platform. All AprilTags are still detected.



(a) AprilTag detection performance under varying light conditions

(b) AprilTag detection performance under roll platform motion

Figure 5.8

Pose estimation performance

The mean runtime of one iteration of the pose estimation pipeline, its standard deviation, and theoretical frequency can be seen in table 5.2. The pose estimator has a median runtime of 31 ms, and a mean runtime of 32 ms, with a standard deviation of 15 ms. The maximal theoretical image frequency it can handle before the estimator is the bottleneck is close to 31 Hz. Furthermore, the results of the AprilTag-based pose estimator can be seen in figure 5.9. The figure shows that the estimator is able to estimate the position of the platform well. The estimator has the fewest estimates when the drone is moving quickly in the horizontal XY plane. Nevertheless, it manages to produce some estimates even though there are quick motions and possible motion blur in the images. The estimator is able to produce good results when the drone is doing an opposite motion to cancel out the speed in one direction. This is due to the rotational compensation in the transformation from the camera frame of reference to the body frame of reference. The Root Mean Square Error (RMSE) error of the AprilTag-based pose estimator is shown in 5.1. The RMSE error is comparable for the X and Y axis at 7.04 cm and 6.05

cm respectively. The RMSE error on the Z axis is the lowest at 5.72 cm. The mean RMSE error along all three axes is 6.27 cm.

	RMSE for AprilTag based pose estimator
X	7.04 cm
Y	6.05 cm
Z	5.72 cm
Total	6.27 cm

Table 5.1: Apriltags based pose estimator RMSE error

	AprilTag based pose estimator
Mean runtime	32 ms
Median runtime	31 ms
Standard deviation runtime	15 ms
Theoretical frequency	30.93 Hz

Table 5.2: Apriltags based pose estimator performance metrics

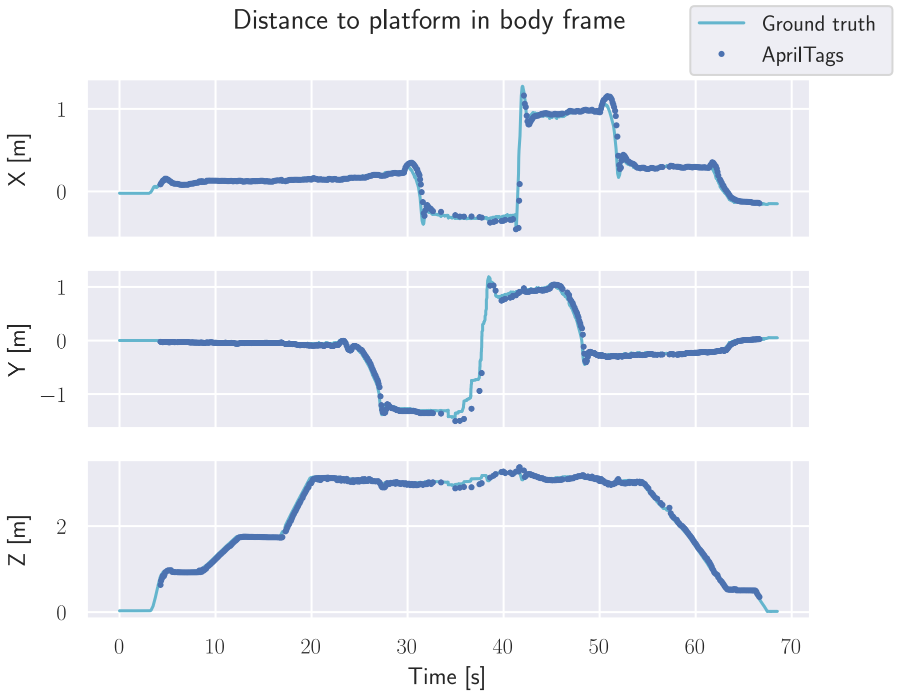


Figure 5.9: Results from the Apriltag based pose estimator

5.3 DNN-CV

Figure 5.4 in the previous section shows the detector performance from various altitudes. In this figure the class probability threshold for the bounding boxes to be used to estimate the position was set to 99%. However, it was tested whether lower class probability thresholds lead to detections from larger altitudes. The result was comparable for different class probability thresholds, and the detection range of the DNN-CV position estimator for the platform is in the interval [0.7m, 5m].

Figure 5.10 depicts the result from running the DNN-CV estimator on the same dataset as for the AprilTag-based estimator. The detection threshold for the bounding boxes to be used in the position estimator is set to 99 %. This is in order to filter out false detections, and only use bounding boxes for detections with a high probability of correct detection. The estimator is able to follow the general shape of the ground truth well but suffers from an offset. Furthermore, the RMSE error of the DNN-CV position estimator is shown in table 5.3. The RMSE is 11.43 cm, 10.20 cm, and 27.62 cm along the x, y, and z-axes respectively. The mean RMSE error along all three axes is 16.41 cm. In figure 5.11 the output from the YOLO is shown when the drone is flying at a high altitude. The bounding box encapsulating the platform is not encapsulating the platform perfectly.

	RMSE for DNN-CV position estimator
X	11.43 cm
Y	10.20 cm
Z	27.62 cm
Total	16.41 cm

