Master's thesis

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

Martin Falang

Autonomous UAV Landing on a Boat - Perception, Control and Mission Planning

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2022

NTNU
Norwegian University of
Science and Technology

Martin Falang

Autonomous UAV Landing on a Boat - Perception, Control and Mission Planning

Master’s thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2022

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

Abstract

The potential use cases for small unmanned aerial vehicles (UAVs) are ever-increasing as they have become significantly more capable and cheaper over the past decades. One area where they could provide meaningful contributions is in search and rescue (SAR) missions, where they could help speed up the search for victims.

Landing is a common challenge for all autonomous UAV missions. This thesis aims to land a Parrot Anafi quadcopter UAV autonomously on a helipad mounted on the DNV ReVolt marine vessel at sea.

The thesis breaks down the landing problem into perception, control, and mission planning. For perception, two different computer vision algorithms are combined in a model-based Kalman filter to estimate the relative position between the Anafi and the helipad. The control system used is a cascaded structure of position, velocity, and attitude control, where different velocity and position control methods are investigated. The thesis finally investigates whether artificial intelligence (AI) planning is a suitable form of mission planning for this problem.

The thesis concludes that the perception and control system was reliable enough to land the UAV on an entirely or nearly stationary helipad, indoors and with the helipad mounted on the ReVolt at sea. However, for landings where the helipad moved significantly, more accurate position estimation and a better-suited guidance system are necessary to land reliably.

The thesis demonstrates that AI planning could be used in this problem to generate a valid action sequence to solve a mission based on predefined goals. However, the actual effectiveness of such algorithms could not be determined due to the missions tested being insufficiently complex.

Sammendrag

De potensiale bruksområdene for ubemannede droner (UAV-er) fortsetter å øke ettersom de har blitt betydelig mer kapable og billigere i løpet av de siste tiårene. Ett område hvor UAV-er kan bidra positivt er innenfor søk og redningsaksjoner, hvor de kan hjelpe til å effektivisere søket etter ofre.

Landingen er en felles utfordring for alle autonome UAV-oppdrag. Målet med denne oppgaven er å lande en Parrot Anafi UAV-drone autonomt på en landingsplattform montert på det maritime fartøyet DNV ReVolt.

Oppgaven bryter ned landingen i oppfattelse, regulering, og oppdragsplanlegging. Innen oppfattelse brukes to forskjellige datasynalgoritmer sammen i et Kalman-filter for å estimere den relative posisjonen mellom UAV-en og landingsplattformen. Reguleringssystemet som brukes er en kaskaderegulator med posisjon-, hastighet-, og vinkelregulering, hvor bruk av forskjellige posisjons- og hastighetsregulatorer utforskes. Til slutt undersøker oppgaven hvorvidt algoritmer basert på kunstig intelligens for oppdragsplanlegging (AI planning) kan brukes for å planlegge oppdragene til en slik type UAV.

Oppgaven konkluderer med at oppfatnings- og reguleringssystemene var pålitelige nok til å lande UAV-en på en helt eller delvis stasjonær landingsplattform, både innendørs og med landingsplattformen montert på DNV ReVolt-båten på sjøen. I de landingene der landingsplattformen beveger seg betydelig kreves imidlertid mer nøyaktig oppfatning og bedre reguleringssystemer for å kunne lande pålitelig.

Oppgaven demonstrerer at AI planning kan brukes i dette problemet til å generere en gyldig sekvens med handlinger som løser et oppdrag basert på forhåndsbestemte mål. Imidlertid kunne ikke den faktiske effektiviteten til slike algoritmer bedømmes da oppdragene i denne oppgaven ikke var kompliserte nok.

Preface

This thesis is written in the spring of 2022 and is the conclusion to my master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway.

During the final years of my studies, I have specialized in the field of autonomous systems, which has developed into an area of great personal interest. I believe autonomous systems have the potential to have a meaningful impact on society in a positive way if used correctly. This, along with the fascination for the underlying technical concepts, is my main motivation for pursuing this field.

My motivation for choosing this thesis was the hands-on opportunity to combine multiple different elements of autonomous systems such as guidance, control, state estimation, computer vision, and mission planning into one complete system that can be tested in real-world experiments on a small quadcopter drone. Another motivational factor is the potential for the project to have an eventual real-world impact, as it works towards the end goal of creating an autonomous drone system capable of aiding human rescuers in search and rescue missions at sea.

The project is a continuation of the work done by Peter Bull Hove [1] and Thomas Sundvoll [2], who in their master's theses explored the use of computer vision for estimating the pose of a quadcopter when landing on a helipad. Sundvoll started the project by creating a helipad and developing pose estimation algorithms based on traditional computer vision methods, while Hove continued the project by combining the methods of Sundvoll with deep learning-based computer vision in a Kalman filter. Specifically, the part used from Hove is the deep learning-based pose estimation system, while the physical helipad designed by Sunvoll along with its Gazebo model is used as the landing platform in this thesis.

This thesis is also a direct continuation of my specialization project [3], where I ported the works of Hove and Sundvoll to the modern Anafi quadcopter drone and added new techniques for pose estimation using traditional computer vision. The contributions from that project that are used in this thesis are an interface between the Robot Operating System and the Anafi quadcopter that has been further modified in this thesis, a corner identification algorithm for finding known points on the helipad in an image, and scripts for recording perception output for evaluation.

Anastasios Lekkas from the Department of Engineering Cybernetics (ITK) at NTNU has been my academic supervisor during this thesis. He has helped shape the course and aims of the thesis and provided valuable high-level insight and guidance during the entire project.

Miguel Hinostroza, also from ITK, has generously provided me with an implementation of the Graphplan algorithm, along with a valuable insight into the field of AI planning. He has also helped with assistance and insight during the experiments with landing on the DNV ReVolt vessel.

Tom Arne Pedersen and Christopher Strøm from DNV have supplied me with the DNV

ReVolt vessel for testing autonomous drone landings on a boat and have generously helped with tests both on land and at sea.

Paolo De Petris from ITK has supplied me with access to the Arena52 autonomous drone lab at NTNU, and has greatly assisted with the setup and use of its motion capture system.

Terje Haugen at ITK has provided me with a portable power generator to use to power the laptop when testing outside where no power outlets are available.

The department has also provided me with two Parrot Anafi FPV quadcopters for the project, along with a Komplett Khameleon P9 Pro laptop.

Software released by Parrot used are the Olympe API 3.0.0 for interfacing with the Anafi on the laptop and the Parrot-Sphinx 1.8 using Gazebo 7.0.1 for flying the Anafi in simulation.

Other open-source software used in this thesis include Python 3.6.9, ROS Melodic with rospy 1.14.12, darknet_ros from Legged Robotics [4], and Python libraries OpenCV 4.5.4-dev, SciPy 1.5.4, NumPy 1.19.5, and scikit-learn 0.24.2.

All figures in this thesis are created by the author unless stated otherwise.

Acknowledgements

Firstly I would like to thank my academic supervisor Anastasios Lekkas for his guidance and support throughout the year with both the specialization project and this thesis. His help, both technical and academic, has greatly helped me with the technical results and the writing of this thesis.

The results from testing on the DNV ReVolt would not have been possible without Tom Arne Pedersen and Christopher Strøm from DNV. They eagerly helped with the testing and patiently stood in the cold water of Nidelva holding the ReVolt while I ran all my experiments, for which I am very grateful.

Also a big thanks to Miguel Hinostroza who helped me incorporate mission planning into my thesis on short notice, and to Paolo De Petris for assisting with indoor testing. Thanks to ITK at NTNU for providing me with access to the drone lab and all the tools needed to develop and test the systems in my thesis.

As my work is a continuation of Peter Bull Hove's master thesis from 2021, I would like to thank him for helping me get started on the project and for providing me with his results and files.

I would also like to thank my classmates for all the invaluable teamwork and discussions we have had during our studies, which have been instrumental in navigating the courses and projects we have had here at NTNU. Finally, I would like to thank my loving friends, family, and Hanna in particular for always supporting me.

Table of Contents

List of Tables

List of Figures

Abbreviations

NED North, east, down. 9, 11, 22, 30, 31, 52, 53, 54, 86, 87, 93

PDDL Planning Domain Definition Language. 25
PID Proportional integral derivative. 2, 3, 4, 6, 22, 23, 30, 55, 58, 59, 60, 74, 75, 77, 87, 88, 89, 93, 94
PnP Perspective-n-Point. 12, 13, 14, 43, 48, 49
PP Pure pursuit. 6, 59, 75, 76, 87, 88, 94

RMSE Root mean square error. 70, 71, 73, 74
ROS Robot Operating System. 3, 5, 6, 34, 37, 38, 40, 41, 49

SAR Search and rescue. 1, 2, 4, 5, 61, 79, 88, 90, 91, 94, 95
SIFT Scale-Invariant-Feature-Transform. 13, 46
SLAM Simultaneous localization and mapping. 4
SVD Singular value decomposition. 15

TCV Traditional computer vision. 3, 12, 43, 50, 51, 54, 65, 70, 71, 73, 74, 79, 82, 84, 85, 86, 89, 92, 93, 95

UAV Unmanned aerial vehicle. 1, 2, 3, 4, 5, 7, 11, 22, 24, 88, 90, 92, 94, 95

YOLO You only look once. 17, 18, 35, 49, 73, 84, 85, 90, 93

1 Introduction

This chapter presents the background and motivation underlying this thesis, as well as relevant previous work. It also states the objectives of the thesis and what specific contributions have been made.

1.1 Background and Motivation

The accelerating advance of technological progress is enabling technology to solve more and more problems in society. One field benefiting from the increased availability of computing power is the field of autonomous robotics, aimed at relieving humans of jobs classified as dull, dirty, or dangerous [6]. Relieving humans of dangerous jobs is particularly promising, as this can reduce the threat of loss of life for those involved.

Another reason for relieving humans of certain jobs is that autonomous robotics may outperform humans at specific tasks. An example of this is the area of self-driving cars, which bear the promise of one-day reducing road accidents due to self-driving cars eventually outperforming human drivers. Much of autonomous robotics is part of the field of artificial intelligence, where some tasks are already closing in on human performance, such as object detection [7], and others have already surpassed human performance, such as playing the game of Go [8].

Jobs are also not binary between being done by humans or autonomous robots. Hybrid solutions are possible where autonomous robotics is an asset to the human operator to increase efficiency. Search and rescue (SAR) missions are one such area where the use of autonomous robotics could be beneficiary. When searching for a victim during a SAR mission, time is the main factor determining the survival rate of the victims [9], and therefore having autonomous solutions to help speed up the search phase of these missions could save lives.

An area of autonomous robotics that could assist rescuers during SAR missions is by using small unmanned aerial vehicles (UAV)s. UAVs have seen much commercial and academic interest over the past years and have been applied in various civil applications such as agriculture, construction and infrastructure inspection, delivery of goods, SAR missions, and surveillance [10]. For use in SAR missions, these drones could be flown by an operator or be completely autonomous, where having an autonomous UAV has the benefit of letting the operator focus on other tasks. Using such an autonomous UAV has been shown to be able to reduce the search time the rescuers use to find a victim, such as in [11] where the authors show that using an autonomous UAV helps to find mission persons in avalanches faster.

Although the potential for autonomous UAVs for such missions are high, the challenges are also many. The UAV must be an asset to the rescue team and not a liability, meaning the system must be robust and reliable enough to require minimum human intervention. All different aspects of the mission, such as takeoff, searching, and landing, must therefore be autonomous, where notably landing the UAV can be challenging due to limited time and space. This problem has therefore been an active area of research for over a decade [12].

Creating a fully autonomous UAVs for SAR missions is challenging due to the different subsystems that need to be implemented such as perception, control, and mission planning. Relevant works in these areas include deep neural network (DNN) based object detection [13, 14], UAV control using various methods such as a proportional–integral–derivative (PID) controller [15] or model predictive control (MPC) [16], mission planning using the Graphplan algorithm [17], using reinforcement learning to hover and land on a moving platform in [18, 19], and state estimation of a UAV relative to a moving platform [20].

A *quadcopter* is a type of UAV which has four propellers mounted in an X-shape. This propeller configuration means the drone has high maneuverability and can take off and land vertically, thus not requiring long runways, as is the case for fixed-wing drones. The availability of such UAVs has increased a lot in recent years, with prices falling and performance increasing. Modern quadcopters are often equipped with high-resolution cameras, sophisticated control systems, and collision avoidance while still being affordable to the general public, such as the drones from DJI[1] and Parrot[2]. The capability and availability of such drones make them suitable for experimentation across multiple new domains, such as assisting in a SAR mission through an autonomous search.

This thesis aims to investigate different problems related to creating an autonomous UAV system capable of participating in a marine SAR mission. A commercially available Parrot Anafi drone is used for testing and demonstrations. The reason for using such a drone is that it comes ready-to-fly meaning all effort can be put into creating the necessary software.

1.2 Previous work

1.2.1 Work on the project

This thesis follows the work done by Peter Bull Hove and Thomas Sundvoll in their respective master's theses [1, 2], as well as the specialization project [3] written by the author prior to this thesis.

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

In [2], the author designed a landing platform and a system for estimating the pose of a drone relative to the platform. The landing platform design replicated regular landing platforms used for helicopters while still having distinctive features that a computer vision pose estimation system could utilize. It was made to be mounted on top of DNV's autonomous research vessel *ReVolt* with the eventual goal of performing a landing at sea. The author also created a computer model of the landing platform to be used in simulation. The pose estimation system utilized traditional computer vision (TCV) techniques such as color segmentation, edge detection, and corner detection to determine the drone pose. Experiments showed promising results in the simulator, but the results were not transferable to real-life experiments where the pose estimates were found to be unreliable. Reasons for these poor results were identified to be changes in lighting conditions, the presence of wind, and noisier images than in the simulator.

This work was expanded in [1], where the author added a deep neural network based computer vision (DNN-CV) pose estimation system, which were combined with the TCV approach from [2] and the drone's onboard sensors in a Kalman filter (KF). The results showed that the DNN-CV pose estimation gave noisier estimates than the TCV approach by [2], but worked in real-world experiments, contrary to the TCV approach, which only worked in simulation.

The experiments performed in [1] and [2] were performed on an AR.Drone 2.0 quadcopter developed by the French drone company Parrot[3]. This drone was released in 2010 and is equipped to perform both indoor and outdoor flights but does not have a Global Positioning System (GPS) sensor for absolute position measurements. In [1], the author found that this drone did not perform well outdoors, and this limited the extendability of the simulation results in real life. This was found to be due to the outdated and limited hardware on the AR.Drone 2.0, and the author recommended using an upgraded drone platform to test the system in real-world experiments.

Much of the work in this thesis is based on the specialization project written by the author [3] which upgraded the drone platform to a modern Parrot Anafi, manufactured by the same company as the AR.Drone 2.0. The project created a Robot Operating System (ROS) interface similar to the ROS interface *ardrone_autonomy* [21] available for the AR.Drone 2.0. The project also built upon the work in [1, 2] by porting the DNN-CV pose estimation from [1] to the Anafi platform, and by developing a new TCV pose estimation system based on the work in [2]. The DNN-CV pose estimation system proved to work well after being ported to the Anafi, while the new TCV system did not work in real-world experiments due to noisy images.

1.2.2 Perception and control for autonomous UAVs

There has been much work previously done regarding the field of autonomous UAVs.

Mathematical modeling of quadcopter drones has been investigated in multiple sources such as [22, 23], where the authors also develop PID controllers for stabilizing the drone. More advanced control techniques has also been investigated, such as MPC [24], and reinforcement learning [18, 19].

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

The perception problem of pose estimation on drones has also been covered in the literature, where [25] presents a pose estimation system using visual odemtry and an extended Kalman filter (EKF), while [26] demonstrates the use of simultaneous localization and mapping (SLAM) to navigate a quadcopter indoors.

Regarding autonomous drone landings, [20] proposes a visual pose estimation system for autonomous landings on a moving platform. The authors use fiducial markers in a homography-based approach along with an EKF to estimate the pose of the UAV while using PID controllers to control the drone. Their experimental setup includes the same AR.Drone 2.0 as used by [1] and [2], and demonstrates good accuracy when landing on a moving target. The experiments of [20] are however only performed indoors with no external disturbances, so based on the conclusion by [1], these good results will likely not be achievable outdoors using the same drone.

Vision-based landing on a moving target was also explored in [27], where the authors also use fiducial markers for corrections in an EKF. They use a more advanced drone platform (DJI M100) and use MPC for controlling the drone. They also tested the system outdoors in a gentle breeze, where the drone managed to perform an accurate autonomous landing successfully.

1.2.3 UAVs in SAR missions

A survey on the use of UAV systems for civil applications, including SAR missions, is presented in [10]. The authors highlight that one of the main benefits of using UAV systems in SAR missions is reducing costs, as money is wasted each year on traditional SAR missions using helicopters. The authors also highlight the benefit of using a UAV system in chaotic and disaster-stricken areas where the UAV can provide network communication and medical supplies in inaccessible areas. One of the main challenges listed is having a robust system capable of operating in hostile weather, as this often accompanies SAR missions.

In [11], the authors demonstrate that the use of an autonomous drone equipped with an avalanche beacon can effectively perform a grid-search over an avalanche and speed up the search for buried persons. In [28], the authors use a UAV in order to locate bodies in disaster-struck areas, which often lack infrastructure and resources in the initial moments after a disaster hits. The authors also demonstrate in simulation how a UAV could help carry medical, food, and water supplies to the located victims. This delivery could be considerably faster than delivering the supplies manually, something which could save lives because of the time criticality discussed in [9].

A review on UAV systems for marine SAR missions is presented in [29]. Here the authors discuss the benefits of using UAV systems both for the rescue team and for the victim in need of rescue. UAVs capable of operating in hostile weather can be used to localize victims quickly and safely, reducing the unnecessary time the rescuers have to spend in the hostile weather conditions while searching for the victims. A UAV can cover large distances and provide better situational awareness by delivering aerial images, something which can be beneficial in locating victims that have fallen overboard in an emergency on a ship. The authors also discuss how UAVs can be used to carry out flotation devices for the victims once they are located, increasing the chances of survival until help arrives.

The *EU-ICARUS project* [30] was a project aimed at creating autonomous tools for assisting humans in UAV missions. These tools include UAVs as well as unmanned surface vehicles and unmanned ground vehicles, with a focus on collaboration between the different vehicles. Similarly, the EU project *INGENIOUS* is an ongoing project aimed at aiding the first responders during their operations [31], where SINTEF is developing micro UAVs for indoor localization in GPS-constrained environments [32].

1.3 Objectives

The overall objective of this thesis is to make further progress towards the end goal of having a fully autonomous UAV system for use in marine SAR missions. Within this broad goal, this thesis is mainly concerned with the problem of creating a UAV system capable of *landing the Anafi UAV autonomously on a landing platform attached to a boat*.

To achieve this, several areas of autonomy are examined, including *perception*, *control*, and *mission planning*. The *perception* problem is a continuation and adaptation of the work done in the previous two master's theses [1, 2] and the specialization project [3]. The *control* problem is examined from the ground up due to the different interfaces of the Anafi UAV used in this project and the AR.Drone 2.0 used in [1, 2]. This thesis will compare different perception and control methods to determine which is most suited for the problem of landing autonomously on the landing platform. For *mission planning*, the thesis investigates whether artificial intelligence (AI) planning algorithms can be applied to this domain and if such algorithms can be suited for this type of autonomous UAV SAR system.

These objectives can be summarized by the following questions related to perception, control, and mission planning which this thesis attempts to solve:

- What type of perception system is suited for estimating the Anafi's position relative to the landing platform while landing?
- What type of control system is suited for landing the Anafi reliably on the landing platform?
- Can AI planning algorithms be used for mission planning for autonomous UAVs used in SAR missions?

1.4 Contributions

The contributions made in this project are as follows in chronological order as they appear in the report, with further elaboration upon each point below:

- The release of the *anafi_ros*[4] software package for interfacing with Parrot Anafi drones using ROS.
- Using homography-based pose estimation based on finding known features, with feature detectors optimized using hyperparameter optimization (HPO).

[4]`https://github.com/mfalang/anafi_ros`

- Integrating the DNN-CV system from [1] and homography-based pose estimation in a model-based KF.
- Comparing two different velocity control methods for the Anafi based on a dynamic model of the drone and a PID controller.
- Comparing two different guidance laws for position control of the Anafi based on pure pursuit (PP) target tracking and point stabilization using a PID controller.
- Using the Graphplan algorithm to generate an action sequence given a set of goals for the Anafi to achieve, and creating a system to execute these actions, thereby giving the Anafi the ability to handle complex missions.
- Integrating the perception, control, and mission planning into a unified system and testing it in real-world experiments including landing on the DNV ReVolt vessel at sea.

The anafi_ros ROS package is an improvement of the Anafi Olympe application programming interface (API) and ROS interface created in the specialization project preceding this thesis [3]. This package allows for controlling and monitoring the Anafi through the use of standardized ROS topics. It has been released in a standalone version on Github to make it available to be used in other projects as well.

The specialization project preceding this thesis [3] implemented a homography-based pose estimation system, but the system was not reliable enough to work in real-world experiments. This was due to the system not being able to detect the known points on the helipad landing platform, which this thesis addresses by using a circle detector to filter out only the helipad and then a corner detector to find the known points in the image. The parameters of both detectors are optimized using HPO in the form of a grid search to improve the correct detection rate.

The total perception system developed for this thesis is a model-based Kalman filter based on the constant velocity (CV) model, using the homography-based pose estimation described above, DNN-CV system developed in [1], and velocity measurements from the Anafi as corrections. As the perception system is used during landings, it estimates the relative position between the drone and the helipad.

To control the position of the Anafi, the internal attitude controller of the Anafi is used in a cascaded system with a velocity controller and guidance law. The velocity controller generates attitude setpoints for the Anafi's internal attitude controller, and here a model-based open-loop calculation of the attitude is tested along with a PID velocity controller. Similarly, one level up, the guidance law generates velocity setpoints for the velocity controller based on the current position error. Here the simple pure pursuit (PP) target tracking guidance law is tested along with a point stabilization PID guidance law.

To plan the missions for the Anafi to execute, the AI planning algorithm Graphplan is used to generate an action sequence. The action sequence is generated based on some given mission goals, and action plans for missions of varying complexity are generated. Each action plan consists of the same basic actions in specific orders, and a mission executor is then implemented to follow the given action sequence.

Finally, the thesis integrates the perception, control, and mission planning systems into one system, which is extensively tested in real-world experiments. Autonomous landings are tested with the helipad attached to the DNV ReVolt vessel on land and at sea, as well as a short mission where the drone simulates a search in one area before returning to the landing platform to land. Autonomous landings where the helipad is moving are also tested to determine the capabilities of the perception and control systems.

The main contributions made in this thesis are the contributions integrating the different measurements in the perception system, investigating velocity and guidance control methods suitable for landing the drone, demonstrating the use of the Graphplan algorithm for mission planning of autonomous UAVs, as well as testing the complete system by landing on the DNV ReVolt.