Table 5.3: DNN-CV estimator RMSE error

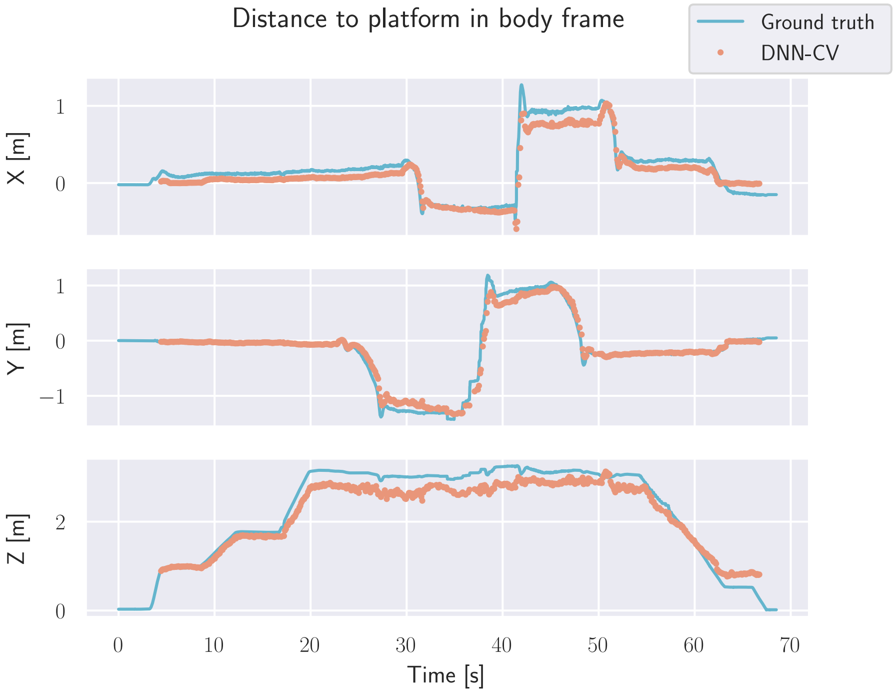


Figure 5.10: Results from the DNN-CV position estimator

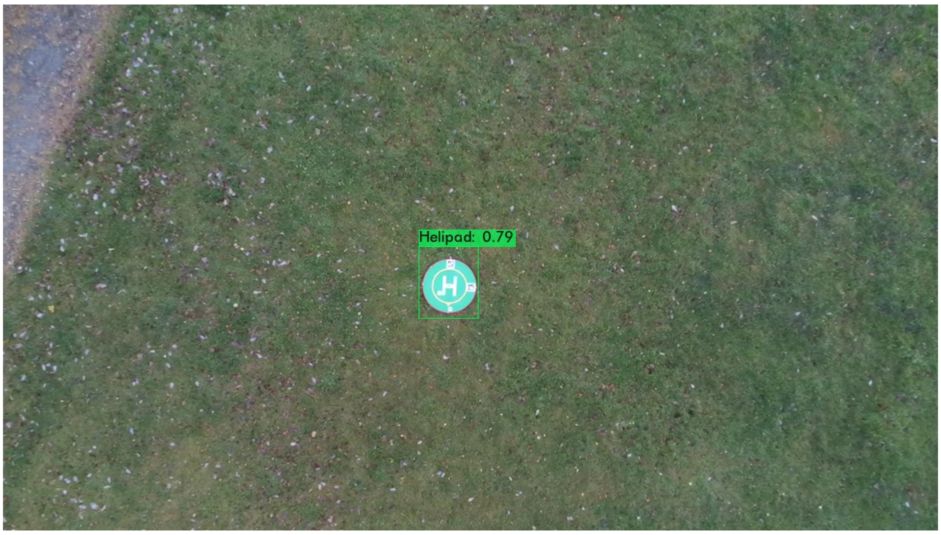


Figure 5.11: Output from the YOLO network at high altitudes

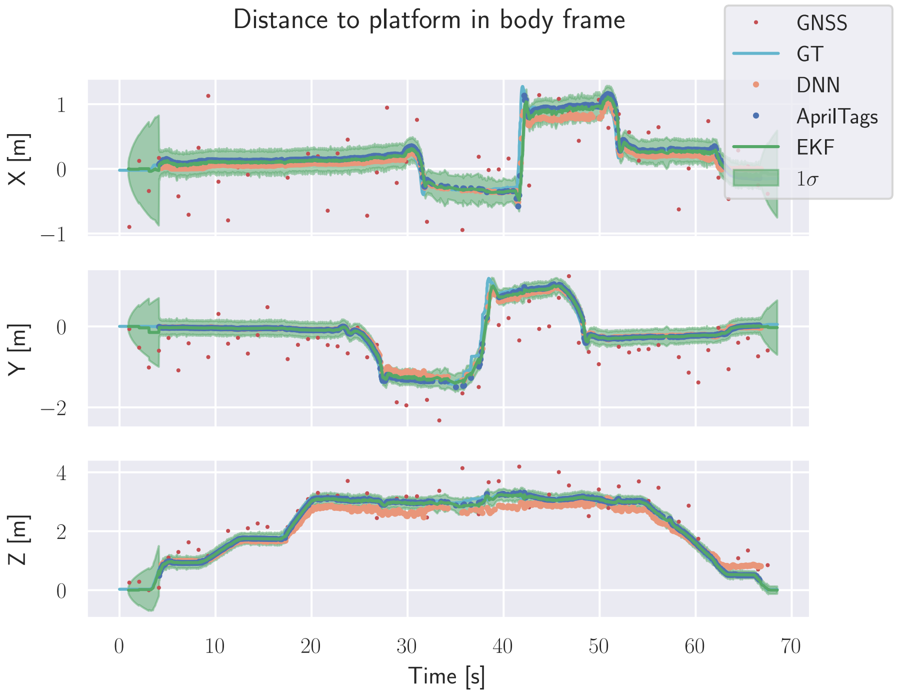
5.4 Kalman filter

Filter with all correction measurement

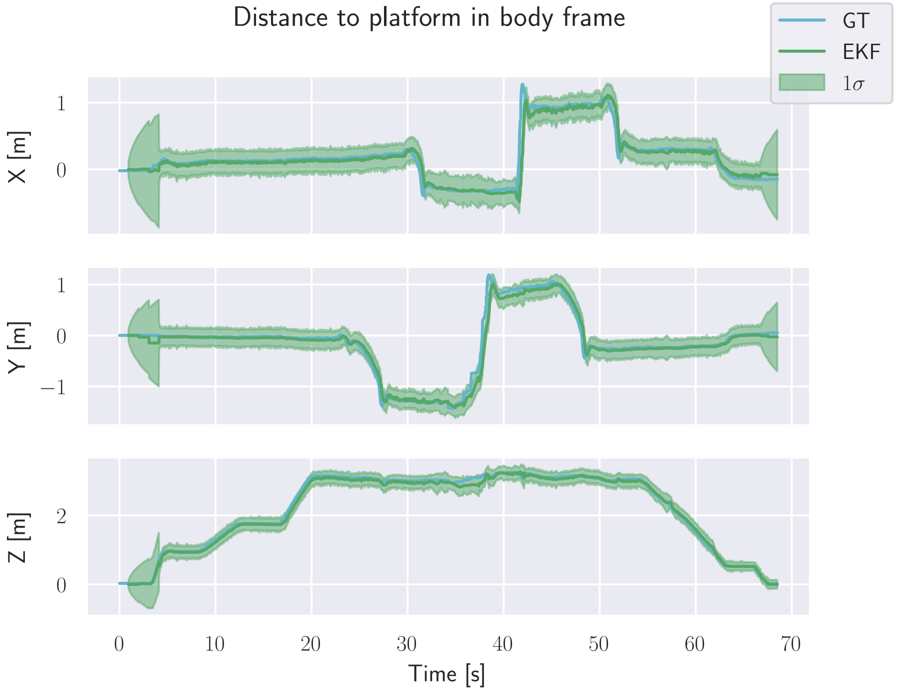
The result of running the EKF with all five correction measurements can be seen in figure 5.12a and 5.12b. Figure 5.12b depicts only the EKF output, its standard deviation, and the ground truth when all corrective measurements are used. In Figure 5.12a, all correction measurements except the altitude correction in the landing phase are also visualized. The EKF manages to follow the ground truth really well. The standard deviation in the EKF output is at its largest when the filter has to rely on only the constant velocity model as in the start and in the end. The RMSE error is shown in table 5.4. The RMSE is 11.37 cm, 8.98 cm, and 6.22 cm in the X, Y, and Z-axis respectively. The mean error along all axis is 8.86 cm. The error is largest in the X-axis due to this being the axis with the fastest position alteration, and generally, the error is most significant when the drone is under heavy acceleration in the horizontal plane highlighted for the x-axis in figure 5.13.