1.5 Outline

This thesis begins with Chapter 2 presenting the theory and necessary background material for the work done in this thesis. Chapter 3 then presents the experimental setup used in this thesis before Chapter 4 presents the methodology used for developing the different subsystems in this thesis. Chapter 5 presents and explains the results from testing the various parts of the system individually and together, both in simulation, the NTNU drone lab, and outside on the DNV ReVolt. Chapter 6 discusses these results and gives an overall assessment of the system, before Chapter 7 concludes the thesis with some closing remarks outlining potential future work on the project.

2

Theory

This chapter presents the necessary theory underlying the work done in this thesis. The chapter starts with basic quadcopter modeling before explaining the topics used in perception (computer vision, hyperparameter optimization, and Kalman filtering). Then the topics needed for the control part of this thesis are presented before the necessary mission planning theory is presented last.

2.1 Quadcopter modeling

This section is based on the quadcopter modeling from the preceding specialization project [3], with the addition of the camera frame and modeling aerodynamical effects.

2.1.1 Basic quadcopter motion

A quadcopter is a UAV with four propellers mounted in an X-shape, as seen in Fig. 2.1. A separate DC motor drives each propeller, and all propeller speeds can be controlled separately by applying variable voltages to each motor. The thrust generated by each motor is linearly dependent on the square of the propeller rotation speed [16], and is denoted F_i in Fig. 2.1. The motors all generate a moment opposite their direction of rotation, which is why diagonal motors rotate in the same direction, and non-diagonal motors rotate in the opposite direction, as this is the only configuration that cancels out the total moments created by all the motors.

The attitude of the quadcopter can be controlled by generating different amounts of thrust for each motor [33]. The relationship between the forces F_i and the resulting attitude-behavior for roll, pitch, and yaw separately is listed below, where any attitude motion can be reached by combining the relationships:

- Pitch
  - Forward: $F_1 = F_2 < F_3 = F_4$

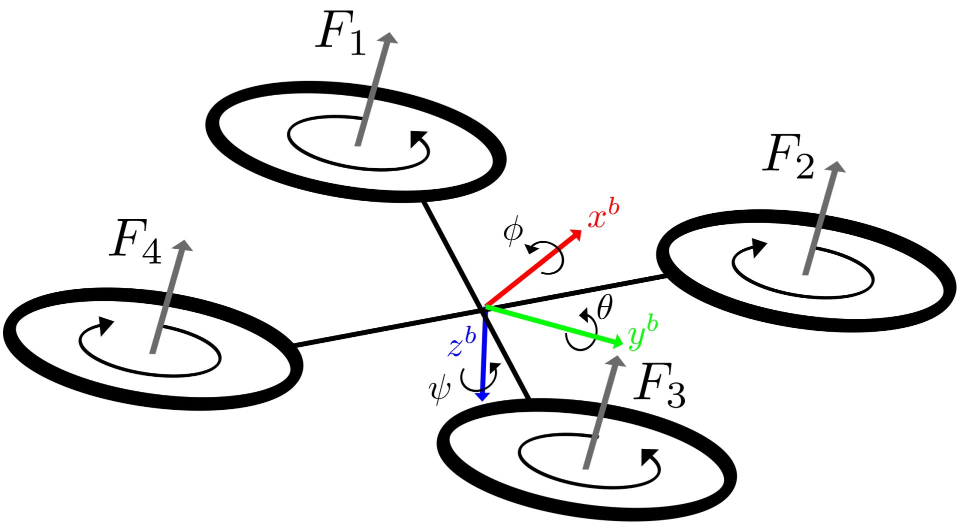


Figure 2.1: Model of a quadcopter UAV. Positive angles defined counter-clockwise in body frame.

  - Backward: $F_1 = F_2 > F_3 = F_4$

- Roll
  - Right: $F_1 = F_4 > F_2 = F_3$
  - Left: $F_1 = F_4 < F_2 = F_3$
- Yaw
  - Clockwise: $F_1 = F_3 > F_2 = F_4$
  - Counter-clockwise: $F_1 = F_3 < F_2 = F_4$

Position is controlled by controlling the attitude of the drone. Given that positive angles are defined counter-clockwise, the drone flies forwards and backward given negative and positive pitch angles, respectively. Similarly, it flies right and left given negative and positive roll angles, respectively. The upward and downward motion is controlled by varying the total thrust of all the motors, where increasing and decreasing the thrust move the quadcopter upwards and downwards, respectively.

2.1.2 Frames of reference

Two frames of reference are used to describe the motion of the quadcopter. The first is a frame attached to the drone, which describes linear and angular velocities. The second is a stationary frame of reference for describing the position and attitude of the quadcopter.

The stationary frame chosen will be the *north, east, down* (NED) frame, denoted $\{n\}$, which is a frame attached to an imaginary tangent plane fixed at a given position on Earth, and has the following axis as defined in [34]:

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

As this frame is based on a tangent plane fitted to Earth's surface, it is not suitable for global navigation due to Earth's curvature. However, this effect is negligible for local navigation, meaning the frame is well-suited for quadcopter control.

The frame attached to the quadcopter is the *body* frame, denoted $\{b\}$, fixed to the origin of the quadcopter, which is useful for describing the velocity of the quadcopter. Its axes can be seen in Fig. 2.1 and are defined as in [34]:

- x^b - Longitudinal axis pointing forward through the nose of the quadcopter.
- y^b - Transversal axis pointing right.
- z^b - Normal axis pointing down.

For quadcopters with a camera, the *camera* frame $\{c\}$ is useful for describing objects seen in the camera. It's origin is in the center of the image plane, with axes defined as in [35]:

- x^c - Right in the image plane.
- y^c - Down in the image plane.
- z^c - Normal to the camera plane, i.e. straight out of the camera.

To convert coordinates from one coordinate frame to another, the rotation and translation between the frames must be known. The general expression for a rigid body transformation for converting a point from frame $\{1\}$ to frame $\{0\}$ is defined as in [36]

$$\boldsymbol{p}^1 = \mathbf{R}_1^0 + \boldsymbol{t}_1^0 \tag{2.1}$$

where $\mathbf{R}_1^0$ is the rotation matrix from frame $\{1\}$ to $\{0\}$, and $\boldsymbol{t}_1^0 \equiv \boldsymbol{t}_{0\to 1}^0$ is the translation from the origin of frame $\{0\}$ to the origin of frame $\{1\}$ expressed in frame $\{0\}$.

2.1.3 Aerodynamical modeling

As the drone flies, it will be affected by aerodynamical effects. One of the main effects will be air resistance experienced by the drone when flying, known as *drag*. The general equation for drag can be found in [37] as

$$F_D = \frac{1}{2}\rho V^2 A C_D \tag{2.2}$$

where V is the drone's speed, A the area exposed to the wind, C_D the drag coefficient, and ρ is the air density.

Given the shape of a quadcopter and how it tilts to fly, the area A and the drag coefficient C_D will change with different tilt angles. A simplification of this model is to use a linear model such as the one given in [23]. Here the drag force along a given axis i is linearly proportional to the drone speed in that axis:

$$F_{D_i} = d_i v_i \tag{2.3}$$

where d_i is the drag coefficient for axis i.

Given that this model is a linearization of Eq. (2.2), it will be accurate at low velocities only. At higher velocities, other aerodynamical effects become significant, such as thrust changing with the angle of attack, blade flapping, and airflow disruption, and these have been examined in [38] and [39].

2.1.4 Equations of motion

The general equations of motion for a UAV with n propellers is found in [16]. These equations of motion include external forces that can be neglected, assuming that the quadcopter is flying unhindered and without wind. Assuming the drone is flying at low speeds, the aerodynamical forces F_{D_i} will be assumed to follow the linear drag model from Eq. (2.3) without any other aerodynamical effects. The authors in [16] assumed that the low-level attitude control of the UAV is handled by a fast onboard control system, which simplifies the attitude equations to first-order models. Given these assumptions and specifying $n = 4$ propellers in the result from [16], the following equations of motion for a quadcopter can be found

$$\begin{aligned} \dot{\boldsymbol{p}}^n &= \boldsymbol{v}^n \\ \dot{\boldsymbol{v}}^n &= \frac{1}{m}\left(\mathbf{R}_{nb}\sum_{i=0}^{4} F_{T_i} - \mathbf{R}_{nb}\sum_{i=0}^{4} F_{D_i}\right) + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \end{aligned} \tag{2.4}$$

$$\begin{aligned} \dot{\phi} &= \frac{1}{\tau_\phi}(k_\phi \phi_{ref} - \phi) \\ \dot{\theta} &= \frac{1}{\tau_\theta}(k_\theta \theta_{ref} - \theta) \\ \dot{\psi} &= \dot{\psi}_{ref} \end{aligned} \tag{2.5}$$

where $\boldsymbol{p}^n$ is the quadcopter position in the NED frame, $\boldsymbol{v}^n$ is the quadcopter linear velocity in the NED frame, $\mathbf{R}_{nb}$ is the rotation matrix between NED and body, F_{T_i} the thrust force from propeller i, g is the gravitational acceleration, k_ϕ, k_θ and τ_ϕ, τ_θ are dc-gains and time constants for roll and pitch closed-loop dynamics, and ϕ_{ref}, θ_{ref}, $\dot{\psi}_{ref}$ are respectively roll reference angle, pitch reference angle, and commanded yaw rate [16].

2.2 Computer vision

This section is based on the computer vision section in the preceding specialization project [3], with modifications to present the necessary background material needed in this thesis.

2.2.1 Traditional computer vision

Computer vision has been an active field of study since the 1970s, and its goal is to describe the world seen in one or more images [40]. During the recent decade, the field has seen an emergence of deep neural network (DNN) based computer vision approaches based on using a general model trained to solve a specific problem. This method differs from the classical computer vision approach of analyzing the problem domain to find a suitable solution, which can now be labeled as traditional computer vision (TCV) [41].

This section will elaborate on the part of TCV concerned with finding the pose of the camera relative to points in the real world, known as the Perspective from N points, or perspective-n-point (PnP) problem [42]. The two main steps in this problem are:

1. Feature detection and identification: Finding point correspondences between 3D points in the real world and 2D pixel points in the image.
2. Pose calculation: Determine the transformation between these point correspondences to determine the camera pose.

and these will be further elaborated upon after establishing the basic camera model used throughout this thesis.

Pinhole camera model and camera intrinsics

A mathematical model of a camera is necessary to describe how 3D points in the real world are projected into pixel coordinates in the image plane. One such model is the *pinhole camera model* where light from the scene passes through an infinitely small pinhole to form an inverted image in the image plane [43]. This simple model is only an approximation due to the pinhole not being infinitely small in practice. Mathematically, the pinhole is known as the optical center C. The distance from the camera center to the image plane is known as the focal length f, which together with C make up the intrinsic properties of the camera.

The focal length projects a 3D point into camera coordinates with origin in the center of the image plane. As the focal length is given in millimeters, it must be multiplied with the pixel density s_x and s_y to get the mapping in terms of pixel coordinates. The image coordinate origin is chosen to be in the top left corner of the image. Therefore the optical center c_x and c_y must be used to offset the pixels into image coordinates instead of camera coordinates.

The general relationship between 3D points and pixel coordinates can be put on matrix form by introducing the concept of *homogeneous coordinates* $\tilde{\boldsymbol{u}} = \begin{bmatrix}\tilde{u} & \tilde{v} & \tilde{w}\end{bmatrix}^\intercal$, which have the same dimension as 3D points. The relationship between the homogeneous pixel coordinates $\begin{bmatrix}\tilde{u} & \tilde{v} & \tilde{w}\end{bmatrix}^\intercal$ and regular pixel coordinates $\begin{bmatrix}u & v\end{bmatrix}^\intercal$ is $u = \tilde{u}/\tilde{w}$ and $v = \tilde{v}/\tilde{w}$. The mapping of a 3D point $\begin{bmatrix}X & Y & Z\end{bmatrix}^\intercal$ from world to homogeneous pixel coordinates will then be as in [35]

$$\begin{bmatrix}\tilde{u}\\ \tilde{v}\\ \tilde{w}\end{bmatrix} = \underbrace{\begin{bmatrix}s_x f & 0 & c_x\\ 0 & s_y f & c_y\\ 0 & 0 & 1\end{bmatrix}}_{\mathbf{K}} \begin{bmatrix}X\\ Y\\ Z\end{bmatrix}. \tag{2.6}$$

The matrix $\mathbf{K}$ is known as the camera calibration matrix [35], which encompasses the intrinsic properties of the camera.

Feature detetion

Feature detection is the process of finding features or objects of interest in the image. It can be used for different purposes, such as stitching together two images or generating a 3D point cloud based on the features in multiple images [40]. In the case of PnP problems, the features that need to be found are the location of the known 3D points in the image. This requires that the neighborhood around the feature is easily distinguishable and known in advance. One way of ensuring this is by using fiducial markers with specific patterns that are unique and will not be found anywhere else, such as AprilTags [44]. Another option is using a generic feature detector to find features and then determine later which of these features are the actual points of interest. In this case, different feature detectors aimed at different types of features exist.

A commonly used detector is a *corner detector*, and corners are good features as they can be localized, meaning that the exact location of a corner can be identified [43]. An example of applications where this is useful is when stitching together images where detecting the same corners in both images is necessary to determine where they overlap. Another case is when features are tracked across multiple images to determine the motion of the camera, such as is the case in ORB-SLAM [45].

One of the classical corner detectors is the Harris corner detector [46]. The Harris corner detector combines information about the magnitude of the gradients in both directions and how the total gradient swings in a local neighborhood [43]. A corner should have large gradients in both directions, and the gradient should swing sharply in a local neighborhood, and this is used in a scoring function to determine if a point is a corner or not. The Shi-Tomasi corner detector builds on the same principles but with a different scoring function which was shown to provide better results than the Harris detector [47]. Other examples of much-used feature detectors are the scale-invariant-feature-transform (SIFT) feature detector [48], which in contrast to the Harris and Shi-Tomasi detector work well on images of different scale, and the features from accelerated segment test (FAST) feature detector [49] which is more suited for real-time applications due to its fast runtime.

Two other types of feature detectors are *edge detectors* and *line detectors*. As edges and lines are continuous, they are not suited for the same applications as corner detectors as they do not describe unique points in the image. Edge detectors are typically found by computing the gradient of the image, while lines can be detected using a Hough transform, or Canny edge detector [40].

Similar to edges and lines, circles in an image can be found using a *circle detector*. One way of detecting circles is by using the Hough transform from above in an alternative way. Detecting circles using the Hough transform involves first finding all edges in an image, then creating circles from all of these edge points with a given radius [50]. If many enough of these circles intersect in a given point, then there is a circle with the given radius in that point. To detect circles of different sizes, the algorithm can be run multiple times with a varying radius to search for.

Camera pose estimation

Given established correspondences between image points and 3D world frame points, the transformation between these points must be found to determine the camera pose relative to the world coordinates. As stated above, this is known as the PnP problem, and in general, the world coordinates can be in any known configuration. Solutions to this problem include for example using the direct linear transformation (DLT) in combination with nonlinear optimization to minimize the reprojection errors [42].

In the particular case where the 3D world points are planar, meaning they all lie in the same plane, the problem is simplified and can be solved by using the concept of *homography* [42]. Homography is the method of computing the 2D projective transformation between two planes [35], where the planes can be, e.g. two images or one image and a physical plane. The transformation will be an estimate of the relative pose between the two planes, so given that one plane is the image plane and the other is a physical plane, the camera pose relative to the known points in the physical plane can be found.

The goal of homography is to estimate the *homography matrix* $\mathbf{H}$, which transforms a set of 2D points in the real world into image coordinates. The rotation and translation describing this transformation can then be extracted from $\mathbf{H}$.

To derive the homography matrix, first define $\boldsymbol{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^\mathsf{T}$ to be any point in the world frame of reference, and $\boldsymbol{X}^c = \begin{bmatrix} X^c & Y^c & Z^c \end{bmatrix}^\mathsf{T}$ to be the corresponding point in the camera frame of reference. The general relationship between these two points will then be

$$\begin{aligned} \boldsymbol{X}^c &= \mathbf{R}\boldsymbol{X} + \boldsymbol{t} && (2.7)\\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \boldsymbol{X} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} && (2.8)\\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} && (2.9) \end{aligned}$$

where $\mathbf{R}$ is the rotation between them, and $\boldsymbol{t}$ is the translation between the origins of each point. Using the assumption that the physical point lies on a plane, $\boldsymbol{X}$ becomes $\boldsymbol{X} = \begin{bmatrix} X & Y & 0 \end{bmatrix}^\mathsf{T}$, and the equation simplifies to

$$\begin{aligned} \boldsymbol{X}^c &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} && (2.10)\\ &= \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. && (2.11) \end{aligned}$$

Defining the points $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^\mathsf{T}$ as $x := (u - c_x)/s_x f$ and $y := (v - c_y)/s_y f$, the following relationship between $\boldsymbol{x}$ and $\boldsymbol{X}$ can be found similarly to the derivation in [35]

$$\begin{aligned} x &= \frac{X^c}{Z^c} = \frac{r_{11}X + r_{12}Y + t_x}{r_{31}X + r_{32}Y + t_z} \\ y &= \frac{Y^c}{Z^c} = \frac{r_{21}X + r_{22}Y + t_y}{r_{31}X + r_{32}Y + t_z}. \end{aligned} \tag{2.12}$$

This equation can be put on matrix form by converting $\boldsymbol{x}$ to homogeneous coordinates $\tilde{\boldsymbol{x}} = [\tilde{x}\ \tilde{y}\ \tilde{z}]^T$ such that

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.13}$$

where $\mathbf{H}$ is the homography matrix. To find $\mathbf{H}$, the system in Eq. (2.12) is first transformed into a linear system by using the DLT as in [35]

$$\mathbf{A}\boldsymbol{h} = \mathbf{0} \tag{2.14}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_n \end{bmatrix}, \mathbf{A}_i = \begin{bmatrix} X_i & Y_i & 1 & 0 & 0 & 0 & -X_i x_i & -Y_i x_i & -x_i \\ 0 & 0 & 0 & X_i & Y_i & 1 & -X_i y_i & -Y_i y_i & -y_i \end{bmatrix} \tag{2.15}$$

$$\boldsymbol{h} = \begin{bmatrix} r_{11} & r_{12} & t_x & r_{21} & r_{22} & t_x & r_{31} & r_{32} & t_z \end{bmatrix}^{\mathsf{T}}. \tag{2.16}$$

The vector $\boldsymbol{h}$ can then be solved for using the singular value decomposition (SVD) $\mathbf{A} = \mathbf{UDV}^T$ where $\boldsymbol{h}$ will correspond to the last column of $\mathbf{V}$ [35].

Although $\mathbf{H}$ has 9 entries, it is uniquely defined only up to scale, meaning it has 8 degrees of freedom. As each 2D point has 2 degrees of freedom, the minimum number of 2D point correspondences n needed to determine $\mathbf{H}$ uniquely is $n = 4$ [35].

As $\mathbf{H}$ is uniquely defined only up to scale, the resulting matrix $\mathbf{H}$ found from the SVD will in general not be equal to the $\mathbf{H}$ matrix in Eq. (2.13). This scale ambiguity can be solved by imposing the constraint that the columns of $\mathbf{H}$ corresponding to the columns of $\mathbf{R}$ must be of length 1, and then finding the scaling factor. This means the values for $\boldsymbol{t}$ and the first two columns of $\mathbf{R}$ can be found by dividing the elements of $\mathbf{H}$ by this scale factor. However, this scale factor means there are two possible solutions for $\mathbf{R}$ and $\mathbf{t}$, where only one will represent the actual transformation. To differentiate between the two, the 3D points $\mathbf{X}$ can be transformed back into the camera frame using the two transformations, and the valid transformation will be the one where the 3D points have a positive Z-coordinate, meaning they are in front of the camera.

Once $\mathbf{H}$ has been uniquely determined, given the structure defining $\mathbf{H}$ in Eq. (2.13), the translation vector $\boldsymbol{t}$ as well as the first two columns of $\mathbf{R}$ can be directly extracted. To reconstruct $\mathbf{R}$ from its first two columns, the property that all rotation matrices are orthogonal can be used [34]. This means that the last column of $\mathbf{R}$ can be calculated as the cross product of the two first columns

$$R_3 = R_1 \times R_2 \tag{2.17}$$

which completes the calculation of the relative pose $\mathbf{R}$ and $\boldsymbol{t}$ between the camera and the physical plane.

Pose refinement

The homography matrix $\mathbf{H}$ estimated using the DLT can be further optimized using an optimization method such as Levenberg-Marquardt (LM) optimization. LM optimization is a variation of the Gauss-Newton method for solving least-square minimization problems, which gives faster convergence, and regularization when the problem is overparameterized [35].

Both the Gauss-Newton and LM optimization require an objective function to be minimized, denoted $E(\boldsymbol{p})$, where $\boldsymbol{p}$ is the parameter vector to be optimized. The objective function can be written as

$$E(\boldsymbol{p}) = \sum_{i=1}^{n} r_i(\boldsymbol{p})^2 \tag{2.18}$$

where $r_i(\boldsymbol{p})$ is a residual function which outputs a scalar value based on the parameter vector $\boldsymbol{p}$. The goal of both methods is to iteratively update $\boldsymbol{p}$ in order to minimize $E(\boldsymbol{p})$, i.e. calculate a step $\boldsymbol{\delta}$ which is used to update the estimate $\hat{\boldsymbol{p}}$ using the update rule

$$\hat{\boldsymbol{p}} \leftarrow \hat{\boldsymbol{p}} + \boldsymbol{\delta}. \tag{2.19}$$

The Gauss-Newton method approximates the Hessian of the objective function in order to calculate the step $\boldsymbol{\delta}$. First, the Jacobian containing the partial derivatives of $\hat{\boldsymbol{p}}$ is found, and this will be an $n \times m$ matrix where n is the number of residuals in the objective function, and m is the number of parameters in $\boldsymbol{p}$. The Hessian is approximated as $\mathbf{H} \approx \mathbf{J}^T\mathbf{J}$ [35]. The step in the Gauss-Newton method $\boldsymbol{\delta}^{GN}$ can then be found by solving the linear system

$$\mathbf{J}^T\mathbf{J}\boldsymbol{\delta}^{GN} = -\mathbf{J}^T. \tag{2.20}$$

Solving this system requires inverting $\mathbf{J}^T\mathbf{J}$ which could become singular and cause the method to fail. The LM method avoids this by instead solving the system

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\boldsymbol{\delta}^{LM} = -\mathbf{J}^T \tag{2.21}$$

where $\mu > 0$ is a parameter that ensures that $\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$ is always invertible. The LM method also evaluates the proposed step $\boldsymbol{\delta}^{LM}$ by checking if this step decreases the objective function. If it does not, then μ is updated increased until a step $\boldsymbol{\delta}^{LM}$ that decreases the objective function is found. Each iteration that decreases the objective function similarly decreases μ. This parameter μ in effect makes the LM method switch between acting like the Gauss-Newton method and the gradient descent method [35]. When μ is small $(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\boldsymbol{\delta}^{LM} \approx \mathbf{J}^T\mathbf{J}\boldsymbol{\delta}^{LM}$ and the method resembles the Gauss-Newton method giving it a fast convergence. When μ is large $(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\boldsymbol{\delta}^{LM} \approx \mu\boldsymbol{\delta}^{LM}$, and the update rule becomes based on gradient descent which is in general not recommended, but will ensure that the objective function decreases. This adjusting of μ to change the behaviour of the LM method makes it robust and well-suited for iterative least-squares optimization.