	RMSE for Kalman filter all corrections
X	11.37 cm
Y	8.98 cm
Z	6.22 cm
Total	8.86 cm

Table 5.4: EKF RMSE error with all correction measurements



(a) All signals displayed



(b) Only EKF and ground truth displayed

Figure 5.12: EKF output - corrected with all measurements

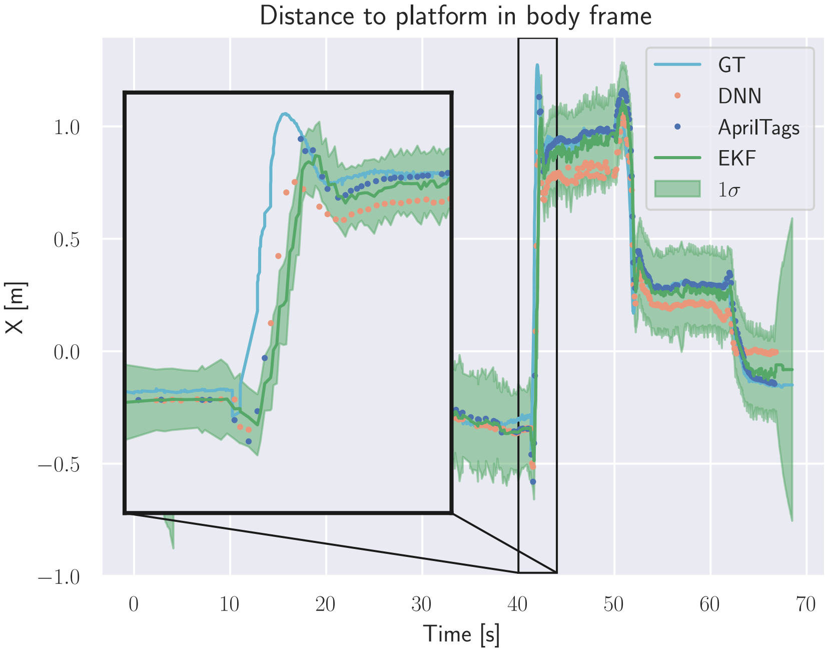
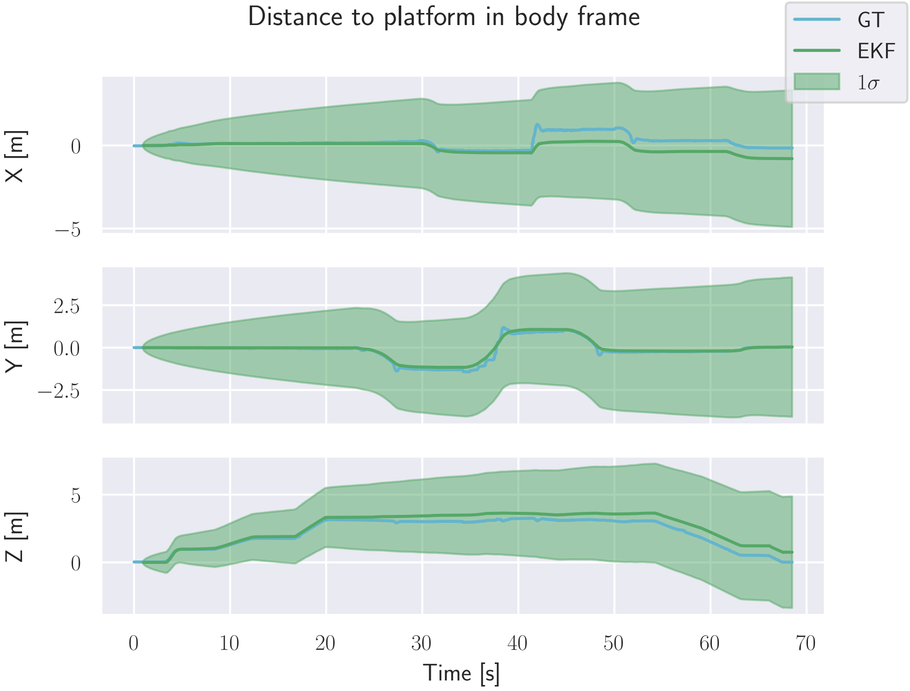
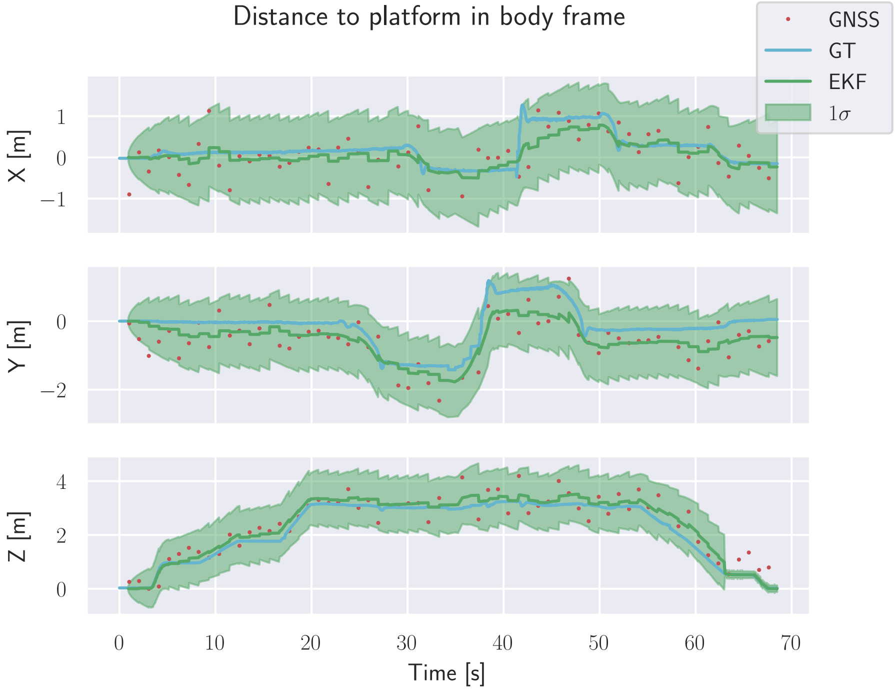


Figure 5.13: Zoomed view of EKF x-axis output with all measurements under fast drone movement

The result of the EKF filter when neither the AprilTag pose estimator nor DNN-CV position estimator is used as corrections is shown in figure 5.14a and 5.14b. In figure 5.14a the filter has to rely completely on the CV-model, and the filter is only corrected with velocity measurements. The filter is very prone to errors as it is dead reckoning, and integrating up the velocity to estimate position. The filter uncertainty is shown to grow throughout the whole flight. On the other hand, when the filter is corrected with the simulated GNSS-measurements the filter uncertainty is kept at a much lower value, and the filter is less prone to errors. This is shown in figure 5.14b.