2.2.2 Deep neural network-based computer vision

Deep learning (DL) is a subset of machine learning based on the concept of an artificial neural network (ANN), which is a programming paradigm inspired by the functioning of the human brain [41]. In contrast to conventional programming, where the programmer specifies how the computer solves a problem, approaches based on ANNs are instead made to learn from observations and figure out their own solutions [51]. A general ANN consists of an input layer, one or more hidden layers with neurons, and an output layer, where weights and biases connect adjacent layers. A deep neural network (DNN) refers to the case where the ANN has two or more hidden layers. DNNs are much used in computer vision since they allow the network to break down a complex classification problem into smaller manageable parts [51].

When training a DNN, the goal is to find the network weights and biases that minimize the error of the output [51]. A cost function based on the weights and biases is used to define the total output error between the predicted and actual output of the network. The gradient of this function is used to perform gradient descent to update the weights and biases, resulting in a higher accuracy of the network. As the parameters in a DNN can reach many million [14], the training process is computationally expensive and typically optimized to run on a dedicated graphics processing unit (GPU), or multiple, to increase performance.

Computer vision is one application where deep learning and DNNs show great promise, as such approaches have been the state of the art in image classification since the introduction of the AlexNet network [13]. These types of networks are typically convolutional neural networks (CNNs), which consist of convolutional layers which detect features in an image, pooling layers that reduce the dimension of the output of the convolutional layers, and fully connected layers to classify the image based on the features extracted earlier in the network [41].

Convolution in image processing is the technique of combining multiple pixel values into one using a weight matrix, and this is much used in traditional computer vision to extract features and filter images. In CNNs, these weight matrices are learned to allow the network to extract the features that give the best detection results.

In computer vision, there is a distinction between image classification and object detection. In image classification, the input image is classified into one of several predefined categories, e.g. when classifying a handwritten digit, the output will be a value from 0 to 9. On the other hand, object detection takes an image and tries to find all instances of each category in the image, e.g. finding all bikes, buses, and other cars in the image coming from an autonomous car.

YOLOv4 object detector

One type of object detector is the you only look once (YOLO) v4 detector [14]. YOLOv4 is a bounding-box prediction algorithm, meaning it estimates the position of each element in the image by placing a box around it, as seen in Fig. 2.2. Along with the bounding box are the category the detector believes the object belongs to and its certainty.

YOLOv4 has been designed for real-time applications and offers state-of-the-art accuracy and frames per second (FPS) performance. Both training and inference (returning bounding boxes for a given input image) can be performed on a single GPU, meaning YOLOv4 is well suited for real-time applications where clusters of GPUs are unavailable.

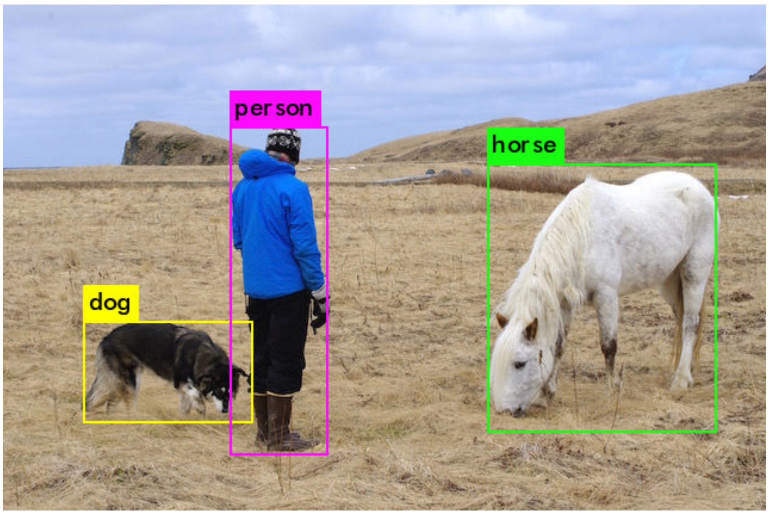


Figure 2.2: Object detection using YOLO v4 on a test image with person, dog and horse. This image has been created using the YOLO detector on a sample image provided in Darknet network backbone implementation [5].

2.3 Hyperparameter optimization

As much of today's research in the fields of machine learning and artificial intelligence involves complex models with many hyperparameters, the field of hyperparameter optimization (HPO) aimed at automatically optimizing these parameters has seen increased activity in recent years [52]. Automating this task has the benefit that it reduces human effort, improves the performance of the machine learning model, and increases the reproducibility of the results compared to a manual search [52].

The HPO problem can be defined as finding the optimal set of hyperparameters $\boldsymbol{\lambda}^*$ by evaluating different values on $\boldsymbol{\lambda}$ on a training and validation dataset, D_{train} and D_{val} respectively. Given the model $\mathbf{A}$ which relies on N hyperparameters, $\boldsymbol{\Lambda}$ can be defined as the configuration space, meaning all the possible configurations of the N hyperparameters. Given some cost function $\mathbf{V}$ which evaluates the performance of the hyperparameters $\boldsymbol{\lambda}$, the problem can be formally defined as

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}}{\arg\min}\, \mathbf{V}(\mathbf{A}_{\boldsymbol{\lambda}}, D_{train}, D_{val}). \tag{2.22}$$

There exist many different methods for HPO, where the simplest forms are the black-box approaches where no prior knowledge is needed about how the performance changes based on the hyperparameters. Examples of black-box approaches include *grid search*, *random search*, and *Bayesian optimization*. Grid search searches a predefined search area, and random search searches a random search area, both to find the best combination of parameters in the configuration space. Bayesian optimization is an iterative method that builds a model based on the previous parameters and then determines which points to test next, and has been used effectively in tuning the hyperparameters of neural networks [52].

There are multiple challenges that must be considered when performing HPO, two of them being the cost of the evaluation function and the configuration space complexity. The evaluation function might be very costly, taking weeks or months in extreme cases to evaluate [53], e.g. in the training of deep neural networks, and this may not be feasible due to time constraints in the project. The configuration space might also be very complex with many different hyperparameters, where it is difficult to know which parameters actually affect the performance and what the reasonable range for the given parameters should be [52]. This means that the best hyperparameters found $\boldsymbol{\lambda}^*$ might not be the optimal parameters after all.

2.4 Kalman filtering

This section explains the theory behind the Kalman filter (KF) as well as the extension of the filter for non-linear filtering in an extended Kalman filter (EKF). However, only the standard KF was used in this thesis.

2.4.1 Models

A Kalman filter is a recursive filter that can be used to estimate the states of linear and non-linear systems [34]. The filter consists of two models: a process model and a measurement model. The two models can be summarized in state-space form as

$$\boldsymbol{x}_k = \mathbf{A}\boldsymbol{x}_{k-1} + \mathbf{B}\boldsymbol{u}_k + \boldsymbol{v}_k \qquad \boldsymbol{v}_k \sim \mathcal{N}(0, \mathbf{Q}) \tag{2.23}$$

$$\boldsymbol{z}_k = \mathbf{C}\boldsymbol{x}_k + \mathbf{D}\boldsymbol{u}_k + \boldsymbol{w}_k \qquad \boldsymbol{w}_k \sim \mathcal{N}(0, \mathbf{R}) \tag{2.24}$$

$$\boldsymbol{x_0} \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \mathbf{P}_0). \tag{2.25}$$

Here, $\mathbf{A}$ and $\mathbf{B}$ are the matrices defining the noise-free state transition $\boldsymbol{x}_{k-1} \rightarrow \boldsymbol{x}_k$ given the previous state estimate $\boldsymbol{x}_{k-1}$ and the current system input $\boldsymbol{u}_k$, and $\boldsymbol{v}_k$ is the process noise, which is assumed Gaussian with zero-mean and variance $\mathbf{Q}$. For the measurement model, the matrices $\mathbf{C}$ and $\mathbf{D}$ relate the measurement $\boldsymbol{z}_k$ to the state $\boldsymbol{x}_k$ and input $\boldsymbol{u}_k$, and $\boldsymbol{w}_k$ is the measurement noise, which is assumed Gaussian with zero-mean and variance $\mathbf{R}$. Finally, the initial state is normally distributed around the initial estimate $\hat{\boldsymbol{x}}_0$ and the initial covariance estimate $\mathbf{P}_0$. These models follow the Markov assumption that each state is only dependent on the previous state, while each measurement is only dependent current state [54].

Given these models, the Kalman filter will be the optimal solution to the problem of estimating the state $\hat{\boldsymbol{x}}$ given the process and measurement noise. However, the requirement that the models are linear will not always be satisfied in real life. In the case of nonlinearities, the models can be linearized around each estimate, in which case the filter is called an extended Kalman filter (EKF). The equations describing the transition $\boldsymbol{x}_{k-1} \rightarrow \boldsymbol{x}_k$ are now described by the nonlinear function $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$, in addition to the nonlinear function $\boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ relating the measurements to the state. The rest of the equations are the same as the linear case and can be summarized as follows

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{v}_k \qquad \boldsymbol{v}_k \sim \mathcal{N}(0, \mathbf{Q}) \tag{2.26}$$

$$\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k \qquad \boldsymbol{w}_k \sim \mathcal{N}(0, \mathbf{R}) \tag{2.27}$$

$$\boldsymbol{x}_{\mathbf{0}} \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \mathbf{P}_0). \tag{2.28}$$

Assuming that the system obeys the Markov assumption, these equations can be linearized around the previous posterior estimate $\hat{\boldsymbol{x}}_{k-1}$ and the current prior estimate $\hat{\boldsymbol{x}}_{k|k-1}$ [54]. The linearization can be performed by using a Taylor expansion as in [54], which will result in the matrices $\mathbf{F}$ and $\mathbf{H}$ being the Jacobians of the nonlinear functions $\boldsymbol{f}$ and $\boldsymbol{h}$ as follows

$$\mathbf{F}(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k) = \mathbf{F}_k = \frac{\partial}{\partial \boldsymbol{x}_{k-1}} \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\bigg|_{\boldsymbol{x}_{k-1}=\hat{\boldsymbol{x}}_{k-1}} \tag{2.29}$$

$$\mathbf{H}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k) = \mathbf{H}_k = \frac{\partial}{\partial \boldsymbol{x}_k} \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k)\bigg|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_{k|k-1}}. \tag{2.30}$$

2.4.2 Filtering

The Kalman filter algorithm conceptually involves two steps: (i) Given a current estimate, predict the next state estimate based on the system model, and (ii) update this prediction based on data from new measurements. To make sure that the posterior estimate is optimal, the covariance $\mathbf{P}_k$ of the estimate is also predicted and updated at each step. When predicting without any measurements to correct the prediction, called dead-reckoning, the covariance estimate will grow due to the model uncertainty $\mathbf{Q}$. When a new measurement arrives, the covariance will be reduced, as the estimate is now more confident due to it being a mix of the prediction and measurement.

In the prediction step, both the predicted state $\hat{\boldsymbol{x}}_{k|k-1}$ and the predicted state covariance $\mathbf{P}_{k|k-1}$ are calculated. These are calculated based on the nonlinear transition function $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$, the linearized transition matrix $\mathbf{F}(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k)$, and process model covariance $\mathbf{Q}$.

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{2.31}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^{\intercal} + \mathbf{Q} \tag{2.32}$$

The *innovation* $\boldsymbol{\nu}$ is defined as the error between the predicted measurement $\boldsymbol{z}_{k|k-1}$ and the actual measurement $\boldsymbol{z}_k$ [54], where both are defined as follows

$$\boldsymbol{z}_{k|k-1} = \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k) \tag{2.33}$$
$$\boldsymbol{\nu}_k = \boldsymbol{z}_k - \boldsymbol{z}_{k|k-1}. \tag{2.34}$$

The goal of the filter is to fuse the state prediction with the measurement in an optimal way, and this is done by adding a weighting of the innovation to the predicted state. This weighting is known as the Kalman gain $\mathbf{W}$ [54], and this gain will be optimal in the linear case under the assumptions described above. The optimality proof is lost in the case of the EKF due to the linearization, but EKFs still show excellent performance in most navigation systems [34]. The Kalman gain is calculated based on the predicted covariance $\mathbf{P}_{k|k-1}$, and the innovation covariance $\mathbf{S}_k$ which is a result of $\mathbf{P}_{k|k-1}$ and the measurement covariance $\mathbf{R}$. The equations for the innovation covariance $\mathbf{S}_k$ and Kalman gain $\mathbf{W}_k$ can be summarized as follows

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\intercal + \mathbf{R} \tag{2.35}$$
$$\mathbf{W}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\intercal \mathbf{S}_k^{-1}. \tag{2.36}$$

This gain $\mathbf{W}_k$ form the basis for the update rules of the Kalman filter together with the predicted state and covariance. Both the posterior state and covariance estimates are updated using update rules as defined as in [54]

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{W}_k \boldsymbol{\nu}_k \tag{2.37}$$
$$\mathbf{P}_k = (\mathbf{I} - \mathbf{W}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{2.38}$$

where $\mathbf{I}$ is the identity matrix with the same dimension as the number of states in the system.

2.5 Guidance and control

A fundamental part of autonomy is how the autonomous robot interacts with the environment to achieve its goals. This can be, e.g. the voltage applied to the motors of a robot arm or the engine velocity and steering wheel angle in an autonomous car. When the goal of the system is to move from one position to a given position autonomously, the two main parts of the problem are (i) computing the path or trajectory that will lead the system to the given position and (ii) being able to follow that path. These two problems are known in the literature as *guidance* and *control* [34].

2.5.1 Guidance

Guidance provides a methodology for the motion behavior needed for the achievement of motion control objectives [55]. Different types of guidance systems include point stabilization, trajectory tracking, target tracking, and path following [34], where point stabilization is a special case when the desired position is constant. In trajectory tracking, the goal is to follow a time-varying trajectory, whereas path following is time-invariant, meaning the system only needs to follow the path regardless of the time it takes to do so. Target tracking is used when there is no information about the path or trajectory, such as following an object with an unknown movement pattern. Target tracking can also be used to track stationary objects, making it analogous to point stabilization [55].

When performing points stabilization, the guidance system can be a simple proportional–integral–derivative (PID) position controller. This controller can also be used for trajectory tracking, given that the system response is faster than the trajectory [34]. Reference models are typically used for trajectory tracking, and these are elaborated on below. Different guidance laws can be used in target tracking when following a moving object. The simplest is the *pure pursuit* guidance law, which is inspired by a predator chasing a prey in the animal world [34]. Here the desired velocity $\boldsymbol{v}_d^n$ in the NED frame is

$$\boldsymbol{v}_d^n = -\kappa \frac{\boldsymbol{p}^n - \boldsymbol{p}_t^n}{\|\boldsymbol{p}^n - \boldsymbol{p}_t^n\|} \tag{2.39}$$

where $\kappa > 0$ is a constant.

The more advanced line-of-sight (LOS) guidance law computes the desired course angle that will lead the system onto an imaginary line between a reference point and the target [34]. This type of guidance is useful for underactuated systems such as marine vessels and fixed-wing aircraft, which move horizontally by changing their heading angle. For more details regarding LOS guidance, see [34] and [55] for marine applications, and [37] for UAV applications.

2.5.2 Reference models

The reference is the input to the control system and specifies desired responses, e.g. for the desired position, velocity, or attitude. This reference can be stationary in the case of point stabilization or can be varying in the case of target tracking where the reference will be changing as the target moves [34]. In the case of a varying reference, if the reference is computed at discrete intervals, it will have jumps where it changes instantaneously. This can be unwanted in a control system because it can lead to aggressive maneuvering and cause the control system to go into saturation.

A *reference model* can be used to smooth the reference signal out. The simplest reference model is simply a low-pass filter that smooths out the reference signal [34], and this corresponds to a reference model of order 1. The order of the filter determines both the complexity and how many smooth states it will compute. A 1. order reference model for velocity will only produce a smooth trajectory for the velocity, while a 3. order reference model will produce a smooth trajectory for velocity, acceleration, and jerk. When using a velocity reference model, it should be of at least order 2 to obtain smooth reference signals for both velocity and acceleration [34].

To achieve accurate tracking using the reference model, the reference model must have a lower bandwidth than the system dynamics [34]. This means that the system dynamics are faster than the reference model dynamics, which means the system will be able to track the reference well. The parameters defining the behavior for a 2. order reference model are the relative damping ratio ζ and the natural frequency ω_n. Increasing the natural frequency increases the response time of the model, while the relative damping ratio determines if the system is under-, over-, or critically damped. An underdamped system with $\zeta < 1$ will have oscillations, while an overdamped system with $\zeta > 1$ will have a slow response with no oscillations.

The general differential equations for a 2. order reference model with n states is as given in [34]

$$\ddot{\boldsymbol{\nu}}_d + 2\boldsymbol{\Delta}\boldsymbol{\Omega}\dot{\boldsymbol{\nu}}_d + \boldsymbol{\Omega}^2\boldsymbol{\nu}_d = \boldsymbol{\Omega}^2\boldsymbol{r} \tag{2.40}$$

where $\boldsymbol{\nu}_d$ is the desired velocities for each state, $\dot{\boldsymbol{\nu}}_d$ the desired accelerations for each state, and $\boldsymbol{r}$ the raw velocity references for each state. The matrices $\boldsymbol{\Delta}$ and $\boldsymbol{\Omega}$ are defined based on the relative damping ratios and natural frequencies for each state as follows

$$\begin{aligned} \boldsymbol{\Delta} &= \text{diag}\{\zeta_1, \zeta_2, ..., \zeta_n\} \\ \boldsymbol{\Omega} &= \text{diag}\{\omega_{n_1}, \omega_{n_2}, ..., \omega_{n_n}\}. \end{aligned} \tag{2.41}$$

This can be put on state space form by defining $\boldsymbol{x}_d = [\boldsymbol{\nu}_d^\intercal \ \dot{\boldsymbol{\nu}}_d^\intercal]^\intercal$, as done in [34]

$$\dot{\boldsymbol{x}}_d = \mathbf{A}_d\boldsymbol{x}_d + \mathbf{B}_d\boldsymbol{r} \tag{2.42}$$

where

$$\mathbf{A}_d = \begin{bmatrix} \mathbf{0}_{n\times n} & \mathbf{I}_n \\ -\boldsymbol{\Omega}^2 & -2\boldsymbol{\Delta}\boldsymbol{\Omega} \end{bmatrix}, \quad \mathbf{B}_d = \begin{bmatrix} \mathbf{0}_{n\times n} \\ \boldsymbol{\Omega}^2 \end{bmatrix}. \tag{2.43}$$

The velocity and acceleration signals for a 2. order reference model is shown in Fig. 2.3. Here, it can be seen that increasing the reference model's natural frequency increases its response time. A faster model follows the reference trajectory more accurately but at the expense of requiring a higher acceleration. When designing a reference model for a specific system, a trade-off between response time and maximum allowed acceleration must be made.

2.5.3 Control using successive loop closure

Control is the action of controlling the system's actuators to achieve a given control objective, in this case reaching a setpoint, trajectory or path given by the guidance system [34]. Different control methods can be used for this, such as the traditional PID controller, nonlinear control methods such as integrator backstepping and and feedback linearization [56], or methods based on optimization such as the linear quadratic regulator (LQR) or model predictive control (MPC) [57].

In the case of a PID controller, a linear model or transfer function mapping the input to the output signal is required. In some cases, such as controlling the position of a DC motor [58], this mapping can be easily found, and a single PID controller will be able to guide the system to the desired output. However, the mapping from input to output may be nonlinear or overly complicated in other cases. In such a scenario, it can be beneficial to split the problem into several simple controllers in succession rather than designing one

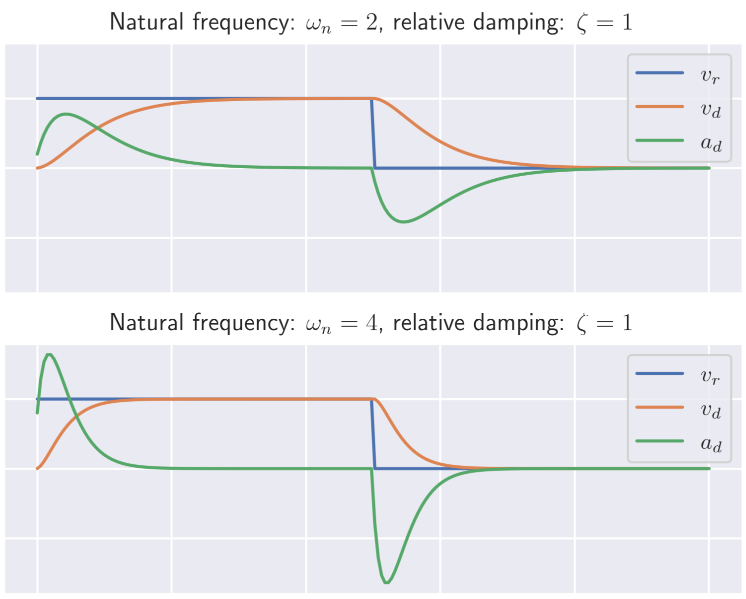
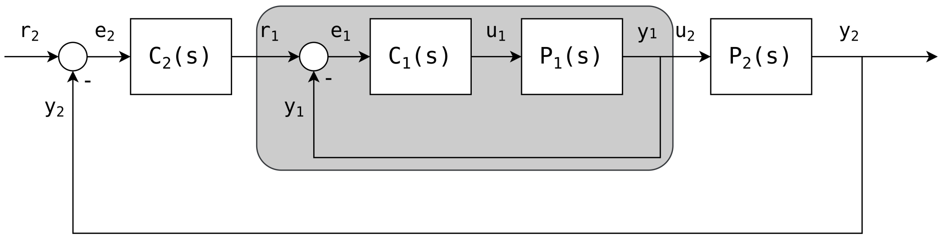


Figure 2.3: Comparison of two velocity reference models using with different natural frequencies ω_n.

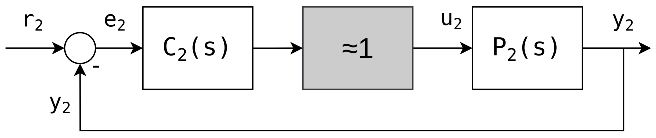
complicated controller for the entire problem, and this idea is known as successive loop closure [34, 37].

A block diagram showing successive loop closure with an inner and outer loop is shown in Fig. 2.4. Here, from the outer loop's perspective, the inner loop is a unity gain, and this is the main assumption in successive loop closure design. For this assumption to be satisfied, the inner loop has to, in theory, have an infinitely fast response, while in practice, 5-10 times faster than the outer loop is sufficient, meaning the inner loop bandwidth is 5-10 higher than the outer loop [34, 37].

Examples of successive loop closure applied to small fixed-wing UAVs can be found in [37]. Here successive loop closure is used in a lateral course hold autopilot, and a longitudinal altitude hold autopilot. In the lateral case, the outer loop is a course controller with a roll controller as the inner loop. For the longitudinal case, the outer loop is the altitude controller with a pitch controller as the inner loop. Similar applications of successive loop closure can be found for marine crafts and underwater vehicles in [34].



(a) Inner and outer control loop.



(b) Outer loop under assumption used in successive loop closure.

Figure 2.4: Block diagram showing the assumption underlying successive loop closure.

2.6 AI planning

AI planning, also known as automated planning and scheduling, is an area of artificial intelligence (AI) which chooses and organizes actions based on their expected outcome [59]. The *agent* is defined as the system interacting with the environment, and it chooses its actions based on the goal of reaching some predefined objective [60]. The first algorithm created to solve the AI planning problem was the STRIPS algorithm developed at Stanford Research Institute in 1971 [61], and since the field has seen the development of various different planning algorithms and techniques such as graph-based planners, temporal planners, and stochastic planners [59, 60].

2.6.1 Domain, actions and goals

To formalize the problem of AI planning, the following three inputs are defined as in [62]:

1. Initial state: The state of the agent in the world at time zero.
2. Goal: The desired end state of the agent.
3. Domain problem: Description of the possible actions that can be performed by the agent.

Formalizing these inputs in a standard language has the advantage that it makes comparing AI planning algorithms easier, and to this end, the planning domain definition language (PDDL) has been created [63]. To define a simple domain, the *propositions* or *state variables* are first defined. These can be thought of as variables that can either be true or false, e.g. if the agent is in a specific location or not. Each action is then defined with a *precondition* and *effect*. The precondition defines what proposition must be true or false to perform an action, e.g. the action of moving from location 1 to location 2 predicates that the agent is in location 1 to start. The effect defines what happens to the state variables after the action is performed, e.g. when moving from location 1 to location 2, the state variable for location 1 will be changed to false, and the state variable for location 2 to true.

2.6.2 Planning graphs and the Graphplan algorithm

An AI planning problem can be formulated using a concept known as *planning graphs*. Planning graphs are data structures that are used to estimate if a set of propositions is reachable from the initial state and, if so, with which actions [62]. A simple example of this could be a scenario where there are emails that have to be read and replied to, and a planning graph for this problem can be seen in Fig. 2.5. Here the states define having a new email available ("Have(email)") and having read the email ("Read(email)"). The possible actions are to read the email ("Read email") and to reply to the email ("Reply email"). Lines directly between two states represent *inactions*, or no-operations, i.e. if there is an email available and the agent does nothing, the email is still available. The dashed lines represent *mutex* states, which are two states that are mutually exclusive, i.e. in this example, it is impossible to have read an email and be finished with it since it must be replied to first.

Replying to an email requires that the email is already read, and the email will be considered done when it has been replied to. If the goal of the agent is not to have any emails, given that it started with one email, a solution for this problem can be seen to be the sequence of actions of first reading the email and then replying to it, highlighted in Fig. 2.5. This planning graph thus represents the simple problem of reading and replying to emails. Although this is a trivial case, the methodology is the same for a general problem with any number of state variables and actions.

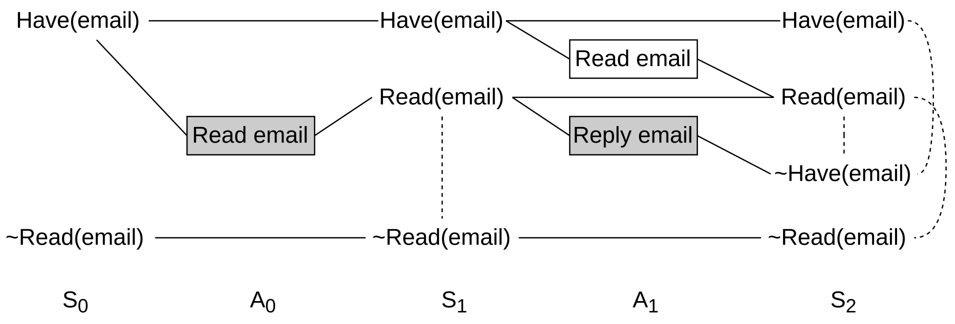


Figure 2.5: Example of a planning graph. Here "∼" represents the "not" operator.

Given a general planning problem that can be expressed in such a planning graph, the *Graphplan* [17] algorithm can be used to traverse the graph to find the set of actions that will bring the agent into the goal state.

Graphplan involves two main components: Forward planning graph expansion and backward searching the planning graph [59]. The graph is expanded during the forward expansion until either a solution may be found or that graph has reached a fixed point. A solution may be found if the last layer of state variables includes the goal state, while a fixed-point means the next layer of state variables is the same as the current. Given a layer including the goal state variables to start from, the planning graph can be searched to see if there is a valid set of actions from the initial state to this goal state.

These two components form the basis of the flow of the algorithm, which starts by expanding until it finds a layer where a solution may be found, and then if no solution is found from this layer it continues expanding until one is found. The termination criterium for the algorithm is reached when further expansion of the graph will not yield more possible solutions, which is evaluated by checking if the same exact nodes are checked in two layers in a row. The complete algorithm and explanation can be found in [59], and is summarized through the following steps:

1. Expand planning graph until solution may be found or solution cannot be found at all.
2. While no solution is found, expand the graph by one layer and test again until found. If no nodes change between two layers, terminate.

If there exists a solution to the planning problem, the Graphplan algorithm can be proven to find it [59].

3 Experimental setup

This chapter describes the experimental setup used throughout the testing done in this thesis. Apart from new tests using the DNV ReVolt vessel, the setup used in this thesis is essentially the same as the one used in the specialization project preceding this thesis, meaning several sections are based on [3].

3.1 Parrot Anafi

The hardware platform used as a basis for the experiments in this thesis is an Anafi FPV quadcopter, seen in Fig. 3.1. This is a 315 g drone with four rotors and one dynamically stabilized camera, released in 2018 by the French drone manufacturer Parrot. The most relevant aspects of the drone will be summarized below, while the full specifications can be found in [64].

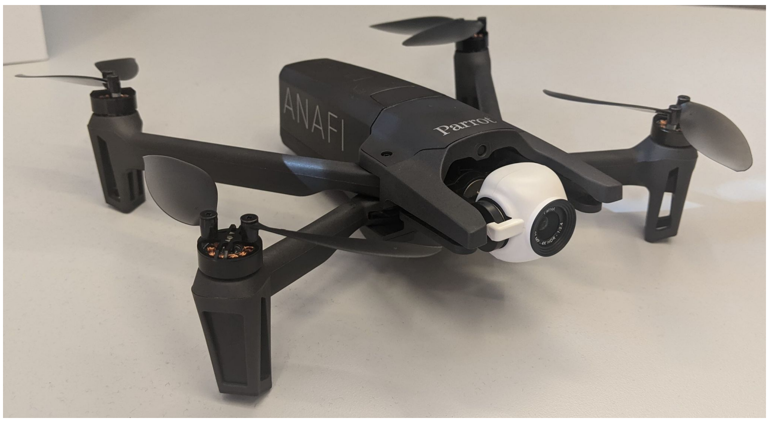


Figure 3.1: Parrot Anafi FPV.

3.1.1 Camera

The Anafi includes one camera stabilized in 3 axes through a hybrid of software stabilization and a mechanically stabilized 2-axis gimbal. The gimbal can also be used to tilt the camera a total of 180° in a given direction, e.g. straight up, ahead, or down.

The drone can capture high-quality video and images stored locally or lower-quality video streamed live to a smartphone or computer. Table 3.1 shows the resolutions and framerates in frames per second (FPS) for both modes, as well as the bitrate and end-to-end latency when streaming.

Table 3.1: Camera specifications.

Camera mode	**Resolution**	**FPS**	**Bitrate**	**Latency**
Local storage	4096×2160	24	-	-
Wi-Fi streaming	1280×720	30	5 Mb/s	280 ms

3.1.2 Sensors

Besides the camera, the additional sensors included on the Anafi are:

- IMU with
  - 3-axis gyroscope
  - 3-axis accelerometer
- Magnetometer
- Barometer
- GPS sensor
- Ultrasonic sensor

The output of these sensors is not directly available and is instead used in the internal state estimation on the drone.

3.1.3 State estimation

The internal state estimation of the Anafi features an 18-state extended Kalman filter (EKF) with the following states:

- Postion in NED frame (x,y,z)
- Velocity in body frame (x,y,z)
- Attitude (phi, theta, psi)
- Horizontal wind (x,y)

- Accelerometer biases (x,y,z)
- Gyroscope biases (x,y,z)
- Barometer bias (z)

The only measurements made available externally are the attitude in the body frame, and the velocity converted into the NED frame. Additionally, to calculate the ground distance covered, the drone will merge the EKF position estimate with an optical flow estimate from the camera.

3.1.4 Low-level control

The drone includes an inner control loop running at 200 Hz that includes separate PID controllers for altitude, horizontal position, and attitude. The outputs of the internal state estimation are used as the measurement in the control loops, meaning that the position controller will make wind compensations.

3.1.5 Interface

The drone can be accessed and controlled using either the Parrot FreeFlight 6 app[1] on a smartphone, using the Parrot Skycontroller 3[2], or by using the Parrot Olympe application programming interface (API) [65] from an Ubuntu computer. The common communication interface to the drone is Wi-Fi, where the Anafi sets up an access point that the smartphone, controller, or computer connects to. When controlling the drone from a computer through Olympe, using Wi-Fi directly gives a severely limited range and decreases the quality of the signal. To avoid this, the Skycontroller 3 can be connected to the computer and used as a relay point between the computer and drone as seen in Fig. 3.2, which improves the signal strength and quality. Using the Skycontroller 3 as a relay allows for controlling the drone from a computer with the same operating range as when flying it directly with the Skycontroller, which is up to 4 km [66].

The Olympe API provides methods of both commanding and monitoring the Anafi through the use of ARSDK messages. An ARSDK message can be either a *command* or a *response*. The commands are sent to change the drone's behavior, while the responses report back the current state of the drone.

The commands related to motion control are attitude control, relative position control, and global position control. The drone accepts new setpoints for the local and global positions once its internal estimation system considers the current setpoint reached, while new attitude setpoints are always accepted every 50 ms. The attitude setpoints provide the same control as a standard radio controller, namely control of the roll and pitch angles, yaw rotational rate, and throttle. They will default back to stabilized hovering if no setpoint is received after 50 ms. The drone also has build-in commands for takeoff and landing, where the takeoff command makes the drone take off and hover at around 1 m. The land command automatically lands the drone regardless of initial altitude.

[1] https://www.parrot.com/en/apps-and-services
[2] https://www.parrot.com/us/support/anafi/how-does-the-skycontroller-3-work

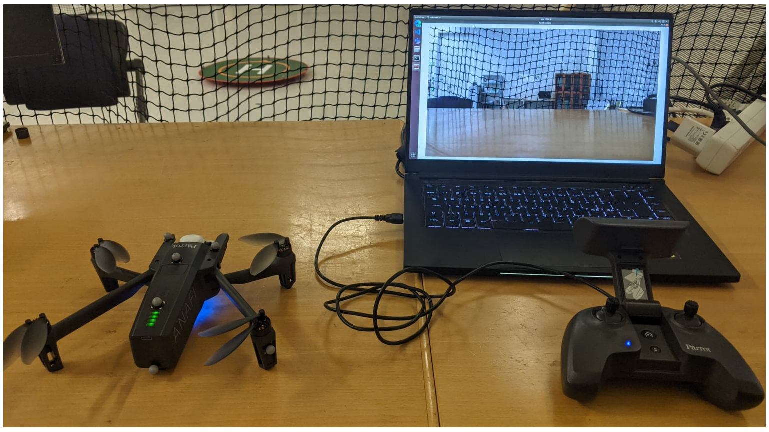


Figure 3.2: Image showing the Anafi connected to the computer using the Skycontroller 3 as a relay.

The motion-related responses are the Anafi's attitude in the body frame, and velocity measurements in the NED frame regularly sent at 5 Hz, as well as the global position from the GPS sensor, regularly sent at 1 Hz. The measurements triggered regularly are not meant for closed-loop command control, and the velocity measurements have been observed to be about 1 s delayed compared to the actual drone velocity. In addition to this, the Olympe API also provides various other information about the drone, such as battery information, gimbal attitude, current flying state, and saturation limits for maximum angles and velocities. The API also provides live access to the Anafi's camera, both for taking discrete images in high resolution and for lower-resolution streaming.

Note that Parrot performed a significant upgrade of all of their software in November 2021, so the version now available online differs from the version used in this project. The version used in this project is Olympe 3.0.0.

3.1.6 License and insurance

From January 1st, 2021, the Civil Aviation Authority (CAA) of Norway has updated the rules and regulations regarding the use of drones, following the new rules from the European Union [67]. These new rules require that the pilot of all drones weighing more than 250 grams, or any drone with a camera, register and receive their A1/A3 certificate and insure the drone. The certificate can be received by following an online course and exam provided by the CAA, and the insurance can be bought as valuables insurance in an insurance company.

3.1.7 Parrot-Sphinx simulator

Parrot-Sphinx [68] is a simulator released by Parrot for its drones and is based on the Gazebo simulator, introduced in [69]. The simulator includes a model for the Anafi drone and a framework for connecting to the drone through a simulated Wi-Fi interface. This means that the only difference when connecting a simulated drone and a real drone is the

IP address the computer connects to, meaning all other functionality is independent of the environment used.

This simulator was also upgraded in November 2021 to include the use of the Unreal Engine to render realistic environments for the drone. As with the new Olympe API, this project does not use this new version of Parrot-Sphinx, and instead uses the previous version, which does not support the use of the Unreal Engine. The version used in this thesis is Parrot-Sphinx 1.8, which is almost identical to the standard Gazebo simulator in the way that it operates.

To simulate the helipad used for takeoff and landing, a model has been provided in [2], and this can be imported into Gazebo. The floor of the simulation is made into a concrete pattern by using a standard model in Gazebo. The overall setup can be seen in Fig. 3.3.

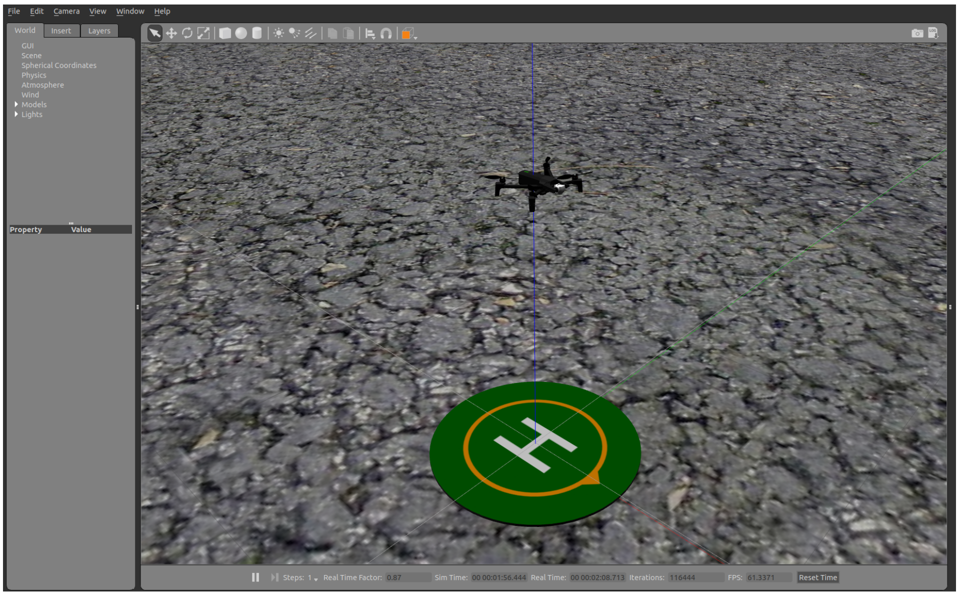


Figure 3.3: Screenshot of Anafi drone hovering above helipad in Parrot-Sphinx simulator.

The simulator provides information about the pose of the different models in the scene through Gazebo topics which can be subscribed to using the shell command-line tool `parrot-gz`, which works identically to Gazebo's `gz`[3] tool. This information can be parsed to extract and save the ground truth pose of the drone and the helipad.

3.2 Landing platform

To be able to land on a boat, a landing platform, hereby referred to as the *helipad*, has been designed and manufactured in [2]. The author also created a Gazebo model of the helipad to be used in simulation, as mentioned above. The physical and simulated helipad can be

[3]`http://manpages.ubuntu.com/manpages/bionic/man1/gz.1.html`

seen in Fig. 3.4. The helipad has a total diameter of 80 cm, with an ”H” in the middle spanning 1/3 of the total diameter on the long side and 1/4 of the total diameter on the short side. It also has a yellow circle with a diameter of 50 cm and an arrow to indicate the orientation of the helipad. The helipad is designed to be mounted on the DNV ReVolt autonomous marine vessel.

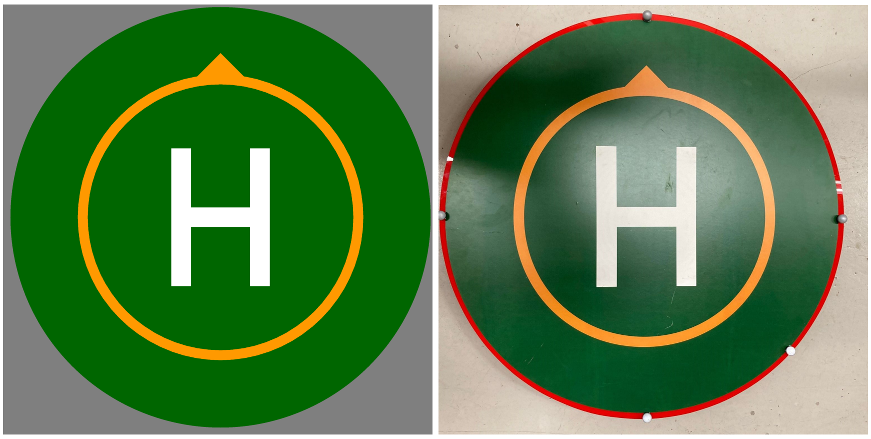


Figure 3.4: Physical and computer model of the helipad.

The features are designed to be distinct in both form and color, which makes the images of the helipad easier to work with in computer vision. The design is minimalistic and resembles the standard design of helipads. The landing pad does not include variable-sized features like the ones used in [20], meaning the drone needs to be at a certain altitude before it can use images of the helipad for pose estimation.

3.3 Physical testing on the ReVolt

A scale model of the proposed autonomous electric ship DNV ReVolt[4] vessel was kindly provided by DNV in Trondheim to use for realistic testing.

The helipad was mounted on top of the ReVolt, as seen in Fig. 3.5, and experiments were carried out both on land and at sea. The testing on land was carried out in an open parking lot with minimal obstacles. The more realistic tests at sea were carried out close to the outlet of river Nidelva in Trondheim, where the exact position can be seen in Fig. 3.6. During the sea tests, the ReVolt was controlled manually by having two people standing in the water and holding it in place.

3.4 Development environment setup

The base platform for the development and experiments has been a Komplett Khameleon laptop computer, whose specifications can be seen in Table 3.2. The manufacturer de-

[4]https://www.dnv.com/technology-innovation/revolt/

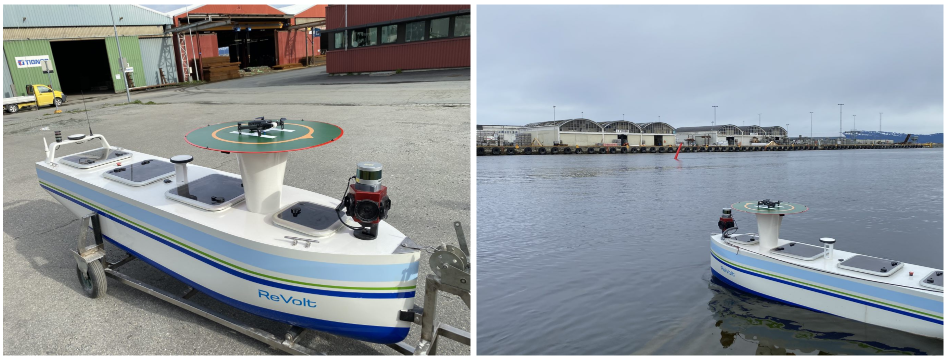


Figure 3.5: Helipad mounted on ReVolt vessel for drytesting and seatrials.

Table 3.2: Computer specifications and software versions used in this thesis.

Computer specifications	
Manufacturer	Komplett
Computer type	Laptop
Model name	Khameleon P9 Pro
CPU	Core i7-9750H
RAM	32 GB
GPU	GeForce RTX 2070
Software versions	
Ubuntu	18.04 Bionic Beaver
Python	3.6.9
ROS	Melodic
Olympe API	3.0.0
Parrot-Sphinx simulator	1.8
Nvidia drivers	495.44
Cuda	10.2

signed the computer to restrict performance severely when not connected to a power source. For this reason, a petrol-driven Honda EX500 generator provided by ITK at NTNU was used when testing outside where there was no access to a power outlet.

Table 3.2 also shows the versions of the different software used in this thesis. As mentioned, the Olympe API and the Sphinx simulator received a major upgrade in 2021, but the versions used in this thesis are the versions prior to this upgrade.

The Olympe API requires the use of Python 3 as well as Ubuntu 18.04 or 20.04, and the version of the Parrot-Sphinx simulator used requires Ubuntu 18.04. The Robot Operating System (ROS) version available for Ubuntu 18.04 is ROS Melodic, which by default uses Python 2. This cross-dependency problem has been avoided by using Python 3 with ROS Melodic, which works out of the box as long as no packages with support only for Python 2 are used. All Python 2 dependent ROS packages have therefore been avoided, meaning that ROS Melodic is compatible with Python 3.