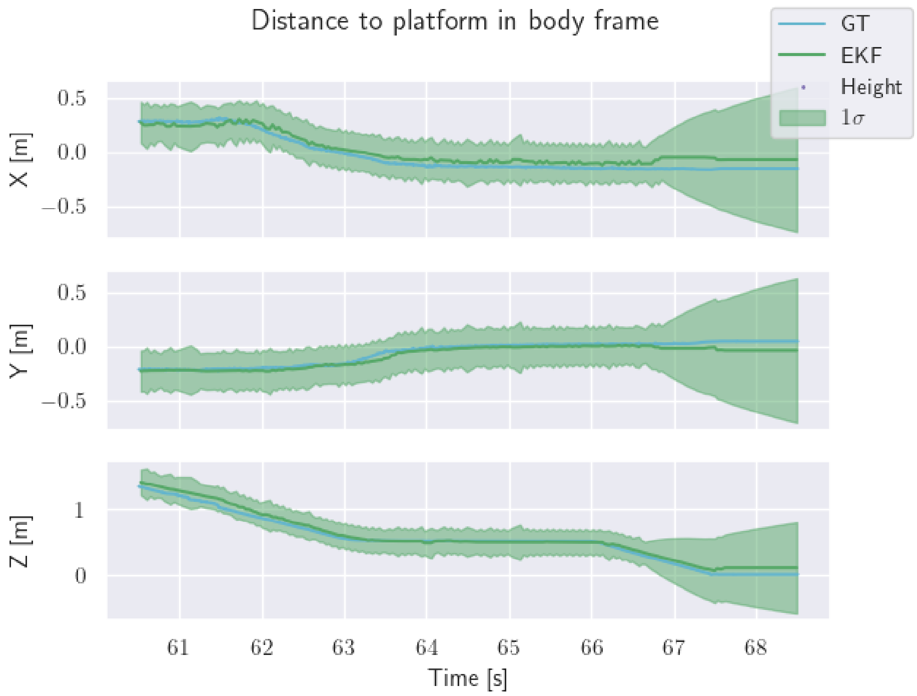
(a) EKF - no corrective measurements



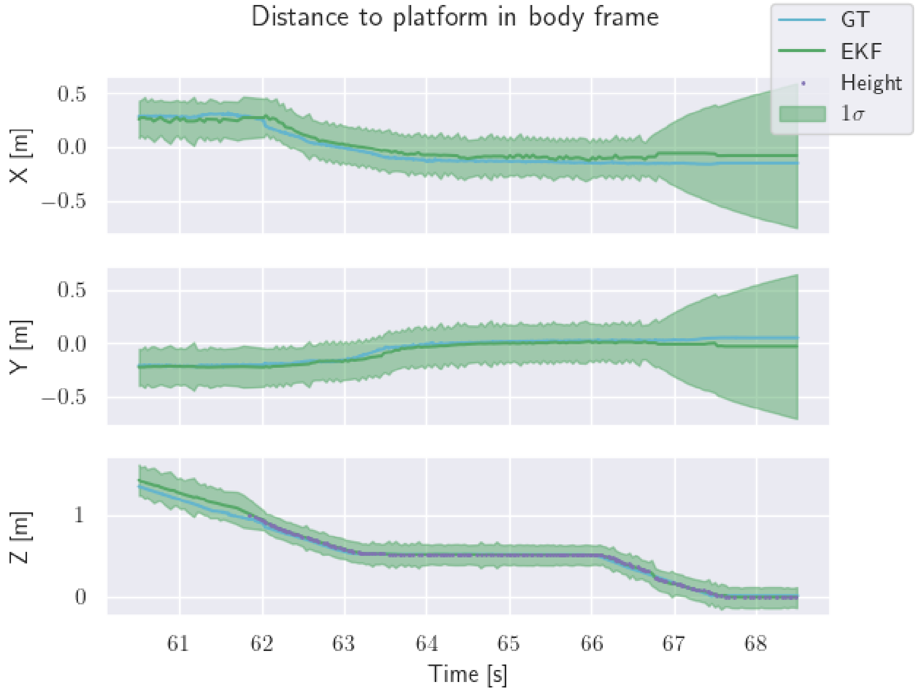
(b) EKF - only corrections from GNSS

Figure 5.14: EKF output - limited correction measurements

Landing phase without internal altitude measurement



(a) Without internal altitude corrections



(b) With internal altitude corrections

Figure 5.15: EKF output in landing phase

Figures 5.15a and 5.15b show the EKF filter output when the internal altitude measurement is not used and used, respectively. In figure 5.15a the uncertainty grows in the z-axis in the last phase of the landing, below 20 cm. This is due to the fact that the filter at that point doesn't have any other correction measurement than the velocity measurement. It, therefore, has to rely completely on the CV model. On the other hand, in figure 5.15b when the internal altitude measurement is used as correction in the landing phase the uncertainty in the z-axis remains low, and the performance is upheld.

Mission away from platform

In order to verify that the filter is able to determine the position of the platform when the platform is not in plain view in the camera, a simulated mission is performed. The Anafi drone is started stationary on the platform in the corner of the drone lab. The drone takes off and increases its height to 2 meters above the platform before it flies away from the platform. The EKF can in this situation no longer rely on the precise corrective measurements from either the AprilTag pose estimator or the DNN-CV position estimator. The drone is then flown back to the platform, and the two estimators can again produce precise estimates. This mission is shown in figure 5.16. The uncertainty in the filter is low when the platform is in view, and high when the filter has to rely on the CV model corrected with the GNSS measurement when the platform is not in the view of the camera.

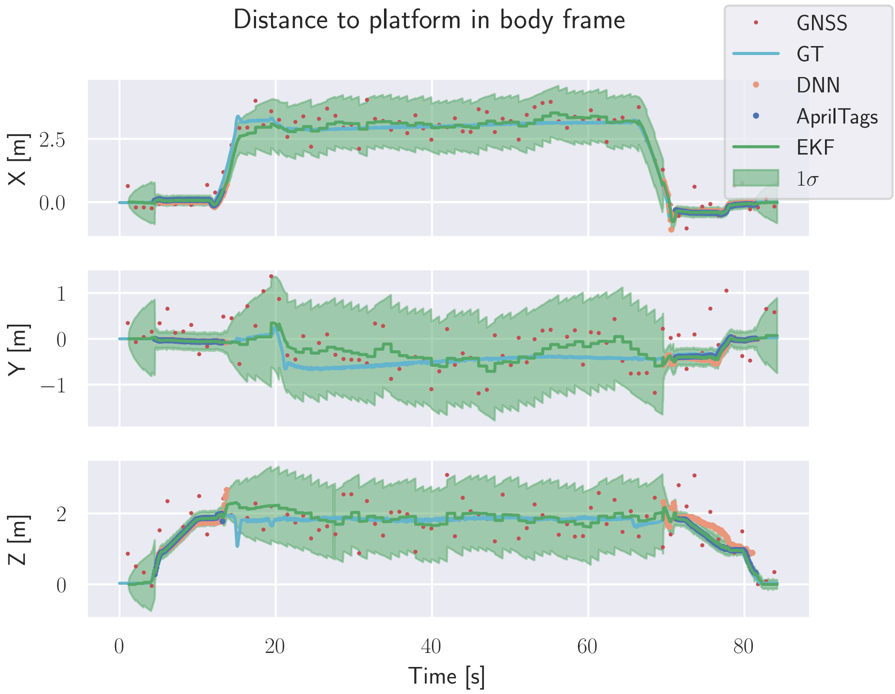
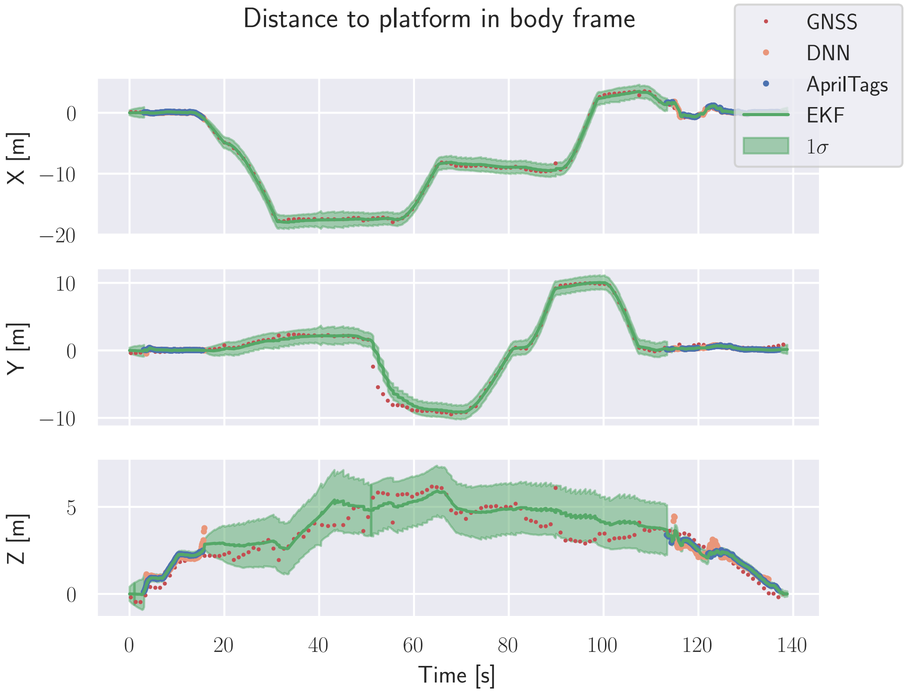
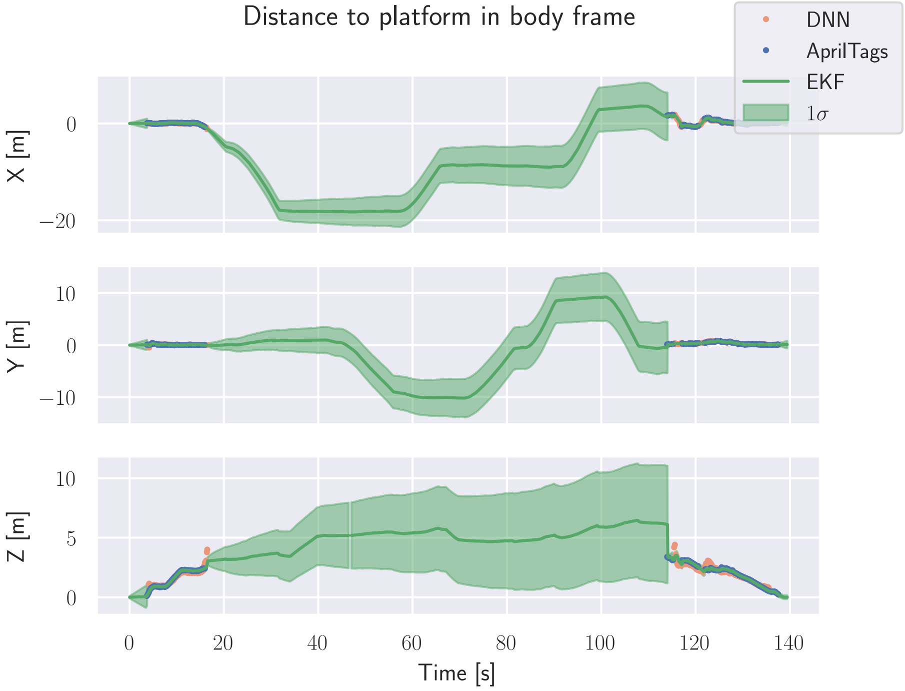


Figure 5.16: EKF outputs from mission in the lab

5.5 Outside testing



(a) EKF with all measurements



(b) EKF with all measurements except GNSS

Figure 5.17: EKF outputs from mission outside

In order to test the developed algorithms at a larger scale the Anafi drone was flown outside. The weather conditions were cloudy, with little wind present and a temperature of 5 °C. The Anafi drone performs similar actions to the ones performed and discussed in section 5.4. The only difference this time is that the Anafi drone is no longer limited by the limited space of the drone lab and that real GNSS-measurement is available. The Anafi drone takes off from the platform, flies away from it, and performs different motions, before flying back. In figure 5.17a all measurements are used to correct, including the transformed GNSS measurements. The drift and covariance along all axes are low during the flight. The axis suffering the most from not having the platform in view is the z-axis where the estimate is the least smooth. The filter estimate will also suffer if the GNSS measurement is inaccurate in the z-axis and delivers an outlier measurement at the timesteps shown in figure 5.17a at about 55 s. The filter estimate is "pulled" a little down due to the outlier measurement. In figure 5.17b, all measurements except the GNSS measurements are used to correct the filter. The uncertainty is growing along all axis. However, the filter seems to be providing smooth estimates in the x and y axis with low to no drift. The same thing cannot be observed on the z-axis. The estimate in this direction drifts upwards and has a large spike downwards when the platform again is in view and the two perception-based estimators provide more accurate estimates.

Chapter 6

Discussion

This section discusses the results of the camera calibration as well as the result of testing the different pose estimators, and the full EKF solution.

6.1 Camera calibration

The result of the re-calibration of the camera is as expected. The found image centers $o_x = 645.9899$ and $o_y = 364.8657$ is much closer to the expected image center one would get from a perfectly aligned optical axis. If the camera was perfect the center would have been at half the image height and width at $o_x = 640$ and $o_y = 360$. Furthermore, the mean reprojection error is low at 0.25 pixels. This is also a good indicator that the calibration is good. Even though the mean reprojection error is low, the camera calibration can still be improved. As the authors of [47] point out, a person novice to calibrating cameras may not provide a perfectly good training set for calibration. The camera calibration can therefore maybe be improved by for instance using an interactive calibration pipeline with live feedback such as *AprilCal* presented in [47].

In this context is it worth mentioning that the offset problem Falang faced with the TCV-module proposed in [1] was a result of highly incorrect camera calibration. Falang used a camera matrix with a completely wrong image center and focal length, resulting in the offsets in the pose estimates from the TCV-module being a function of the detection altitude. According to some further testing of the TCV-module using the newly acquired camera matrix $\mathbf{K}$, the module calculated pose estimates more accurately. However, it is still very computationally and time intensive and hardly ever generates any estimates when there is a lot of noise in the images or extensive rotations of the platform.

6.2 AprilTag-based pose estimation

The AprilTag pose estimator worked very well. It has an overall low run time, enabling fast processing of the incoming images from the drone, and produces pose estimates with good accuracy across all three axes. The pose estimator can produce good estimates down to about 20 cm over the platform. The estimator is also robust enough to still produce good estimates when the platform is subjected to motions, such as a roll motion.

Nevertheless, some problems have to be addressed

- The module is ultimately dependent on the detection of the AprilTags. No detections of the AprilTags at altitudes larger than 6.3 meters.
- Difficult light conditions make it harder to detect the markers due to their glossy texture.
- Few estimates under quick attitude changes.

The detection capabilities of the detector are starting to fall off when the altitude is very large, at altitudes larger than 6.3 meters. This is because the larger Apriltags on the platform no longer can be seen. Nevertheless, based on figure 5.4, it is clear that introducing AprilTags at different sizes serves its purpose. At large altitudes, the two large AprilTags are detected, whereas at lower altitudes when the Anafi drone is closer to the center of the platform the smaller ones are detected. To increase the detection performance at larger altitudes, the size of some of the AprilTags could be increased or a very large AprilTag could be introduced covering most of the platform surface. This was not done in this work due to not interfering with the detection capabilities of the YOLO detector. The YOLO detector is trained on images of the platform without the AprilTags, and by covering up possible features of the platform, the YOLO detector may get reduced detection performance.