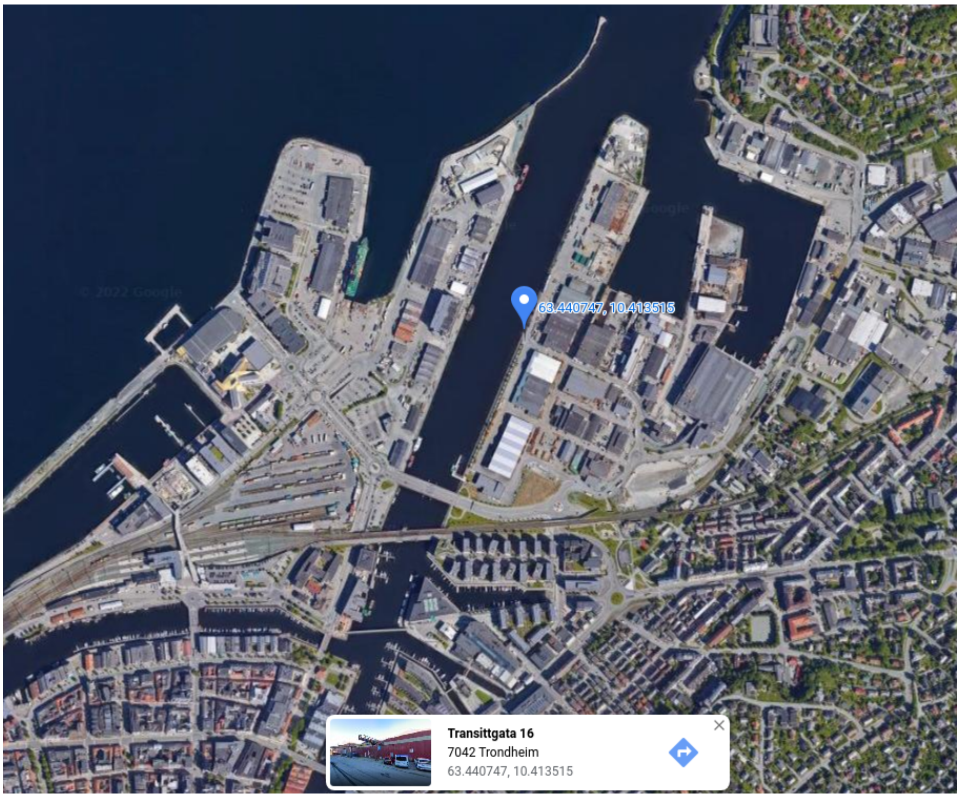


Figure 3.6: Location of seatrials in Trondheim.

Nvidia and Cuda drivers are necessary for running the YOLO object detector with high performance, as the detector utilizes the computer's GPU when possible.

This project uses Git Large File Storage to store all repository dependencies in one place and can be found on Github[5].

3.5 Qualisys motion capture system

Motion capture systems capture the movement of real-world objects and describes them in a 3D virtual environment. There are multiple ways of achieving this, but the state-of-the-art systems today are optical motion capture systems [70]. Marked-based optical motion capture systems use reflective markers that reflect infrared light, and multiple cameras then capture this reflection. By attaching multiple markers to an object, the object's orientation can also be obtained by tracking the motion of all of the markers simultaneously.

One manufacturer of such systems is Qualisys, and their reported accuracy is down

[5]`https://github.com/mfalang/autodrone`

to $1\,\mathrm{mm}$ positional and $0.1°$ for orientation[6]. This means this motion capture system can provide a reliable ground truth pose of the drone and helipad when markers are attached to them. This can then be used, e.g. to validate the results from other pose estimation techniques [20], or to provide reliable feedback when developing a new control system [24].

3.5.1 Marker placement and model creation

A Qualisys motion capture system[7] installed in NTNU's drone lab will be used to get the ground truth pose of the drone and helipad in real-world experiments. Reflective markers are placed at various places on the Anafi drone body and around the edge of the helipad, using smaller markers on the Anafi due to space and weight limitations. The markers are placed asynchronously around the axes of the objects to avoid getting ambiguous poses which could result in incorrect outputs from the system. 9 markers are placed on the Anafi because these smaller markers are harder for the system to detect. They are placed on the top, back, battery, and at the end of each leg, while for the helipad, 5 markers were placed around the edge. The marker placements can be seen in Fig. 3.7.

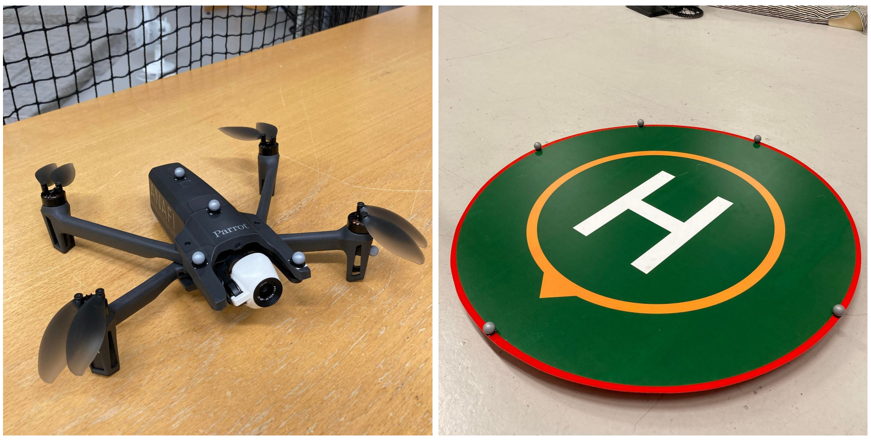


Figure 3.7: Anafi and helipad with attached optical markers.

To set up the motion capture system, the program Qualisys Track Manager[8] is used. Here there is defined a body called ”anafi” for the markers related to the Anafi and a ”helipad” model for the helipad. These objects will then be recognized by the motion capture system each time it sees the given configuration of markers, meaning that the configuration of the markers must remain constant throughout the testing.

[6]https://cdn-content.qualisys.com/2019/09/AN_Cybernetics.pdf
[7]https://www.qualisys.com/
[8]https://www.qualisys.com/software/qualisys-track-manager/

3.5.2 Retrieving the pose

A ROS wrapper around the Qualisys software makes the system output the pose of the objects through ROS topics. The computer with the motion capture system is connected to the same network as the computer controlling the drone, which also starts a ROS core. The motion capture computer is then set up to use the ROS core on the drone computer, meaning the topics will become available here.

As ROS messages are timestamped, it is important that the drone computer and the motion capture computer have synchronized clocks. The Network Time Protocol provides an interface to do this, but the availability of this service has proved to be unreliable on the motion capture computer. Therefore, the measurements are re-timestamped upon arrival at the drone computer to be consistent with the other timestamps coming from the drone computer.

4 Methodology

This chapter explains the work done in this thesis. It starts with a section on the general software development and the anafi_ros package before presenting the work done in computer vision, control, and mission planning. Some intermediary results are presented here instead of in the results chapter as they are not the main focus of this thesis.

4.1 Software development

This section highlights the software development strategy used when developing this thesis on the Parrot Anafi. The foundation for this development was laid in the preceding specialization project [3], and the following contents will summarize the overall system architecture, design philosophy, drone interface, and changes made from the specialization projects.

4.1.1 System architecture

The system architecture used in this thesis can be seen in Fig. 4.1. Here each rounded square represents a ROS node, where the nodes are

- *Drone interface* - Interface exposing Olympe functionality to ROS.
- *Mission planner* - Generating action sequence based on predefined mission goals.
- *Perception* - Estimating drone pose based on images and sensor output.
- *Control* - Overall system for executing an action sequence from the mission planner consisting of the two submodules
  - *Mission executor* - Responsible for executing and monitoring each action in the action sequence.
  - *Guidance* - Moving the drone to a desired position setpoint.

- *Ground truth parser* - Takes ground truth data from either the Sphinx simulator or the motion capture system and converts it into common frames of reference.

- *Output saver* - Saves ground truth and estimation outputs for later use and analysis.

Nodes added in this thesis are the mission planner, ground truth parser, and control system with guidance and mission execution.

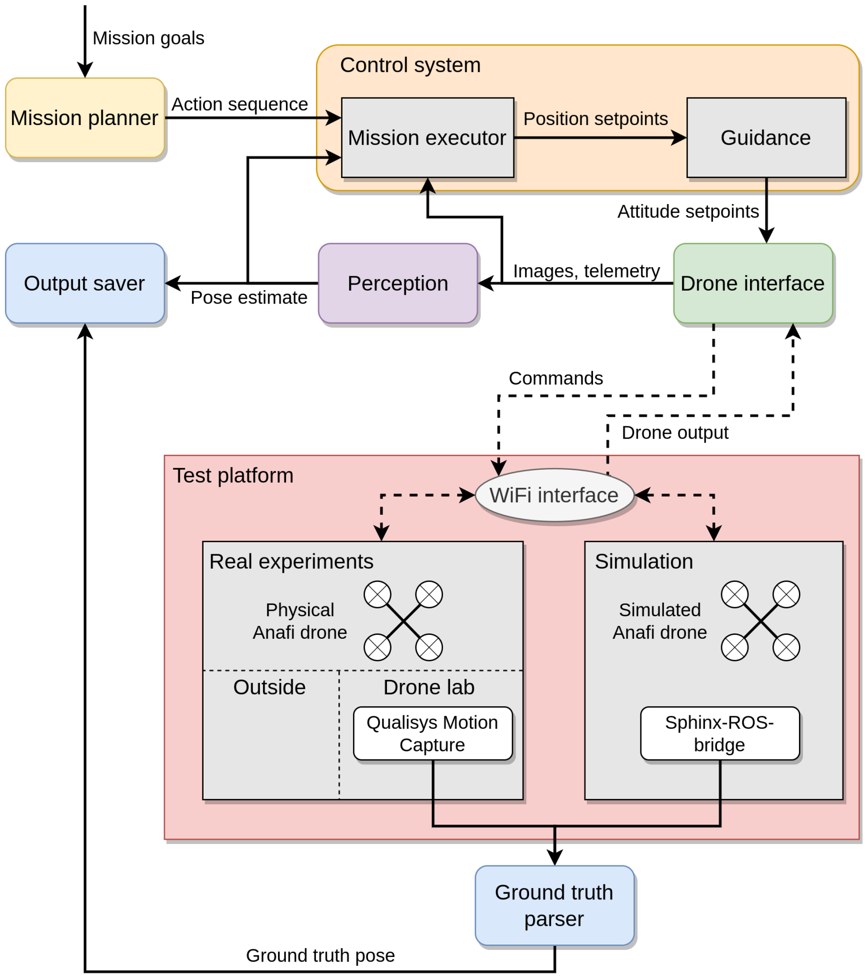


Figure 4.1: Software architecture describing high level interactions between the different modules in the system.

4.1.2 Design objectives

This thesis builds upon the same design objectives used in the preceding specialization project [3] which will be summarized below.

The overall design goal of the software in this thesis is to create a code base that is suitable for further development on the project in the future. Given that this may involve different algorithms, drone platforms, and test environments, the following three design objectives are identified as important to facilitate future development:

1. *modularity*
2. *platform independence*
3. *testability*

Modularity

Modularity here refers to dividing functionality up into separate blocks with standard interfaces. This makes it easier to maintain the code base as it allows for fixing, updating, reworking, or replacing different modules independently without worrying about other parts of the code breaking. In this project, the different components mission planner, control, perception, drone interface, ground truth, and output saver are split into separate ROS packages. This means that e.g., the perception system could be updated without the risk of affecting the control system. One of the design philosophies of ROS is to allow the use of external libraries in ROS by adding lightweight wrappers [71]. The code in this thesis attempts to achieve this by placing all specific ROS code in one file, which is then using the algorithms defined in separate libraries.

Platform independence

Platform independence here refers to developing a code base capable of running on different hardware without having to make modifications to the algorithms used. This independence is essential as the project is likely to transition to using a different drone in the future, where the results from the Anafi must be easily transferable. Platform independence is here achieved by creating a drone interface ROS package that encapsulates all drone-specific information and presents a standardized interface to all the other modules which interface with the drone. All parameters specific to the drone such as mass, camera parameters, and image dimensions, are standardized and published to a ROS parameter server[1] so that they are accessible by the other modules. ROS topics serve as the communication channel between the user and the drone, relaying both commands and responses to and from the Olympe API.

The algorithms used in this thesis can then be used on a new drone by creating a new drone interface using the same standard interface as the other modules. There might be drone-specific details to each drone that makes the drone interfaces have to be somewhat different, but following this philosophy will nevertheless result in code that is as platform independent as possible.

[1]`http://wiki.ros.org/Parameter%20Server`

Testability

Testability here refers to being able to test algorithms in a simulator and real-world experiments without any changes to the algorithms themselves. Although the simulation environment will never be able to replicate real-world conditions fully, it might uncover bugs and mistakes before they reach real-world experiments, reducing the chance of faulty software causing the drone to crash. The drone interface is made to provide a standardized interface regardless of the environment to make it possible to use the same code both in simulation and real-world experiments. In this way, the perception and control algorithms are oblivious to whether they are run in a simulation or a real-world experiment.

Having ground truth measurements to evaluate the performance of the algorithms is essential both in simulation and real-world experiments. For the simulation, an interface between the Sphinx simulator and ROS was created in [3] to extract the ground truth pose. In real-world experiments in the drone lab, these measurements are available through the motion capture system. The output of both of these systems is converted to appropriate reference frames and saved, meaning that the ground truth data from the simulator is directly comparable to that from the motion capture system.

4.1.3 Anafi_ros Olympe ROS bridge

The anafi_ros ROS package is a continuation and improvement of the drone interface created in [3]. The interface has tried to replicate the drone interface [21] used for controlling the AR.Drone 2.0 in the two master's theses preceding this project [1, 2].

When using the Skycontroller as a relay for connection, the interface allows for switching piloting modes from the Olympe API to the Skycontroller. This is used as a failsafe to resume control of the drone when necessary in real-world experiments, making the interface safer to use.

As mentioned in Section 3.1.5, communication with the Parrot Anafi happens through the Olympe API using ARSDK messages, where *command* messages allow changing some state on the drone, while *response* messages allow reading back the current state of the drone. These messages are relayed over ROS topics as discussed above.

The command topics can be seen in Table 4.1, and have been left unchanged from [3]. The Anafi has built-in takeoff and land commands, meaning these topics invoke these commands directly. The topic for setting the attitude is most relevant when performing control and should be supplied with new data at 20 Hz, which is the rate at which the drone accepts new attitude setpoints. The topic for setting the relative position uses the onboard position controller on the Anafi, which is useful for moving short distances. However, it is not suited for real-time control as the drone itself determines when the desired position has been achieved and does not accept new setpoints while executing a movement. Setting the saturation limits can also be done in-flight, which could be useful if different levels of aggressiveness are required for different maneuvers.

[2]Common ROS message, see `http://docs.ros.org/en/lunar/api/std_msgs/html/msg/Empty.html`

[3]Custom message type drone_interface_msgs/AttitudeSetpoint

[4]Custom message type drone_interface_msgs/PositionSetpointRelative

[5]Custom message type drone_interface_msgs/SaturationLimits

Table 4.1: Command topics for ROS Olympe bridge.

Topic name	Message type	Use
takeoff	Empty[2]	Arrival of a message triggers Olympe takeoff command.
land	Empty	Arrival of a message triggers Olympe land command.
set_attitude	AttitudeSetpoint[3]	Attitude setpoint for internal attitude controller ($\phi, \theta, \dot{\psi}, \dot{z}$).
set_position_relative	PositionSetpointRelative[4]	Position setpoint for internal relative postion controller ($\Delta x, \Delta y, \Delta z, \Delta \psi$).
set_saturation_limits	SaturationLimits[5]	Updates the saturation limits for tilt angle, tilt speed, yaw speed and vertical speed.

Similarly, the output topics relaying the response messages from the Anafi can be seen in Table 4.2. Here, contrary to [3], all telemetry data is bundled together as this is updated synchronously at 5 Hz from Olympe. As Olympe provides no information about when new data arrives, the interface regularly polls the latest drone response and checks if the polled velocity differs from the previous velocity published on the topic. Due to there always being some noise, the two following measurements will never be completely the same even when the drone is stationary. The GPS measurements are put in a separate topic, as this response message is only provided from the drone at 1 Hz, where a similar technique as with the telemetry is used to check when a new measurement is available. The image topic is the topic that published the most often, as the drone streams video at up to 30 FPS. An improvement made in this thesis is checking the images for errors using internal error checking of Olympe, and then removing these images from the camera stream. As the images typically include more errors while the drone performs maneuvering, the image topic here publishes less frequently due to more images being rejected.

Table 4.2: Output topics for ROS Olympe bridge.

Topic name	Message type	Use
telemetry	AnafiTelemetry[6]	Telemetry from the Anafi: Drone attitude (ϕ, θ, ψ), body velocity (v_x, v_y, v_z), altitude relative launch (z_r), gimbal attitude (ϕ_g, θ_g, ψ_g), battery information, and flying state.
gps	NavSatFix[7]	Satellite fix status, position (longitude, latitude, altitude), and position covariance.
image_rect_color	Image[8]	Image, image dimensions, and encoding.

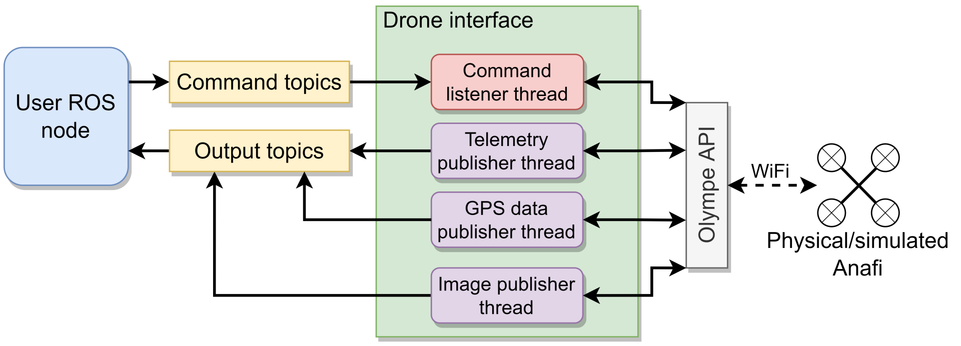


Figure 4.2: Diagram showing the interface between a ROS node and a physical or simulated Anafi drone by the anafi_ros Olympe ROS bridge developed for this thesis.

The functionality of the drone interface is split up into four threads as seen in Fig. 4.2. One thread listens for and relays commands to the drone, while three separate threads relay information about telemetry, GPS measurements, and images. Using multiple threads increases real-time performance and creates a more robust system where the drone will still be able to accept commands if one of the output threads crashes unexpectedly.

4.2 Perception

4.2.1 Traditional computer vision

The traditional computer vision (TCV) part of the perception system attempts to estimate the pose of the drone by solving the perspective-n-point (PnP) problem discussed in Section 2.2.1. The system is based on the work done in the preceding specialization project [3], but addresses some of the shortcomings and issues with the previous system, in particular shortcomings related to corner detection and real-time performance.

The main problem in [3] was that the system was not able to detect the 13 total corners of the helipad ”H” and arrow as the strongest features in the image in real-world experiments. This was concluded was due to jitter in the images, improper tuning of the Shi-Tomasi detector, and no image segmentation to filter out all features outside the helipad. In this thesis, the image jitter problem was severely reduced by using the SkyController 3 as a data relay instead of a direct Wi-Fi interface and removing images based on Olympe's internal error check.

The corner detector is also tuned using a grid search strategy as suggested in [3], and a segmentation system using a circle detector based on the Hough transform is implemented. This thesis uses an OpenCV implementation of the homography-based pose estimation and

[6]Custom message type drone_interface_msgs/AnafiTelemetry

[7]Common ROS message, see `https://docs.ros.org/en/api/sensor_msgs/html/msg/NavSatFix.html`

[8]Common ROS message, see `http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Image.html`

subsequent optimization to increase the real-time performance, as this implementation is more optimized than the solution in [3]. Each improvement is further discussed below.

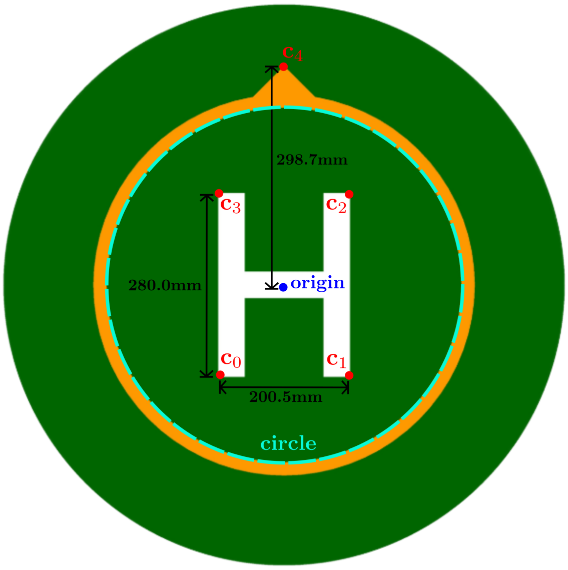


Figure 4.3: Helipad with metric distances and corner definitions.

Helipad segmentation

Segmenting the image to only contain the helipad is important to ensure that no features outside the helipad are incorrectly detected. In the specialization project, this was attempted using green color segmentation, but this was unsuccessful due to the edges of the segmented image having too much noise [3]. Color segmentation was also used in the first master's thesis written on this project [2], but as shown in the next master's thesis, this proved to be too sensitive to varying lighting conditions making it unsuitable for flying outside [1]. Due to these shortcomings, this thesis will segment the helipad based on the Hough transform circle detector. The OpenCV implementation `HoughCircles`[9] is used to detect the circles, where the image is first converted to grayscale.

The system will attempt to detect the circle in the inner part of the orange circle of the helipad, marked in cyan in Fig. 4.3. This circle is used as it is the only complete circle on the helipad, as the outer rim of the helipad has motion capture markers placed around it. Segmenting out an image only containing the corners $c_0, ..., c_4$ can therefore be done by creating a mask from this circle and increasing the radius by 40% to include the arrow corner as well. Increasing the radius can safely be done as the overall helipad radius is about 60% larger than the yellow circle radius.

[9] `https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d`

Using such a strategy was attempted in the specialization project, but the parameters for the `HoughCircles` function giving the desired circle detected were never found [3]. However, in this thesis, this problem is automated using a grid search.

The grid search algorithm requires defining a cost function $\mathbf{V}_{circle}$, which in turn requires a measure to evaluate whether the detected circle is the correct one. Images from flying in the drone lab are therefore labeled with the correct circles using a program developed to store the center and radius of the labeled circle for each image. A total of 151 images with the drone flying at different altitudes and orientations were labeled, where two such images from the labeling process can be seen in Fig. 4.4.

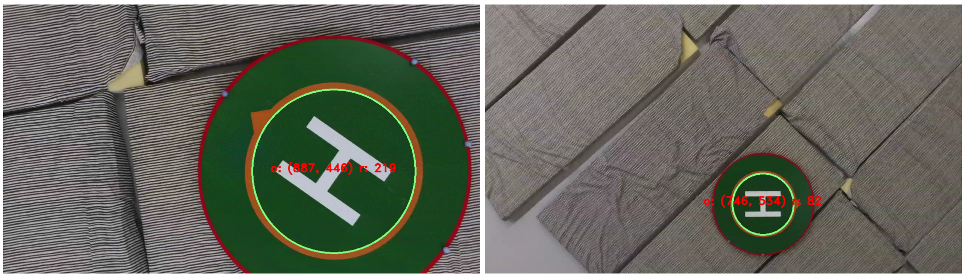


Figure 4.4: Two sample images labeled with the ground truth circle.