The glossy texture of the AprilTags makes them hard to detect when the light is reflected directly off them and into the camera. This rarely occurred at the drone lab, but it is expected to be a larger problem outside in sunny conditions. This problem can easily be solved by for instance using matte lamination sheets. An even better solution is by re-manufacturing the platform with the AprilTags printed on it. This will also allow for reducing the uncertainty in the precise metric position of the AprilTags corners. Even though the metric positions of the corners are precisely measured with the help of the Qualisys Motion Tracker system, small offsets from the measured values are expected as the AprilTags are taped to the platform.

From figure 5.9 is it clear that where the estimator struggles the most is where the drone suddenly changes attitude. The drone conducts a square motion while being given impulse-like directions in roll, pitch, and yaw in the dataset where the estimator is tested. When the drone receives such prompt commands, its attitude is quickly altered and hence the drone accelerates fast. The drone's gimbal compensates for rotation, and motion is externally imposed on the camera. Again, due to these vibrations, the images are blurry, making it difficult for the detector to pick up the AprilTags. However, the estimator continues to generate accurate estimates when the drone reaches the end of a corner and the gimbal re-stabilizes. Additionally, it should be highlighted that such quick movements as the drone conducts in this data set, where its velocity approaches ±0.8 m/s, are impossible during a controlled landing and these errors should not be of great concern given the use-case of the perception pipeline.

Despite these issues, the new estimator built on AprilTag detection outperforms the TCV strategy Falang presented in [1]. The RMSE from replicating movements akin to those made by Falang is reduced from 17.58 cm to 6.27 cm, more than halving. The new AprilTags-based pose estimator also has a substantially smaller processing need, allowing it to produce estimates faster. The new AprilTag estimator is able to produce estimates close to 31 Hz, whereas the TCV technique created by Falang output estimates at approximately 1.5 Hz from the indoor lab tests. The TCV approach had even worse performance and provided estimates at 0.5 Hz when tested outdoors at sea with roll movements imposed on the platform. Even though the AprilTag pose estimator has not yet been tested on the sea, the introduction of extreme roll motions of 45 degrees side-to-side movements in the lab environment indicated that the AprilTag pose estimator could still provide estimates. Finally, at lower altitudes, the AprilTag-based pose estimator outperforms the TCV approach in terms of detection capability. The TCV approaches' detection capacities declined abruptly at lower altitudes because the platform's features could no longer be detected, whereas the AprilTags can be detected down to altitudes of about 20 cm. By including AprilTags of varying sizes, the detection performance at lower and higher altitudes can be improved even further.

6.3 Deep neural-network-based position estimation

Whereas the author of [1] found the DNN-based approach to be the best estimator is it this time turned the other way around. The DNN-based position estimator does not produce as good estimates as the AprilTag-based pose estimator. Nevertheless, it shows potential. It manages to give estimates at a good frequency and due to the new rotational compensation from the camera frame to the body frame, it manages to estimate the position well under heavy rotations of the drone body. Furthermore, the estimation capabilities of the DNN-CV estimator are best when the drone is performing slow movements and is under little acceleration in the same way as the AprilTag-based pose estimator. However, the DNN-CV estimator seems to produce more estimates in situations with larger accelerations than the AprilTag-based estimator.

As the DNN-based position estimator is largely based on the same approach as the one used in [1] it suffers from the same problems.

- Offsets due to incorrect bounding box sizes.
- Detection performance falls off at altitudes larger than 5 meters.
- Bad position estimate capabilities at lower altitudes.

When the Anafi drone is flying at higher altitudes the output from the YOLO network is not able to produce bounding boxes that perfectly encapsulate the platform. As the algorithm used to calculate the position from the bounding boxes (Algorithm 1) evaluates the ratio between the pixel radius and the metric radius of the platform to estimate the altitude, any margins around the bounding boxes to the platform will lead to error in the position estimate. It is evident from figures 5.10 and 5.11 that this is the case. The YOLO detector produces larger bounding boxes than it should, and hence the estimate in the z direction will be lower than it should. This error will also propagate into the estimates in the x and y direction, as these are based on similar triangles where the z estimate is used. The network could be retrained to reduce the offset in the bounding boxes from the YOLO network at higher altitudes. The current network is a YOLOv4 network trained by the author of [2], and the images used in the training were from the camera of the previously used Parrot AR2.0 drone. Although the images coming from the Anafi drone have excellent resolution, the network is trained on the lower resolution images from the AR2.0 drone, hence the detection will not benefit from the Anafi's resolution. The detection performance should consequently be better at higher altitudes if a newer network is trained on images with better resolution. Newer versions of YOLO have also been made available since the training of the network, where the newest and fastest network is the YOLOv7, reporting both better inference speed and accuracy than older versions [48]. By retraining the network, the probability of false detections will also most likely be reduced. The problem that the algorithm produces incorrect estimates at lower altitudes is a result of the algorithm's requirement to see the whole platform in the image for it to be able to calculate the pixel-to-metric radius ratio.

6.4 Kalman filter

The Extended Kalman Filter was able to fuse the two estimates from AprilTags pose estimator and the DNN-CV position estimator together with corrective measurements from the Anafi drone in a constant velocity (CV) model. The RMSE is low, and hence the filter can be said to provide estimates at a high rate with good precision. In situations with fast accelerations in the horizontal plane, as mentioned earlier, the AprilTag-based estimator provides few to no estimates and the filter has to rely on the few corrective measurements from the DNN-CV-based estimator. As the DNN-CV estimator suffers from an offset, the corrective measures are not able to "pull" the EKF estimate as far up as it should. In addition, the velocity measurement is updated too slowly at 5Hz to capture the drone's rapid movement and consequently estimate the position correctly in situations with high accelerations. The model in the filter is in other words integrating up a too-low velocity. However, as mentioned earlier, it is unlikely that fast accelerations like this will occur. When comparing the RMSE to the RMSE of the other two estimators, it is larger than the RMSE of the AprilTag-based estimator. However, the error of the AprilTag-based estimator is only calculated when the estimator provides an estimate. Hence, at the large accelerations sequences, the AprilTag-based estimator does not contribute with any error to the RMSE whereas the error of the EKF does as it provides an estimate at 25 Hz no matter what.

The inclusion of the corrective GNSS-measurement is the most helpful when the filter does not receive any precise correction measurements from either the Apriltag-based estimator or the DNN-CV estimator. Due to the large measurement covariance given to the GNSS corrections, the measurement will not influence the filter much when the filter has correction measurements from the two much more precise perception-based estimators. On other hand, it has been proven to help the filter not drift too much in cases where the platform can not be seen. The filter can be changed to use only GNSS corrections when the platform cannot be seen and perception-based estimators cannot provide corrections. The output of the EKF may be somewhat non-smooth when flying away from the platform in an indoor lab environment. This is particularly evident when multiple consecutive simulated GNSS measurements are significantly different from the ground truth value in the same direction, causing the estimate to deviate from the ground truth. The simulated GNSS measurements are generated from ground truth data obtained from a motion capture system and augmented with independent, identically distributed random Gaussian noise, which does not accurately reflect the noise characteristics of real GNSS signals. By contrast, when the system is tested outdoors and the EKF uses real GNSS measurements to correct the filter, the output is generally smoother.

The internal altitude measurement has also been proven to be very helpful for the filter in the landing phase when the perception-based estimators no longer can provide any corrections. The landing phase is also the most crucial phase to having good estimates of the altitude in order to know when to turn off the motors.

6.5 Outside testing

The Extended Kalman Filter has been demonstrated to function effectively, even when flying outside. In the horizontal plane, the corrective GNSS estimates are demonstrated to be the most reliable. As GNSS signals' vertical precision is lower than their horizontal accuracy, this is expected [49]. This is compensated for in the filter by setting the covariance of the vertical GNSS measurement larger than for the horizontal when flying outside. However, it might be claimed that this worse vertical accuracy has little bearing on the use of this pose estimate approach. When the platform is not in plain view in the camera, the filter primarily uses the GNSS measurements to rectify any drift in the location estimation. Therefore, everything is fine as long as the GNSS measurements are accurate and free of systematic errors and within a suitable range of the true values. When the platform is visible in the camera's field of vision, the correcting GNSS measurements have little impact on the position estimate. Additionally, in the landing phase, where the need for accurate altitude estimates is greatest, the filter uses the corrective altitude measurement from the internal EKF of the Anafi.

It can be argued that the EKF output in the horizontal direction appears to be performing impressively even when flying away from the platform and the GNSS correction is not applied. When no corrections are provided, the estimate's uncertainty increases over the course of the sequence, but when accurate position estimates from camera-based pose estimators are again provided, the estimate from the filter doesn't change quickly. This suggests that even without corrections, the filter estimate does not drift. Unfortunately, this is completely dependent on accurate and reliable velocity readings from the Anafi drone. In the testing conducted outside, the drone's velocity is kept low and there are no abrupt horizontal accelerations. This is to simulate the control commands from the control algorithms developed by Solbø in [4], which do not involve rapid step-response motions. The drone may nevertheless be susceptible to such fast accelerations from disturbances like the wind, even though the control algorithms may not involve them. In these situations, it is possible that the velocity estimations won't be as accurate, and since the EKF relies on dead reckoning in the absence of GNSS corrections, any velocity inaccuracies will affect the position estimate. As a result, the addition of GNSS measurements will improve the EKF's long-term precision and deliver accurate position estimations over a wider time horizon even when the platform is not in plain view in the camera.

Chapter 7

Conclussion and Future work

This chapter will first summarize the findings and results from this thesis. The following section will discuss the further work that has to be done on the perception system before it can be claimed to be capable of aiding in a SAR mission at sea.

7.1 Conclussion

This thesis looked into how to employ two perception-based methodologies combined with additional sensor data in an Extended Kalman Filter to estimate the position of a landing platform. The ultimate objective of this thesis was to determine the location of the platform with accuracy under probable abrasive situations, even when the platform was not visible to the camera. Using a Parrot Anafi drone, the experiments for this thesis were carried out both indoors, in a lab setting, and outside.

Two distinct methodologies served as the foundation for the two camera-based location estimate algorithms that were combined in an EKF. The first one uses feature detection, homography estimation, and extrinsic extraction as part of a more conventional computer vision approach. The platform used in this thesis was altered by adding fiducial markers from the AprilTags system that are easier to detect. This strengthened and sped up the detection pipeline. Different sized AprilTags were used, with the smaller AprilTags being detected at lower altitudes and the larger AprilTags being detected at higher altitudes. It was recommended that in the future, larger AprilTags should be used to improve the capabilities for detection at even greater altitudes. Another issue with the detection pipeline was the inability to identify the impacted AprilTags when direct light was reflected off the AprilTags and into the camera. As suggested, the AprilTags could in the future be laminated with matte sheets or the platform could be remanufactured with the AprilTags printed on it because this reflection was a direct effect of utilizing AprilTags laminated with a glossy texture. Overall, the AprilTags-based pose estimation method was able to generate precise estimates at a high rate even when the platform was proposed for extensive motions.

This thesis also used a DNN-CV strategy for camera-based location estimation, which was based largely on Hove's work in [2]. The only changes made to the algorithm in this thesis were using the new focal length and image center from the new camera calibration matrix, as well as using a new transformation from the camera reference frame to the body reference frame, which accounts for rotations of the Anafi drone. This technique was based on the results of a YOLO network trained by Hove in [2], and a position estimation technique that uses the idea of similar triangles proposed by Sundvoll in [3]. This approach faced the same problems that Falang's system had [1], including offsets and a lack of lower-altitude detection capability. This thesis also discovered that the detection capabilities of this approach started to decline at altitudes greater than 5 meters. In the same way, as Falang proposed in [1], the offset problem was proposed to be overcome by retraining the network using images from the Anafi drone with the new platform with AprilTags on it. A more recent YOLOv7 network that offers quicker inference speed and better detection capabilities while lowering miss-detections was suggested for the re-training. To reduce the issue of false detection, the class probability threshold for the bounding boxes in this thesis was set at 99%.

A Constant Velocity model Extended Kalman Filter, proposed by Falang in [1], was used to fuse the two perception-based methodologies with additional sensor data to estimate the position of the landing platform. Instead of using the newly available optical flow-based velocity measurement at 30 Hz, the filter used the velocity measurements coming from the Anafi drone at 5 Hz. This was due to the fact that the faster optical flow-based measurement was shown to be inaccurate. Additionally, the filter was upgraded to include two more corrective measurements: GNSS measurement and an altitude measurement during the landing phase. In the landing phase, the two perception-based approaches were combined with the altitude measurement to enhance vertical accuracy. The EKF has more global navigational qualities due to the GNSS measurement that provides the x, y, and z coordinates in a NED frame with origin at the take-off position of the Anafi drone. Reducing drift in a dead reckoning approach has been shown to improve the estimator's long-term precision capabilities. It has been demonstrated that the EKF's estimates are largely accurate. However, when the Anafi drone is subjected to rapid acceleration, the EKF is found to be the least accurate. This is because neither of the two perception-based estimators offers very many estimates in these circumstances, and the velocity measurements are also at their least reliable in these conditions. Nevertheless, the EKF provides estimates within reasonable limits in order to perform a controlled landing.

7.2 Future work

This section will go over the requirements the system must meet in order to be put to use in a real-world SAR mission at sea.

In addition to the things mentioned in this thesis with adding more AprilTags with a matte texture and retraining the YOLO-network, there are also other objectives that have to be completed. Even though the position estimator yields good results in open-loop, closed-loop control utilizing the control algorithms developed in Solbø's thesis [4] is unworkable due to the latency problems covered in section 4.4. Therefore, the drone must be upgraded in order for the system to be useable in a scenario when an autonomous landing is required at sea. Close loop control must be supported by the new drone. The most crucial part missing from the developed perception system is the system's ability to detect persons in the water and report their position to a rescue team. The network utilized in the DNN-CV approach, for example, may perform this detection. Retraining this network using images of people floating in the ocean might be an effective strategy. In this context, a search pattern to detect people in the water is also required, and research on the most effective method of doing this must be done. Finally, the drone's perception system should have some kind of environmental awareness because it should be able to operate autonomously without burdening the rescue team. This will allow the drone to determine where it would be safe to land in the event that it detects low battery levels or other issues. Including a segmentation network that designates areas as safe or unsafe could accomplish this, but other approaches may also be usable.