Given the ground truth and estimated circle center $\boldsymbol{c} = \begin{bmatrix} x & y \end{bmatrix}^\mathsf{T}$ and $\hat{\boldsymbol{c}} = \begin{bmatrix} \hat{x} & \hat{y} \end{bmatrix}^\mathsf{T}$, and radius r and $\hat{r}$, the estimation error of a single image is defined to be the distance between the centers plus the absolute value of the radius error

$$e_{circle} = \|\boldsymbol{c} - \hat{\boldsymbol{c}}\| + |r - \hat{r}|. \tag{4.1}$$

The overall cost function $\mathbf{V}_{circle}$ for all the images is defined as the mean of the error across each image

$$\mathbf{V}_{circle} = \frac{1}{N} \sum_{i=1}^{N} e_{circle,i}. \tag{4.2}$$

Grid search is performed using the Scikit-Learn library for machine learning in Python [72]. Although there are several more advanced hyperparameter optimization (HPO) methods available, grid search is chosen because of its simplicity.

Although there has been added functionality for testing all the parameters used in the `HoughCircles` function and using different types of blur, only some of the parameters are searched due to the exponential complexity increase when adding more parameters to the search grid. In addition, the runtime of the `HoughCircles` function is highly dependent on the combination of parameters, meaning testing lots of parameters is not feasible. All the blur parameters have been fixed, meaning the images are blurred using Gaussian, Median and Bilateral blur.

Some of the `HoughCircles` parameters have also been fixed as these can be determined based on the specific problem. These fixed parameters include the minimum and maximum radius of the circles detected, set to 50 and 500 pixels, respectively, accounting for the approximate smallest and largest circles where all 13 corners can be visible and detectable in the image. Similarly, the minimum distance between two circles has been set to 1000 pixels to ensure only one circle is detected in the image. The parameter determining the inverse ratio of the accumulator resolution to the image resolution has also been fixed at 1 as it seemed to have little effect on the performance.

The changed parameters are two method-specific parameters, where one is a part of the internal Canny edge detector used by the Hough circle transform, and the second is a parameter determining the voting threshold of the accumulator threshold. These parameters are by OpenCV referred to as ”param1” and ”param2” which they will be referred to in this thesis. The values tried searched for both param1 and param2 are $[20, 30, 40, 50, 60, 70, 80, 100]$. These values resulted in 64 different combinations to try for all of the images when training, which totaled 105 images. The test set used in the grid search is then the remaining 46 images.

Corner detection

Corner detection is here used to detect the 12 corners of the helipad ”H” and the arrow, to then be able to later identify the four outer corners of the helipad ”H” and the arrow, marked in red in Fig. 4.3. Different feature detectors have been tested, such as the FAST and SIFT feature detectors, but the Shi-Tomasi and Harris detectors from OpenCV's `goodFeaturesToTrack` function[10] are used in this thesis. Although FAST and SIFT, and FAST in particular, have better real-time performance, the `goodFeaturesToTrack` function has the advantage that it can return the n strongest features detected. This is particularly useful in this project as it is beneficial to only detect the corners of the helipad ”H” and arrow to then determine which detected feature correspond to which corner $c_0, ..., c_4$.

The detector is tuned using a grid search strategy as suggested in [3]. The corner detector uses a labeling method similar to the circle detector. All 13 corners are labeled and saved with the correct coordinates in each image, where two such images can be seen in Fig. 4.5. The same dataset of 151 images is labeled, and each image is segmented to include only the corners and the arrow using the circle detector described above.

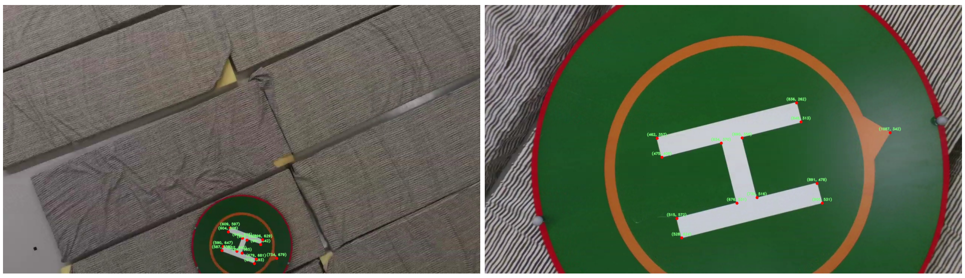


Figure 4.5: Two sample images from corner labeling.

When the detector returns an estimate including 13 different points, the 13 points are matched to the 13 ground truth points using the k-nearest neighbor classifier from Scikit-

[10]`https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541`

Learn[11] with $k = 1$. The error for a single image is then defined as the average error between each pair of corresponding ground truth points $\boldsymbol{p} = \begin{bmatrix} x & y \end{bmatrix}^\intercal$ and estimated points $\hat{\boldsymbol{p}} = \begin{bmatrix} \hat{x} & \hat{y} \end{bmatrix}^\intercal$

$$e_{corner} = \frac{1}{13} \sum_{i=1}^{13} \|\boldsymbol{p}_i - \hat{\boldsymbol{p}}_i\| \,. \tag{4.3}$$

The overall cost function $\mathbf{V}_{corner}$ across all the images is similarly as for the segmentation defined as the mean of the errors across each image

$$\mathbf{V}_{corner} = \frac{1}{N} \sum_{i=1}^{N} e_{corner,i}. \tag{4.4}$$

The problem of exponentially increasing computational complexity is present here as well. However, luckily the `goodFeaturesToTrack` function uses more or less the same time computing features based on all the different sets of parameters. The function also includes a parameter for choosing which of the Harris and Shi-Tomasi detector is used. Most of the parameters are searched, except for the maximum number of corners, fixed at 13 since this is the number of corners desired, and the free parameter k in the Harris detector as it seemed to make little impact on performance and was fixed to reduce computational complexity. For more information about the parameters in `goodFeaturesToTrack`, see the OpenCV documentation[12]. The images were also blurred a fixed amount using a Gaussian filter.

This results in a total of 4620 parameters to search. The training dataset is reduced compared to when finding the segmentation parameters as each ground truth image here also has to be accompanied by a mask to segment out the helipad, meaning the memory usage is increased. To avoid memory problems, the overall size of the training set is reduced to 37 images, while the test dataset consists of the remaining 114 images.

Corner identification

Given that the 13 corners found by the corner detector are the correct ones, it must be identified which corners correspond to the corners of the helipad "H" and the arrow.

The general algorithm developed for identifying the point correspondences between the image and the 3D points seen in Algorithm 1 was developed in the preceding specialization project and is left conceptually untouched in this thesis. The algorithm is based on using the known geometry and metric distances of the helipad to identify the corners $c_0 ... c_4$, defined as seen in Fig. 4.3. Assuming the corner detector returns the 13 corners corresponding to all 12 corners on the "H" and the one corner on the arrow, the distance between all the points can be used to identify the points. These distances are stored in a 13×13 matrix. The largest distances can be seen to be between c_4 and c_0, and between c_4 and c_1, and this can be used to identify the arrow and the two lower corners of the "H".

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

[12] `https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541`

If these distances are not the same, the 13 corners are not correct, and no correspondences can be found.

The arrow corner can be uniquely identified from the two largest distances since it will appear in both distances. The other two corners can then be identified by computing where they are in relation to the arrow (if they are below, above, left, or right) and then comparing their x or y pixel values.

Having identified the two bottom corners of the ”H”, the two top ones c_2, c_3 can be identified by calculating the expected distance to them from c_0. Knowing the pixel and the metric distance between c_0 and c_1 from the pixel coordinates and metric measurements in Fig. 4.3, the ratio between pixel distances in the image and metric distances in 3D can be calculated. Using this ratio, the expected pixel distance between c_0 and c_2 and c_0 and c_3 can be calculated. The points c_2 and c_3 can then be found from the matrix of distances by looking up the expected distances in the distances from c_0 and extracting the point indices.

This approach then identifies 5 point correspondences in the image, which is enough to solve the PnP problem.

Algorithm 1 Finding corners of ”H” and arrow based on corners from corner detector. Algorithm developed in [3].

```
1: Let $\boldsymbol{u}$ be pixel coordinates of corners found from corner detector (shape $13 \times 2$)
2: Let $\mathbf{X}$ be 3D coordinates of the points to identify on the helipad (shape $5 \times 2$)
3: procedure IDENTIFYCORNERS($\boldsymbol{u}, \mathbf{X}$)
4:     Let $\mathbf{D}$ be a new $(13 \times 13)$ matrix
5:     Compute distance between all points in $\boldsymbol{u}$ and store in $\mathbf{D}$
6:     Find corners corresponding to the 2 largest distances $d_x, d_y$ in $\mathbf{D}$
7:     if $d_x \neq d_y$ then                                  ▷ Consistency check
8:         return $\emptyset$
9:     end if
10:    Find $c_4$ as corner appearing in both $d_x, d_y$                 ▷ Arrow corner
11:    Let $c_x, c_y$ be the other two remaining corners appearing in $d_x, d_y$
12:    Identify $c_0, c_1$ based on location of $c_x, c_y$ w.r.t. $c_4$
13:    Let $d_{px}, d_m$ be the pixel and metric distance between $c_0$ and $c_1$
14:    $r \leftarrow \frac{d_{px}}{d_m}$               ▷ Ratio between pixel distance and metric distance
15:    Find expected distance $d_{02}, d_{03}$ using $r$ and metric distances $\mathbf{X}$
16:    Identify $c_2, c_3$ as the corners corresponding to $d_{02}, d_{03}$ in $c_0$-row of $\mathbf{D}$
17:    Find $d_{03}, d_{12}$ from $\mathbf{D}$
18:    if $d_{03} \neq d_{12}$ then                                ▷ Consistency check
19:        return $\emptyset$
20:    end if
21:    return $\{c_0, c_1, c_2, c_3, c_4\}$
22: end procedure
```

Pose estimation

Given the 5 point correspondences between the pixel coordinates and 3D distances of the outer corners of the helipad ”H” and the arrow, the pose of the camera relative to the helipad can be computed by solving the PnP problem. One of the main issues in the preceding specialization project was that the runtime of the homography and optimization algorithms were slow due to these being custom made [3]. Especially the Levenberg-Marquardt (LM) optimization was slow, taking around 350 ms to improve the pose estimate. To improve the runtime, the more optimized OpenCV implementation is used in this thesis.

The OpenCV function `solvePnP`[13] was used with the `SOLVEPNP_ITERATIVE` flag set. Using this flag means that the function will create an initial pose estimate by solving the PnP problem and then optimize this estimate using LM optimization. The implementation of `solvePnP` also uses the homography-based solution to the PnP problem, given that the 3D points are planar. This means that this function works similarly to the custom implementation made in the specialization project with homography and LM optimization, meaning the results should be comparable, and the rest of the code used will still be compatible.

4.2.2 Deep neural network-based computer vision

In [1], the author developed a pose estimation system using a deep neural network (DNN) based pose estimation system using the YOLO object detector. This system has been adapted to be compatible with the rest of the code in this thesis, but the algorithms themselves have been largely left unchanged. The same DNN-based computer vision (DNN-CV) system was used in the preceding specialization project, and therefore this section is largely based on the same section from [3].

Object detection

Object detection is performed using YOLO v4 as in [1], with the same trained weights and network parameters supplied by the author. Different weights are used for the simulation and the physical Anafi, and no further training of the model is done for either case.

The backbone used for the YOLO v4 detector is the Darknet neural network framework [5], which has been made available as a ROS package `darknet_ros` [4]. The ROS package used in this project is a fork of [4] that has been modified to support YOLO v4, available on Github[14].

The detector is trained to recognize three different classes: the overall helipad, the ”H”, and the arrow. Each detection is returned as a bounding box containing the pixel coordinates of the lower-left and top-right corner of the bounding box rectangle encapsulating the object.

[13]`https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d`

[14]`https://github.com/tom13133/darknet_ros`

Position estimation

The bounding boxes are used to determine the pixel coordinate of the center of the helipad, ”H”, and arrow. The radius of the helipad is also estimated using the bounding box of the entire helipad. This estimated center and radius are then used to estimate the drone’s position.

The algorithm for estimating the drone position relative to the helipad can be seen in Algorithm 2. Knowing the center and radius of the helipad, the drone’s position can be found by using the similar triangles that arise from using the pinhole camera model. Here the height of the drone relative to the helipad can be found first by using the metric and pixel radius of the helipad and the camera focal length. When this height is known, it can then be used in a similar way to find the horizontal positions as well. This algorithm was originally proposed in [2] and later used in [1].

Algorithm 2 Estimate position from DNN-CV detections. Algorithm based on [1], and originally proposed in [2].

```
1:  Let $c_x, c_y$ be the pixel position of the helipad center
2:  Let $r, R$ be the helipad pixel radius and metric radius respectively
3:  procedure EstimatePosition($c_x, c_y, r, R$)
4:      Let $f$ be the camera focal length (in pixels)
5:      Let $o_x, o_y$ be the pixel coordinates of the image center
6:      Let $r_x, r_y, r_z$ be the camera offset from center of orientation in meters
7:      $\hat{z} \leftarrow \frac{Rf}{r}$                          ▷ Estimate $z$ using similar triangles
8:      $d_x \leftarrow o_x - c_x$
9:      $d_y \leftarrow o_y - c_y$
10:     $\hat{x} \leftarrow -(\frac{\hat{z}d_x}{f} + r_x)$          ▷ Estimate $x$ using similar triangles
11:     $\hat{y} \leftarrow -(\frac{\hat{z}d_y}{f} + r_y)$          ▷ Estimate $y$ using similar triangles
12:     $\hat{z} \leftarrow \hat{z} + r_z$
13:     return $\hat{x}, \hat{y}, \hat{z}$
14: end procedure
```

4.2.3 Model-based Kalman Filter

This section explains the implementation of a Kalman filter (KF) for estimating a reliable position estimate by fusing the measurements from the TCV and DNN-CV systems, the sensor output from the Anafi, and a dynamic model. The KF used in the master thesis previously written on this project [1] is not used because the code was specific to the AR.Drone 2.0’s sensors and interface.

This thesis aims to use a model-based KF to estimate the *helipad position* relative to the *drone body frame*. The reasoning behind this unorthodox choice of coordinate frames is that this makes guidance easier when tracking the helipad to prepare the drone for landing, as then the position estimate from the KF can be directly interpreted as the position error.

The need for a model-based filter arises as the internal EKF on the Anafi only outputs attitude and velocity measurements at 5 Hz, but the internal control system accepts setpoints at 20 Hz, as discussed in Section 3.1.5. The rate of the KF is set to 25 Hz to make sure it is faster than the control system, meaning a recent position estimate will always be available for feedback in the control system.

States

The filter used in this thesis is kept as simple as possible to reduce the development complexity to allow for faster real-world experimental testing of the complete system. Therefore, the only states included are the helipad position relative to the drone body frame $\boldsymbol{p}_h^b$ and the drone velocity in the body frame $\boldsymbol{v}_d^b$. Thus defining the state vector $\boldsymbol{x}$ to be

$$\boldsymbol{x} = \begin{bmatrix} x_h^b & y_h^b & z_h^b & v_{d,x}^b & v_{d,z}^b & v_{d,z}^b \end{bmatrix}^\mathsf{T} . \tag{4.5}$$

Constant velocity model

The dynamic model used in the filter is the constant velocity (CV) model. The CV model assumes that the velocity does not change between samples, meaning the velocity at the previous timestep is used for predicting the motion until the next measurement is available. This model might quickly diverge for a drone performing intensive maneuvers but could yield adequate accuracy given that the maneuvers are slow and the model is corrected frequently enough. When landing the drone in this thesis, slow speeds are desired to achieve a controlled landing, and with corrective measurements arriving at 5 Hz, it is reasonable to believe that this simple model is accurate enough.

The equations of motion describing the constant velocity model in 3 dimensions are based on the equations for the planar case found in [54], and can be summarized as follows

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{G}\boldsymbol{n}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{0}_{3\times3} & -\mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}, \mathbf{G} = \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} \end{bmatrix}, \tag{4.6}$$

where $\boldsymbol{n}$ is process noise which is assumed to be white noise. The white noise is characterized by a diagonal covariance

$$\mathbf{D} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}, \tag{4.7}$$

where $(\sigma_x, \sigma_y, \sigma_z)$ represent the amount of acceleration the drone is expected to undergo in each of the three axes.

Measurements

The measurements used as corrections in the KF are:

- Drone body velocity $\boldsymbol{v}_d^b$ from Anafi internal EKF.
- Helipad position $\boldsymbol{p}_h^b$ from TCV system.
- Helipad position $\boldsymbol{p}_h^b$ from DNN-CV system.

As seen, the measurements arrive from three different sources. As these arrive at independent frequencies, they need individual measurement matrices to independently update the filter based on each measurement. The measurement matrices are based on the states described above and can be summarized as follows

$$\mathbf{H}_{telemetry} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{I}_3 \end{bmatrix} \tag{4.8}$$

$$\mathbf{H}_{tcv} = \mathbf{H}_{dnncv} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3\times 3} \end{bmatrix}. \tag{4.9}$$

Tuning

The tuning parameters used in the KF are shown in Table 4.3. The initial position is zero, and the covariance is small as it is assumed that the drone always starts at rest in the middle of the helipad. The uncertainty parameters were chosen based on trial and error, where the given parameters resulted in a combined estimate where no single measurement dominated the others.

Table 4.3: Kalman filter initial values and tuning parameters.

	Initial position					
	x	y	z	v_x	v_y	v_z
$\boldsymbol{x}_0$	0	0	0	0	0	0
	Initial uncertainty					
	σ_x	σ_y	σ_z	σ_{v_x}	σ_{v_y}	σ_{v_z}
$\mathbf{P}_0$	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	0.03	0.03	0.03	0.01	0.01	0.01
	Measurement uncertainty					
Measurement	σ_x	σ_y	σ_z	$\sigma_{v,x}$	$\sigma_{v,y}$	$\sigma_{v,z}$
Velocity	-	-	-	0.4	0.4	0.4
TCV position	0.5	0.5	0.5	-	-	-
DNNCV position	0.5	0.5	1	-	-	-

4.3 Coordinate frames

This section will briefly describe the coordinate frames used in this thesis and the conversions between them. The different frames are the NED frame, helipad frame, camera frame, and the drone body frame, illustrated in Fig. 4.6. The coordinate frames associated with each system in this thesis are summarized in Table 4.4. The helipad frame is not used by any of the algorithms that ended up in the final part of this thesis. However, it is a frame that can be useful for local navigation when landing, and is left in the thesis as a future reference.

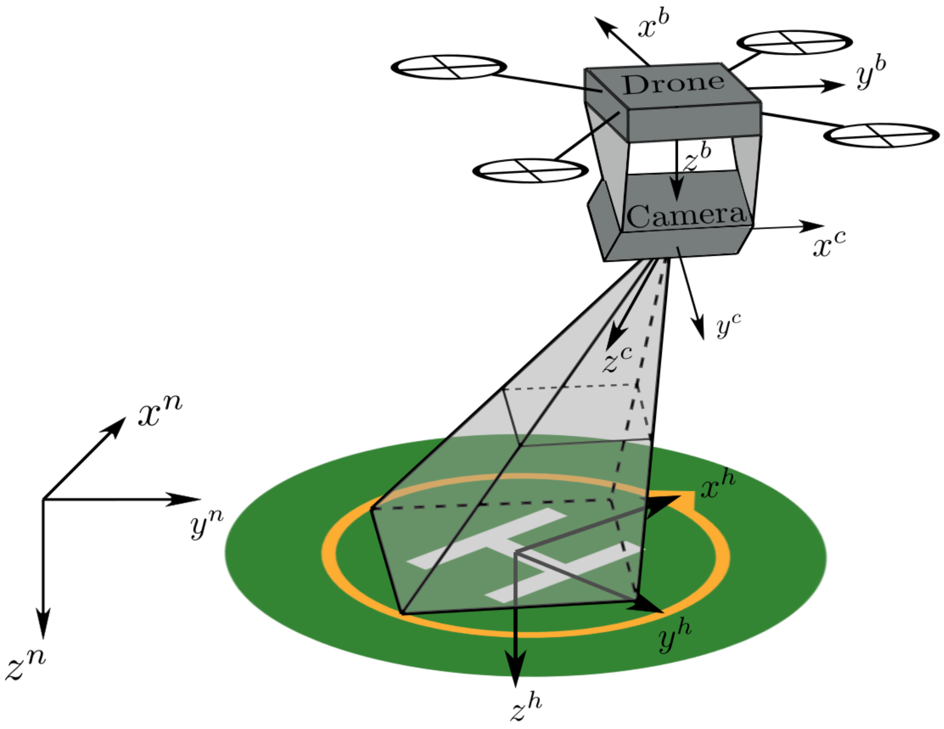


Figure 4.6: Coordinate frames NED $\{n\}$, helipad $\{h\}$, drone body $\{b\}$, and camera frame $\{c\}$.

4.3.1 Conversions

Although converting coordinates between the different frames, in general, involves the complete rigid-body transformation described in Section 2.1, several factors simplify the transformations in this thesis. Each conversion will therefore be described below.

NED to helipad frame

Translating from the NED frame to the helipad frame makes it easier to monitor and visualize the drone's position relative to the helipad. This conversion involves a translation $\boldsymbol{t}_n^h$ between the origins of the frames, as well as a rotation $\mathbf{R}_n^h$ based on the attitude of the helipad in the NED frame. If the helipad is assumed to be level with the ground at all times, the rotation reduces to a rotation around the helipad yaw angle only. The general expression for a point in the helipad frame can then be given as

$$\boldsymbol{p}^h = \mathbf{R}_{z,\psi_h}\boldsymbol{p}^n + \boldsymbol{t}_n^h. \tag{4.10}$$

Table 4.4: Table showing the coordinate frame outputs of different system used in the thesis.

System	Output reference frame
TCV pose estimates	camera $\{c\}$
DNN-CV pose estimates	camera $\{c\}$
Kalman filter	body $\{b\}$
Motion capture system	NED $\{n\}$
Parrot-Sphinx simulator ground truth	NED $\{n\}$

Camera to drone body frame

As the position estimates coming from the TCV and DNN-CV systems are defined in the camera frame, they must be transformed into the body frame before they are used in the Kalman filter. Converting from the camera frame to the body frame involves a rotation around the camera gimbal attitude relative to the body frame $\mathbf{R}_c^b$, and a translation between the camera frame origin and the body frame origin. In this thesis, the camera gimbal is set to always point straight down along the z-axis of the drone body frame, and this is internally controlled by the Anafi. Therefore, it is assumed that this alignment is valid throughout the flying, which simplifies the rotation between the two frames to a single rotation of 90° counterclockwise around the z-axis, as can be verified from Fig. 4.6. The conversion thus becomes

$$\boldsymbol{p}^b = \mathbf{R}_{z,90}\boldsymbol{p}^c + \boldsymbol{t}_c^b \tag{4.11}$$

where $\boldsymbol{t}_c^b$ is the camera offset along each body axis, estimated to be around 70mm in the x-axis and 0mm in the y- and z-axis.

NED to drone body frame

Converting from NED to body is necessary to convert the ground truth data from the simulator and motion capture system to the body frame in order to evaluate the state estimation which is handled in the body frame. The transformation from NED to body is defined as a full rotation around the drone attitude expressed in the NED frame $\mathbf{R}_n^b = \mathbf{R}_{\boldsymbol{\Theta}}$, as well as a translation between the origin of the NED frame and the drone $\boldsymbol{t}_n^b$.

$$\boldsymbol{p}^b = \underbrace{\mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi}}_{\mathbf{R}_n^b}\boldsymbol{p}^n + \boldsymbol{t}_n^b. \tag{4.12}$$

4.4 Guidance and control

A guidance system is needed to align the drone with the helipad while landing. The limitations of the Anafi onboard position control system discussed in Section 4.1.3 necessitate the need for a custom guidance system based on the Anafi's internal attitude control system instead. The final guidance system will consist of three parts: (i) the Anafi internal attitude controller, (ii) a velocity controller, and (iii) a guidance system.

The velocity controller will be responsible for converting a 3D velocity reference into the attitude commands for the internal attitude controller. Similarly, one level up, the guidance system will generate a velocity reference for the velocity controller based on the current target location. The overall structure of the entire guidance and control system can be seen in Fig. 4.7.

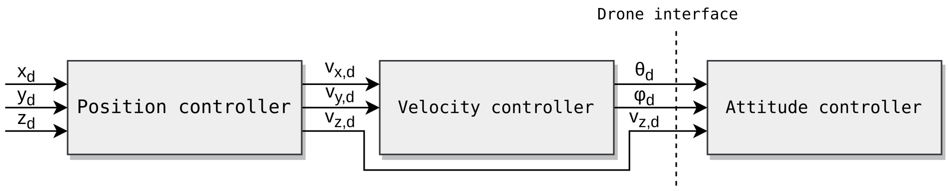


Figure 4.7: Control system architecture showing the layered structure of the overall guidance system. The feedback signals for position and velocity control have been omitted for the sake of simplicity.

4.4.1 Velocity control

As the attitude controller takes desired roll ϕ_d, pitch θ_d, yaw rate $\dot{\psi}_d$, and climb-rate $v^b_{z,d}$, the velocity controller must generate ϕ_d and θ_d from the desired horizontal velocities $v^b_{x,d}$ and $v^b_{y,d}$, while no action is needed for the vertical velocity $v^b_{z,d}$ as this is already an input to the control system. The drone will move using roll and pitch only, meaning no action is necessary for the yaw rate.

Two methods are proposed for generating the horizontal references for the Anafi's internal attitude controller: (i) a model-based method and (ii) a PID controller. Both systems are evaluated by viewing the drone's response in simulation and real-life experiments when applying the varying step response smoothed by a second order velocity reference model as the input. The reference input can be seen in Fig. 4.8. A limit of $\pm 5°$ is set on both roll and pitch to avoid aggressive maneuvers.

Model-based

As the Anafi provides no way to directly determine the thrust force of the motors, it is here assumed that the drone is flying in level flight to be able to determine this force analytically, as suggested in the specialization project preceding this thesis [3]. During level flight, the drone does not move vertically, meaning that the total vertical force F_z produced by the drone's engine must be equal and opposite the gravitational force G acting on the drone. Given information about the drone's current roll and pitch attitude, the horizontal forces F_x and F_y produced by the drone can be derived. Combining this with linear drag forces in both horizontal axes D_x and D_y yields the forces in each axis. The forces acting on one of the horizontal axes during level flight can be seen in Fig. 4.9. Due to the symmetrical rotor configuration of a quadcopter, the roll and pitch dynamics will be decoupled, meaning their dynamic equations can be developed separately. The general dynamic equation for an axis i and angle α, $(i, \alpha) \in [(x, \theta), (y, \phi)]$, can be computed trigonometrically as follows

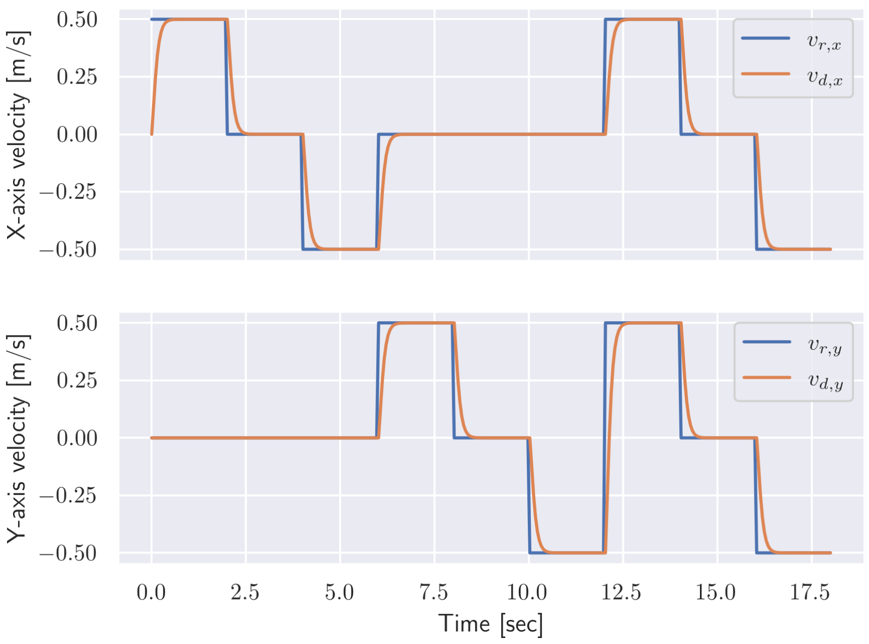


Figure 4.8: Horizontal velocity references used to evaluate velocity controllers.

$$\begin{aligned} ma_i^b &= \sum_j F_j = F_i - D_i && (4.13)\\ &= mg\tan\alpha - D_i && (4.14)\\ &= mg\tan\alpha - d_i v_i^b && (4.15)\end{aligned}$$

where a_i^b is the body acceleration in axis i, d_i is the drag coefficient for the linear drag model in axis i, v_i^b is the body velocity in axis i, m is the drone mass, and g is the gravitational acceleration.

The acceleration a_i^b is where the information about the desired velocity will come in. By viewing this as the *desired* acceleration of the system and then discretizing it assuming the timestep $\Delta t = 1\,\mathrm{s}$, the desired acceleration becomes

$$a_i^b \approx \frac{\Delta v_i^b}{\Delta t} = \frac{v_{i,d}^b - v_i^b}{1} = v_{i,d}^b - v_i^b := \Delta v_{i,d}^b. \tag{4.16}$$

Plugging this into Eq. (4.15) and solving for the given angle $\alpha \in [\theta, \phi]$ gives the following equations for ϕ_d and θ_d

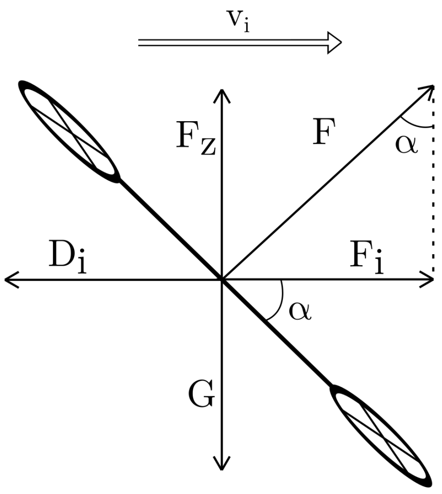


Figure 4.9: Forces acting on the quadcopter along the axis i. The length of the arrows are chosen to indicate a level flight at constant velocity.

$$\theta_d = \arctan\left(\frac{1}{mg}(mv^b_{x,d} + d_x v^b_x)\right) \tag{4.17}$$

$$\phi_d = \arctan\left(\frac{1}{mg}(mv^b_{y,d} + d_y v^b_y)\right). \tag{4.18}$$

The coefficients d_x and d_y in the linear drag model from [23] are unknown in this equation, and therefore need to be estimated. The unknown parameters can be estimated by flying the drone and recording the acceleration and velocity it experiences for each attitude. As only velocity measurements are available for the motion capture system, the acceleration is obtained by differentiation and then smoothed out using a Savitzky-Golay filter[15] to remove noise caused by differentiation. After sampling sufficiently many acceleration, velocity, and attitude correspondences, the coefficients are estimated using least-squares optimization from SciPy[16].

Here the residual function r is defined as the error between the estimated and actual attitude angles given a certain velocity and acceleration. This can be viewed as evaluating how close the calculated desired attitude for a given velocity is to the actual attitude that achieves that velocity.

$$r(d_i, a_i, v_i, \alpha) = \hat{\alpha}(d_i, a_i, v_i) - \alpha. \tag{4.19}$$

This procedure is done for both the x-axis where $\alpha = \theta$, and the y-axis where $\alpha = \phi$. The same dataset is used to estimate both parameters, and this is recorded while flying the

[15] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html`

[16] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html`

drone horizontally in both axes. The resulting estimated drag coefficients found from the least-squares optimization can be seen in Table 4.5. The y-component can be seen to be slightly larger than the x-component, as more of the drone body is directly exposed to the wind in the y-axis than the x-axis due to the Anafi's slender shape.

Table 4.5: Resulting linear drag coefficients from least-squares optimization.

Estimated linear drag coefficients	
$\hat{d}_x$	0.08063504
$\hat{d}_y$	0.09929089

Once the drag coefficients d_x and d_y are estimated, the desired attitude can be computed for a given velocity, hence completing the model.

PID controller

The second proposed method for computing the desired attitude given a desired velocity is using a PID controller. Given the internal attitude controller of the drone, this velocity control system will follow the successive loop closure methodology. Defining the errors between the desired and actual velocity along the two axes to be

$$\begin{aligned} e_x &= v^b_{x,d} - v^b_x \\ e_y &= v^b_{y,d} - v^b_y, \end{aligned} \tag{4.20}$$

the control law for the x and y-axis will be respectively

$$\begin{aligned} \theta_d &= K_{p,x} e_x + K_{i,x} \int_0^t e_x + K_{d,x} \dot{e}_x \\ \phi_d &= K_{p,y} e_y + K_{i,y} \int_0^t e_y + K_{d,y} \dot{e}_y. \end{aligned} \tag{4.21}$$

As the Anafi does not expose the raw data from its internal IMU, no information about the drone's acceleration is directly retrievable. Therefore, the derivative of the error $\dot{e}_i$ has to be obtained by differentiating the difference between the current and previous error with respect to the timestep of the controller, which is $0.05\,\mathrm{s}$ due to the control system running at $20\,\mathrm{Hz}$. If the error contains noise, then this noise will be amplified when differentiating it, which could be solved by introducing filtering techniques such as the ones discussed in [73], but this is not investigated in this thesis. Similarly, numerical integration is performed to compute $\int_0^t e_i$, where the integral is approximated as a sum of the previous errors. The differentiation and integration techniques for axis i are therefore defined as

$$\dot{e}_i \approx \frac{e_{i,k} - e_{i,k-1}}{\Delta t} \tag{4.22}$$

$$\int_0^t e_i \approx \sum_{j=0}^{k} e_{i,j} \Delta t. \tag{4.23}$$

The PID gains used for the controller can be seen in Table 4.6. These were found by trial and error where the goal was to obtain a steady response that was sufficiently fast (the results will be shown in Chapter 5). The controller used was a PI controller with a small integral gain to help reach the setpoint in the case of disturbances or unmodeled dynamics. The error integral was saturated at the same values as roll and pitch to avoid instability due to integrator windup in the controller. The derivative term was not used as this was found to cause instability.

Table 4.6: Table showing the PID gains used for the velocity controller.

	$\boldsymbol{k_p}$	$\boldsymbol{k_i}$	$\boldsymbol{k_d}$
X-axis	7	0.1	0
Y-axis	7	0.1	0

4.4.2 Guidance and altitude control

The main objectives of the guidance system in this thesis are to track the helipad and control the drone's altitude to prepare for landing. Two methods are examined for horizontal tracking: (i) target tracking using the pure pursuit (PP) guidance law and (ii) point stabilization using a PID controller. A PID controller is used for altitude control, as it is assumed that the helipad will not move vertically and therefore the PID controller should work well. A limit of $\pm 0.3\,\mathrm{m/s}$ is set for the horizontal velocity references, and $\pm 0.1\,\mathrm{m/s}$ for the vertical velocity reference to avoid aggressive maneuvers.

The errors are defined equally for the horizontal guidance and altitude control. As the drone's perception system estimates the helipad position in the drone body frame, the position errors are defined in this frame as well as follows

$$\begin{aligned} e_x &= x_{h,d}^b - x_h^b \\ e_y &= y_{h,d}^b - y_h^b \\ e_z &= z_{h,d}^b - z_h^b. \end{aligned} \tag{4.24}$$

When tracking the helipad horizontally, the desired positions of the helipad $x_{h,d}^b$ and $y_{h,d}^b$ will always be 0. The desired altitude $z_{h,d}^b$ is typically larger when approaching the helipad and then lower when descending for landing after it is aligned horizontally.

Both horizontal methods are evaluated by how well they manage to stabilize above a stationary helipad in real-world experiments.

Pure pursuit

Using the errors defined in Eq. (4.24), the pure pursuit guidance law computes the desired velocity as

$$\boldsymbol{v}_d^b = \kappa \frac{\boldsymbol{e}^b}{\|\boldsymbol{e}^b\|}, \quad \boldsymbol{e}^b = [e_x \; e_y]^\intercal \tag{4.25}$$

and here the sign of the desired velocity $\boldsymbol{v}_d^b$ is positive due to the error definitions used in Eq. (4.24). This guidance law is tuned using the parameter $\kappa > 0$, where larger values cause the drone to approach the helipad faster and smaller values slower.

Point stabilization using a PID controller

This method involves using a separate PID position controller for setpoint regulation in each axis. Given that the motion of the helipad is sufficiently slow compared to the motion achievable by this controller, the drone will be able to track the helipad. Given the structure of the overall control system seen in Fig. 4.7, the system will be a successive loop closure with a total of three loops for attitude, velocity, and position. Due to the need for the inner loops to be sufficiently fast compared to the outer loop, this also forces this outermost PID controller to be slow, placing another constraint on the speed of the helipad.

The equations for computing the desired velocities in each axis are similar to Eq. (4.21) for the velocity controller

$$
\begin{aligned}
v_{x,d}^b &= K_{p,x}e_x + K_{i,x}\int_0^t e_x + K_{d,x}\dot{e}_x \\
v_{y,d}^b &= K_{p,y}e_y + K_{i,y}\int_0^t e_y + K_{d,y}\dot{e}_y
\end{aligned}
\tag{4.26}
$$

where the derivative and integral errors are defined as in Eq. (4.22) and Eq. (4.23) respectively.

The gains used by the PID guidance system can be seen in Table 4.7. These were found by trial and error where the goal was a controller that was as responsive as possible with minimal oscillations (the results will be shown in Chapter 5). It was found that a simple P controller was sufficient.

Table 4.7: Table showing the gains used for the horizontal PID guidance system.

	$\boldsymbol{k_p}$	$\boldsymbol{k_i}$	$\boldsymbol{k_d}$
X-axis	0.5	0	0
Y-axis	0.5	0	0

Altitude control

Similarly to the horizontal case, the desired vertical velocity is computed as

$$
v_{z,d}^b = K_{p,z}e_z + K_{i,z}\int_0^t e_z + K_{d,z}\dot{e}_z. \tag{4.27}
$$

The gains used for the altitude controller can be seen in Table 4.8. Only a P-controller is used here as well, as this proved to work well. This low gain was used to ensure a stable and controlled descent when landing.

Table 4.8: Table showing the gains used for the PID altitude controller.

	k_p	k_i	k_d
Z-axis	0.2	0	0

4.5 Mission planning and execution

This thesis uses the Graphplan algorithm to generate a high-level action sequence for the drone given a predefined mission, and a mission executor to execute and monitor each action in this sequence. The Graphplan algorithm has been kindly supplied by Miguel Hinostroza from ITK at NTNU. His implementation features different heuristics for searching the planning graph for a solution, and the method used for this thesis is a breadth-first search since the problems and missions here are relatively simple.

The work regarding mission planning in this thesis will be a simple proof of concept regarding the use of AI planning for autonomous drones and for autonomous drones used in search and rescue (SAR) missions.

First, the domain and domain variables will be defined, followed by three different mission problems for the mission planner to solve. The final part of this section will focus on mission execution, i.e. how the drone follows the action sequence generated by the mission planner.

4.5.1 Domain and problem definitions for the Graphplan algorithm

The Graphplan algorithm requires (i) a domain consisting of domain variables and legal actions and (ii) a problem definition to solve by finding a valid action sequence. Before these are defined, some assumptions are made about the missions in this thesis to make them feasible for a proof of concept.

Mission assumptions

The goal of the missions is to search for victims in predefined locations and drop lifebuoys once the victims are found. First, it is assumed that there are three locations that the drone can be in, where location 1 is the location of the helipad, and locations 2 and 3 are the locations of two areas where the drone will perform a search.

The drone is assumed to be a SAR drone capable of carrying a lifebuoy which can be dropped after searching and finding a person in one of the search locations. For the scenarios here, it is assumed that the search will be successful and that the drone will therefore always drop a lifebuoy after searching. The drone also has a finite battery that is expected to be enough for searching in one location at a time and which must be replaced before searching the other location.

It is also assumed that the drone must be tracking the helipad prior to landing, which means the drone must align itself with the helipad before landing when arriving at location 1.

Domain variables

The drone's domain/state variables, also known as propositions, can be derived based on the mission assumptions stated above. The domain variables can be seen in Table 4.9. The state of the drone is defined by the states describing the drone's location, whether it has a battery and a lifebuoy, and if it is tracking the helipad or has landed. The state of the mission is defined by the states indicating whether an area has been searched and if a lifebuoy has been dropped in that area.

Table 4.9: Domain variables for a proposed autonomous drone operating in a SAR mission.

Type	Variable
Drone location	drone_1, drone_2, drone_3
Has battery	batt, no_batt
Carrying lifebuoy	buoy, no_buoy
Search status location i	searched{i}, not_searched{i}
Dropped buoy status location i	dropped{i}, not_dropped{i}
Tracking status	tracking, not_tracking
Landed status	landed, not_landed

Actions

The possible actions the drone can perform are listed in Table 4.10. The drone can move between locations, search a location, drop a lifebuoy in a location, resupply battery and lifebuoy at the helipad, track the helipad to prepare for landing, takeoff, and land. The preconditions listed are the domain variables that must be set for the action to take place. The effects are what variables are added or deleted after the action is finished. As the drone is assumed to always drop a lifebuoy after searching, and the battery only lasts for one search at a time, one of the effects of dropping the lifebuoy is that the battery must be replaced.

Missions

A total of three missions of varying complexity are used to test the Graphplan algorithm. The missions are defined based on an initial state of the domain variables and a goal which is the desired state of the domain variables after the mission is finished. The three missions defined for this thesis can be seen in Table 4.11.

The first mission is a simple mission where the goal is to take off and land on the helipad. In the second mission, the goal is to search area 2 and land back at the helipad, while in mission three, the goal is to drop a lifebuoy at areas 2 and 3 before landing back at the helipad. The first two missions are tested in real-life experiments, while the last mission is used as proof of concept that the Graphplan algorithm can find a valid action sequence for a more complicated mission.

Table 4.10: Possible actions the drone can perform. "∼" means to delete a domain variable.

Action	Symbol	Preconditions	Effects
Move drone	Move{ij}	drone_i	drone_j
Search location	Search{i}	drone_i, batt, buoy, not_searched{i}	searched{i}, ∼not_searched{i}
Drop lifebuoy	Drop{i}	drone_i, batt, buoy, searched{i}	no_batt, no_buoy, dropped{i}, ∼batt, ∼buoy, ∼not_dropped{i}
Resupply battery and lifebuoy	Resupply	drone_1, no_batt, no_buoy	batt, buoy, ∼not_batt, ∼not_buoy
Track helipad	Track	drone_1, not_tracking, not_landed	tracking, ∼not_tracking
Takeoff	Takeoff	drone_1, batt, buoy	not_landed, not_tracking, ∼landed
Land	Land	drone_1, not_landed, tracking	landed, ∼tracking, ∼not_landed

Table 4.11: Three missions of increasing complexity used to generate action sequences using Graphplan.

Mission 1	
Initial state	drone_1, batt, buoy
Goal	drone_1, landed
Mission 2	
Initial state	drone_1, batt, buoy, not_searched2
Goal	drone_1, landed, searched2
Mission 3	
Initial state	drone_1, batt, buoy, not_searched2, not_searched3
Goal	drone_1, landed, dropped2, dropped3

Action sequences

When supplied with the missions in Table 4.11, the Graphplan algorithm generated the action sequences seen in Table 4.12, and these can be verified to achieve the given goals. The computation time dramatically increased from the least to the most complicated mission, with mission 3 using 280.85 ms to compute. This is not a problem in this thesis because the mission planner has no real-time requirement since the action sequence will only be generated once prior to takeoff.

4.5.2 Mission execution

A mission executor has been implemented to execute and monitor the actions in the action sequence generated by the Graphplan algorithm. A flowchart describing the behavior of Graphplan and the mission executor can be seen in Fig. 4.10. The executor accepts an

Table 4.12: Table showing the generated action plans for each mission along with the computation time the Graphplan algorithm used to generate them.

Mission	Action sequence	Time
1	Takeoff - Track - Land	0.39 ms
2	Takeoff - Move12 - Search2 - Move21 - Track - Land	2.12 ms
3	Takeoff - Move12 - Search2 - Drop2 - Move21 - Track - Land - Resupply - Takeoff - Move13 - Search3 - Drop3 - Move31 - Track - Land	280.85 ms

action sequence from Graphplan and then executes each of these actions in order. Each action is implemented as a separate method, where the method itself notifies the mission executor of completion by returning.

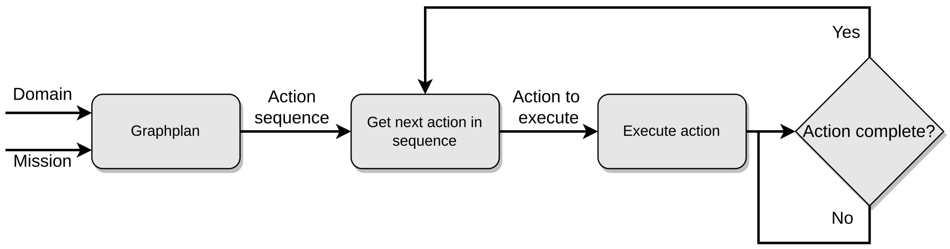


Figure 4.10: Flowchart describing behavior of the mission executor.

The ”Move drone” action uses the internal relative position controller on the Anafi. The locations to move to can be predefined in a config file that supports using both relative position coordinates and GPS coordinates. However, since GPS guidance has not been implemented in the drone interface, only relative coordinates can be used.