Bibliography

[1] 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.

[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] 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.

[4] Ø. Solbø, “Guidance and control for landing of autonomous multirotor,” O.S. Bragstads Plass 2D, 7034 Trondheim, 2022.

[5] P. Inverardi, “The challenge of human dignity in the era of autonomous systems,” in *Perspectives on Digital Humanism*, H. Werthner, E. Prem, E. A. Lee, and C. Ghezzi, Eds. Cham: Springer International Publishing, 2022, pp. 25–29, ISBN: 978-3-030-86144-5. DOI: `10.1007/978-3-030-86144-5_4`.

[6] M. Tai, “The impact of artificial intelligence on human society and bioethics,” *Tzu Chi Medical Journal*, vol. 32, Jan. 2020. DOI: `10.4103/tcmj.tcmj_71_20`.

[7] V. D. Simone, V. D. Pasquale, V. Giubileo, and S. Miranda, “Human-robot collaboration: An analysis of worker’s performance,” *Procedia Computer Science*, vol. 200, pp. 1540–1549, 2022, 3rd International Conference on Industry 4.0 and Smart Manufacturing, ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2022.01.355`.

[8] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, “Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges,” *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019. DOI: `10.1109/ACCESS.2019.2909530`.

[9] 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,” vol. 621, Sep. 2010.

[10] A. Salagame, S. Govindraj, and S. Omkar, *Practical challenges in landing a uav on a dynamic target*, Feb. 2020. DOI: `10.13140/RG.2.2.12949.65767`.

[11] J. Sun, B. Li, Y. Jiang, and C.-y. Wen, “A camera-based target detection and positioning uav system for search and rescue (sar) purposes,” *Sensors*, vol. 16, no. 11, 2016, ISSN: 1424-8220. DOI: `10.3390/s16111778`.

[12] S. Zhong, Y. Liu, Y. Liu, and C.-S. Li, “Water reflection recognition based on motion blur invariant moments in curvelet space,” *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 22, Jul. 2013. DOI: `10.1109/TIP.2013.2271851`.

[13] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, “Vision-based autonomous quadrotor landing on a moving platform,” in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2017, pp. 200–207. DOI: `10.1109/SSRR.2017.8088164`.

[14] O. Araar, N. Aouf, and I. Vitanov, “Vision based autonomous landing of multirotor uav on moving platform,” *Journal of Intelligent & Robotic Systems*, vol. 85, Feb. 2017. DOI: `10.1007/s10846-016-0399-z`.

[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] Choi, Christopher, “Towards robust autonomous mav landing with a gimbal camera,” M.S. thesis, 2018. [Online]. Available: `http://hdl.handle.net/10012/14192`.

[17] R. Duan, Y. Guo, and P. Lu, “Object pose estimation for uav navigation using an end-to-end lightweight cnn,” in *2021 China Automation Congress (CAC)*, 2021, pp. 2128–2132. DOI: `10.1109/CAC53003.2021.9727784`.

[18] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes,” Jun. 2018. DOI: `10.15607/RSS.2018.XIV.019`.

[19] S. P. Yeong, L. M. King, and S. S. Dol, “A review on marine search and rescue operations using unmanned aerial vehicles,” *International Journal of Marine and Environmental Sciences*, vol. 9, no. 2, pp. 396–399, 2015, ISSN: eISSN: 1307-6892. [Online]. Available: `https://publications.waset.org/vol/98`.

[20] P. Rudol and P. Doherty, “Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery,” in *2008 IEEE Aerospace Conference*, 2008, pp. 1–8. DOI: `10.1109/AERO.2008.4526559`.

[21] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, “Multipurpose uav for search and rescue operations in mountain avalanche events,” *Geomatics, Natural Hazards and Risk*, vol. 8, no. 1, pp. 18–33, 2017. DOI: `10.1080/19475705.2016.1238852`.

[22] S. Sambolek and M. Ivasic-Kos, “Automatic person detection in search and rescue operations using deep cnn detectors,” *IEEE Access*, vol. 9, pp. 37 905–37 922, 2021. DOI: `10.1109/ACCESS.2021.3063681`.

[23] 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`.

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

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

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

[27] C. G. Harris and M. J. Stephens, “A combined corner and edge detector,” in *Alvey Vision Conference*, 1988.

[28] J. Shi and Tomasi, “Good features to track,” in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. DOI: `10.1109/CVPR.1994.323794`.

[29] S. J. K. Pedersen, “Circular hough transform,” Aalborg University, 2007.

[30] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, “Fiducial markers for pose estimation: Overview, applications and experimental comparison of the artag, apriltag, aruco and stag markers,” *Journal of Intelligent & Robotic Systems*, vol. 101, Apr. 2021. DOI: `10.1007/s10846-020-01307-9`.

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

[32] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2e. New York, NY, USA: Springer, 2006.

[33] 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`.

[34] M. A. Nielsen, *Neural networks and deep learning*, misc, 2018. [Online]. Available: `http://neuralnetworksanddeeplearning.com/`.

[35] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. A. Velasco-Hernández, L. Krpalkova, D. Riordan, and J. Walsh, “Deep learning vs. traditional computer vision,” *CoRR*, vol. abs/1910.13796, 2019. arXiv: `1910.13796`.

[36] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: `1506.02640`. [Online]. Available: `http://arxiv.org/abs/1506.02640`.

[37] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: `2004.10934`. [Online]. Available: `https://arxiv.org/abs/2004.10934`.

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

[39] Parrot, *Anafi white paper*, v1.4. [Online]. Available: `https://www.parrot.com/assets/s3fs-public/2020-07/white-paper_anafi-v1.4-en.pdf`.

[40] *Aprilrobotics - apriltags images*, `https://github.com/AprilRobotics/apriltag-imgs`, Accessed: 2022-10-17.

[41] *Parrot sphinx*, `https://developer.parrot.com/docs/sphinx/masterindex.html`, Accessed: 2022-10-17.

[42] *Qualisys motion capture system*, `https://www.qualisys.com/`, Accessed: 2022-10-17.

[43] *Qualisys - cybernetics*, `https://home.hvl.no/ansatte/gste/ftp/MarinLab_files/Manualer_utstyr/QTM-usermanual.pdf`, Accessed: 2022-10-28.

[44] M. H. Jones, *Calibration checkerboard collection*, `https://markhedleyjones.com/projects/calibration-checkerboard-collection`, Accessed: 2022-10-24.

[45] J. Redmon, *Darknet: Open source neural networks in c*, `http://pjreddie.com/darknet/`, 2013–2016.

[46] M. Bjelonic, *YOLO ROS: Real-time object detection for ROS*, `https://github.com/leggedrobotics/darknet_ros`, 2016–2018.

[47] A. Richardson, J. Strom, and E. Olson, "Aprilcal: Assisted and repeatable camera calibration," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1814–1821. DOI: `10.1109/IROS.2013.6696595`.

[48] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. DOI: `10.48550/ARXIV.2207.02696`. [Online]. Available: `https://arxiv.org/abs/2207.02696`.

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