The completion criteria are specific to each action. For the ”Takeoff” and ”Move drone” actions, the completion criteria are that the drone is in the ”hovering” flying state, as it will be in flying state ”takingoff” and ”flying” while not done with these commands. For the ”Track helipad” action, the completion criterium is defined as being close enough to the helipad center horizontally and vertically to use the onboard land command of the Anafi to land safely. The actions ”Search location”, ”Drop lifebuoy”, and ”Resupply battery and lifebuoy” complete immediately as they have not been implemented as part of this thesis. The ”Land” action completes immediately after invoking the onboard land command on the Anafi to avoid problems with the system not exiting if a landing fails in the real-world experiments.

5

Results

This chapter presents results from testing the perception, control, and mission planning and execution systems, including the results from the real-world experiments with the DNV ReVolt. Each result builds upon the previous, as the guidance system uses the perception system, and the mission planning and execution experiments use both the perception and control systems.

5.1 Perception

5.1.1 Corner detection and identification

This section presents the results of the different parts of the traditional computer vision (TCV) system. First, the results of the hyperparameter optimization (HPO) for the helipad segmentation and corner detection are presented, and then corner identification and subsequent pose estimation.

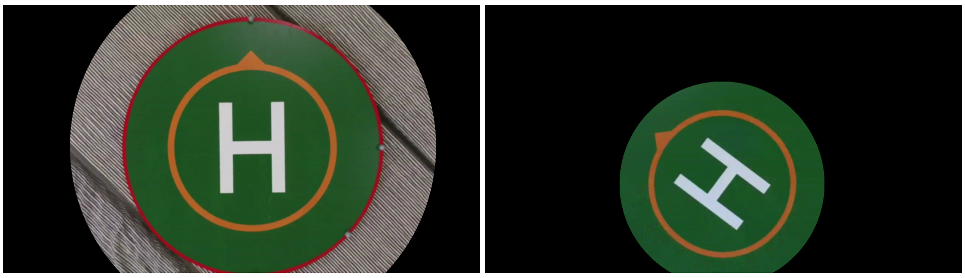
Hyperparameter optimization for segmentation and corner detection

The grid search results and parameters found from the HPO for the Hough circle detector can be seen in Table 5.1. Using a training set of 105 images, the total search duration lasted 19 minutes and 54 seconds. Much of this time was found to be due to inefficient parameter combinations resulting in detection times of more than 1 second per image, while the detection time with the optimal combination was 260 ms per image. The mean prediction errors can be seen to be similar for both the training set and the test set, with a slightly higher error on the test set.

Two example images segmented using the set of parameters found from the grid search can be seen in Fig. 5.1. Overall, 95.4% of the images had correct circle detections across different altitudes and orientations. An example of one of the 7 misdetected images out of the 151 total images can be seen in Fig. 5.1, where the detected circle is too large. All misdetections were of this kind and occurred at different altitudes and orientations. No direct cause of these errors was apparent in the results.

Table 5.1: Table showing the results from the grid search for finding hyperparameters for the Hough circle detector. Results are from flying indoors at different altitudes and orientations but with constant lighting conditions.

Datasets	
Training set size	105 images
Test set size	46 images
Grid search results	
Total combinations tested	64
Total search duration	19 min 54 s
Detection time per image	
Mean	260 ms
Standard deviation	71 ms
Prediction error	
Training set	9.90 pixels
Test set	12.94 pixels
Parameters found	
param1	40
param2	70
Results using parameters found on entire dataset	
Misdetections across both datasets	7/151
Correct detection percentage	95.4%
Mean detection time	179 ms
Median detection time	117 ms
Maximum detection time	2529 ms



(a) Misdetection. **(b)** Correct detection.

Figure 5.1: Figure showing helipad segmentation performed a few images from the test set. Note that the detected circle radius has been increased by 40% to include the helipad arrow in the segmentation.

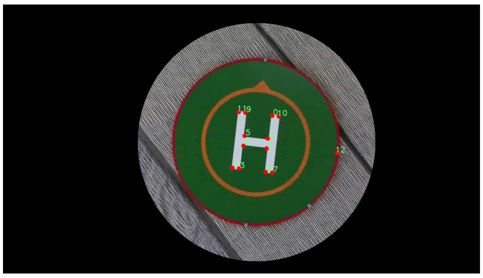
When running the detector on the whole dataset, the mean detection time was 179 ms, but the median detection time was 117 ms due to some images taking much longer to process. The maximum detection time for an image was more than two seconds at 2529 ms. As this was investigated more, it was found that out of the total 150 images, 11 of the images used more than 300 ms to detect the circle, and 3 images used more than 1.5 s. The majority of the images that took the longest were taken at low altitude, with the circle either close to or partially outside the image edge.

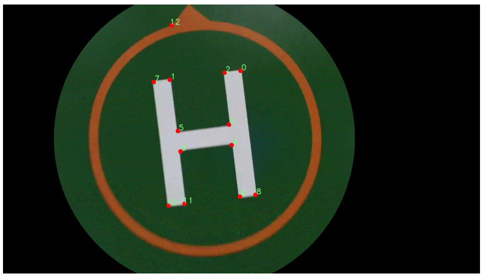
Similarly, the results of the HPO for the corner detection can be seen in Table 5.2. Here the training set is significantly smaller than the test set due to memory limitations, as discussed in Section 4.2.1. The detection time per image is significantly shorter than for the circle detector at 16 ms. The detection time was also found to be more constant across the different parameter combinations than for the circle detector. The Harris detector was found to give better results than the Shi-Tomasi.

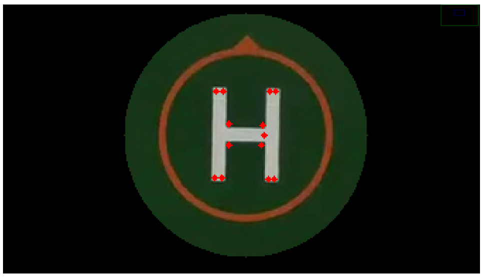
Table 5.2: Table showing the results from the grid search for finding hyperparameters for the OpenCV `goodFeaturesToTrack` corner detector. Results are from flying indoors at different altitudes and orientations but with constant lighting conditions.

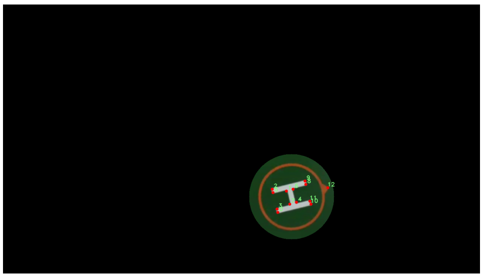
Datasets	
Training set size	37 images
Test set size	114 images
Grid search results	
Total combinations tested	4620
Total search duration	25 min 16 s
Detection time per image	
Mean	16 ms
Standard deviation	2 ms
Mean prediction error	
Training set	2.94 pixels
Test set	8.71 pixels
Parameters found	
Block size	7
Gradient size	17
Min distance	1
Quality level	0.0001
Use Harris detector	Yes
Results using parameters found	
Misdetections across both datasets	9/151
Correct detection percentage	94.0%
Mean detection time	18 ms
Median detection time	17 ms
Maximum detection time	30 ms

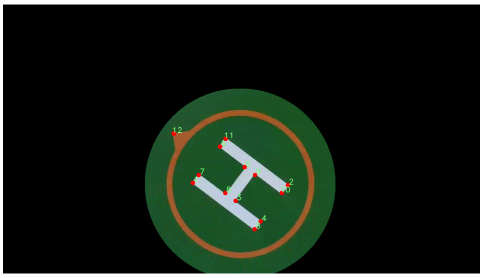
The mean prediction error can be seen to be about three times larger for the test set compared to the training set. Fig. 5.2 shows corner detection performed on six different images in the test set, three of which include misdetections of the corners and three are correct detections. Across the whole dataset, 90.1% of the images had the correct corners identified. 15 of the 151 total images had misdetected corners, and all the misdetections were identified to fall into one of the following categories: (i) incorrect segmentation of the helipad (thus an error from the circle detector propagating), (ii) corners being too close to the edge, or (iii) the helipad being too far away. Four of the errors were due to incorrect helipad segmentation, two due to the corners being too close to the image edge, and the remaining three were due to the helipad flying at too high an altitude. It was also found that high altitudes were a bigger problem than incorrect image segmentation in the case where both were present.


(a) Misdetection due to bad segmentation.


(b) Misdetection due to corner too close to image edge.


(c) Misdetection due to helipad being too far away. Image has been scaled and cropped, and numbers removed to highlight result.


(d) Correct detection far away from helipad.


(e) Correct detection medium distance to helipad.

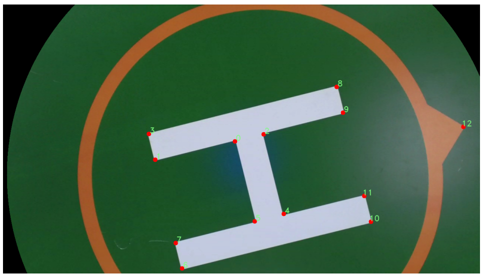

(f) Correct detection very close to helipad.

Figure 5.2: Figure showing corner detection on a few sample images from the dataset. Most errors were due to bad segmentation, but the detector also suffered at high altitudes, and if a corner was close to the edge.

The overall success rate was 94%, and the detector was able to find the corners in all different altitudes and orientation combinations apart from the three cases mentioned above. 7 of the total 9 misdetections were due to errors propagating from incorrect helipad segmentation, meaning most corners were correctly detected once the helipad was correctly segmented.

When run on all the images in the dataset, the mean and median detection times were very close at $18\,\mathrm{ms}$ and $17\,\mathrm{ms}$, respectively, while the maximum detection time was $30\,\mathrm{ms}$. Due to the small variation, no types of images were found to perform notably worse than the rest.

Corner identification

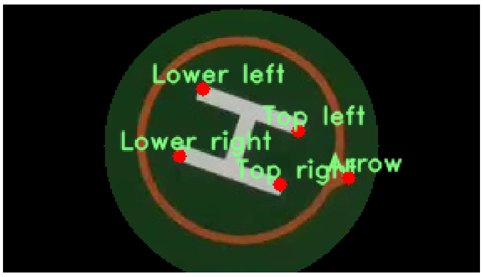
The results from identifying the arrow and the 4 outer corners of the ”H” using Algorithm 1 are presented here. As no ground truth labeling system has been created for the corner identification, the results are based on a visual inspection of the identified corners (arrow, lower left, lower right, upper left, upper right) to see if these coincide with the correct corners.

The results from running the corner identification algorithm across the entire dataset can be seen in Table 5.3. The runtime of the algorithm was consistent across all identifications at $22\,\mathrm{ms}$ as this was the mean, median and maximum runtime.

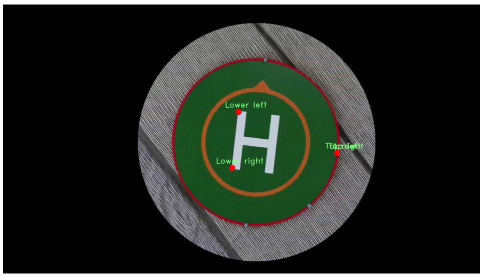
43 images were misidentified across the whole dataset, meaning that the arrow and four outer corners of the ”H” were not identified correctly. As the corner identification relies on the corner detection, the corner detector errors will carry over into these results and account for 11 of the 43 misclassifications. The largest source of errors was misidentifying the upper left and right inner corners of the ”H” as the upper outer corners. This accounted for 74.4% of the total errors, and these errors were frequent when the drone was flying at a high altitude. The last type of error found was when the corner detector found the correct corners, but the corner identification algorithm could not produce a valid result. This happened in 2 of the 43 total misdetections. Examples of the different error types can be seen in Fig. 5.3, along with a few successful corner identifications.

Table 5.3: Table showing the results from identifying the arrow and outer corners of the ”H” based on 13 corner detections from the corner detector.

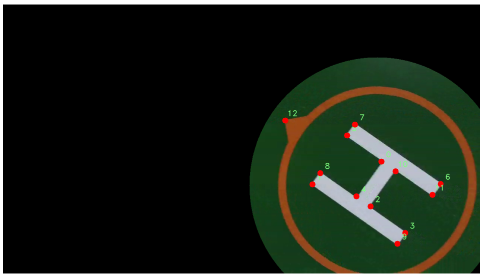
Dataset	
Total number of images	151
Results using parameters found on entire dataset	
Misdetections across both datasets	43/151
Correct detection percentage	71.5%
Mean detection time	22 ms
Median detection time	22 ms
Maximum detection time	22 ms
Distribution of error types	
Top left and/or right corners of ”H” identified on inner corners	32/43 (74.4%)
Incorrect corners detected from corner detector	11/43 (20.9%)
Corners correctly detected, but not able to find valid identification	2/43 (4.7%)



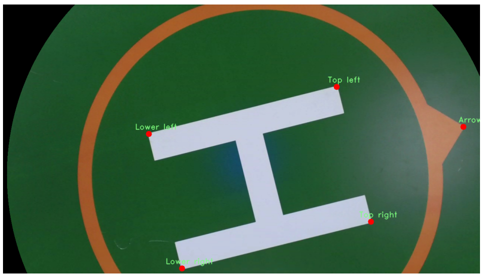
(a) Misidentification of top corners of "H". Image has been cropped to highlight effect.



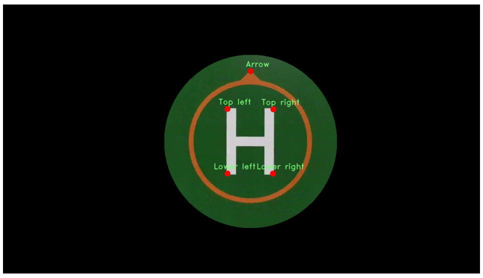
(b) Misidentification of motion capture marker detected as arrow.



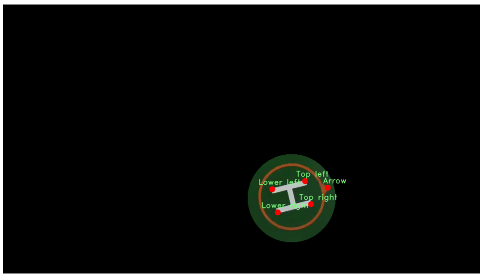
(c) No solution found when all 13 corners are correct.



(d) Correct detection close to helipad.



(e) Correct detection medium distance to helipad.



(f) Correct detection farther away from helipad.

Figure 5.3: Figure showing successful and failed identifications of the features on a few sample images from the dataset. Most misidentifications were due to misidentifying the top corners of the "H", as well as errors coming from corner detection earlier in the pipeline.

5.1.2 Position estimation

This section will present the results from the different position estimation systems developed. First, the position estimates from the TCV system will be presented, then from the deep neural network-based computer vision (DNN-CV) system, and then the Kalman filter (KF) results will be presented with and without the TCV and DNN-CV measurements as corrections. The qualitative behavior of each system will be presented along with a quantitative comparison of all the methods based on the root mean square error (RMSE) of each system with respect to the ground truth.

The drone was flown in a predefined pattern based on the Anafi's relative position controller. The pattern flown was a square, with piece-wise ascent and descent and a rotation at the end. The ground truth position of this pattern, along with labels showing each movement of the drone, is shown in Fig. 5.4.

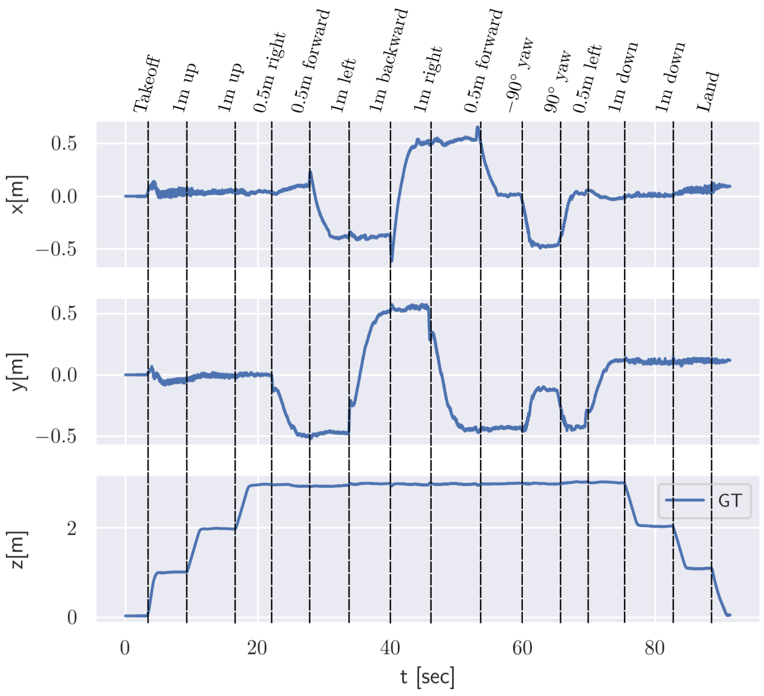


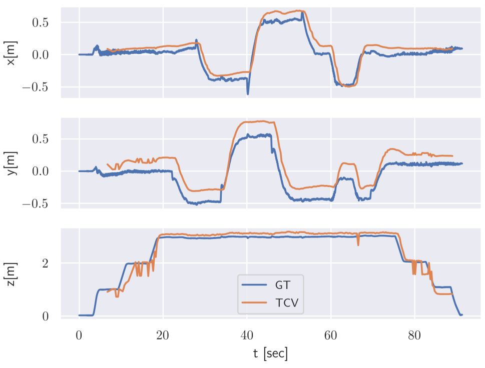
Figure 5.4: Flight pattern used to test perception system. Drone flown inside in the drone lab where ground truth is from the motion capture system.

Traditional computer vision

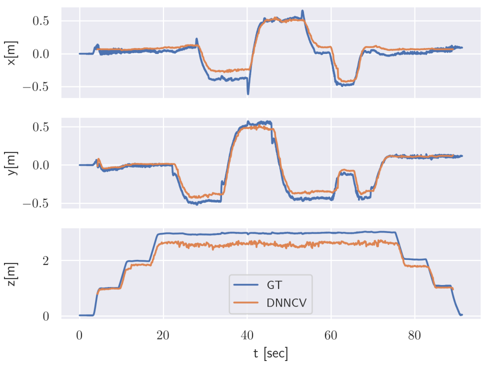
The results from the TCV position estimation system can be seen in Fig. 5.5a. For the x-axis, there seems to be a small vertical offset and some time delay, but overall the estimate follows the ground truth well. The y-axis suffers from a larger offset, as does the z-axis, although harder to see in Fig. 5.5a as the z-axis has a different scale than the two other axes. The cause of these offsets was not found, and it was attempted to correct for them manually. Manual correction fixed the problem in these tests but caused other problems in other experiments, so this solution was discarded.

The z-axis estimate can also be seen to be unstable at the lower altitudes around 20 and 80 seconds, and this is due to the corner misidentifications mentioned above. The estimates can also be seen to only be available from around a $1\,\mathrm{m}$ altitude, as this is the altitude where all the helipad features are visible.

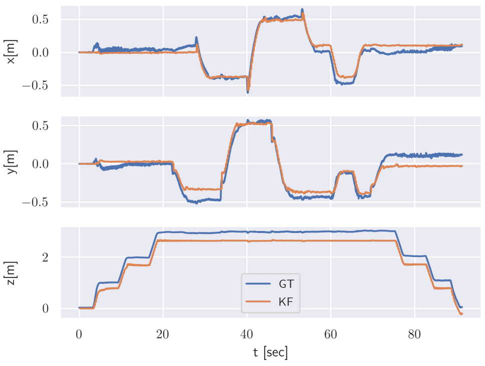
The RMSE for each of the axis as well as the combined RMSE can be seen in Table 5.4. The error was smallest in the x-axis at around $11\,\mathrm{cm}$, while larger in the y- and z-axis, both at around $20\,\mathrm{cm}$. The overall RMSE across all the axes was $17.58\,\mathrm{cm}$.



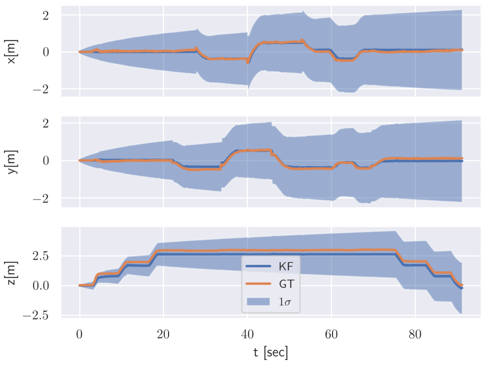
(a) TCV pose estimate vs. ground truth.



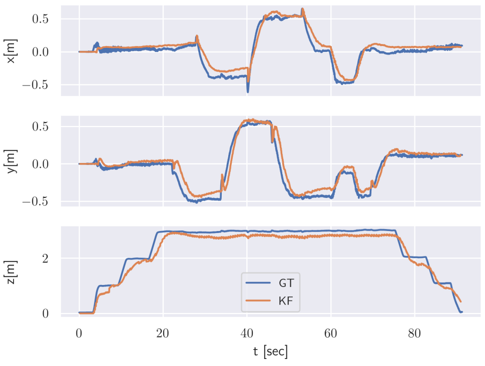
(b) DNN-CV pose estimate vs. ground truth.



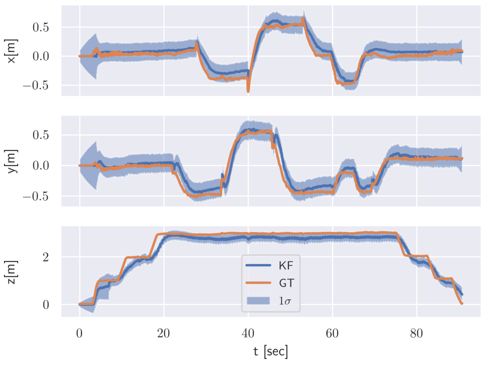
(c) KF with only CV model vs. ground truth.



(d) KF with only CV model vs. ground truth with standard deviation.



(e) Complete KF vs. ground truth.



(f) Complete KF vs. ground truth with standard deviations.

Figure 5.5: Figure showing helipad position relative to the drone body frame $\boldsymbol{p}_h^b$ estimates from the TCV, DNN-CV, and Kalman filter systems. The complete filter performs the best, but suffers from offsets in the TCV and DNN-CV systems.

Table 5.4: Root mean square error for each position estimation system when flying in a predefined square. Input data to each system is recorded from the same flight.

	RMSE for each position estimation system			
	TCV	**DNN-CV**	**KF CV model**	**KF complete**
X	$11.03\,\mathrm{cm}$	$7.95\,\mathrm{cm}$	$6.92\,\mathrm{cm}$	$8.42\,\mathrm{cm}$
Y	$20.30\,\mathrm{cm}$	$6.14\,\mathrm{cm}$	$9.12\,\mathrm{cm}$	$8.84\,\mathrm{cm}$
Z	$19.84\,\mathrm{cm}$	$33.64\,\mathrm{cm}$	$32.16\,\mathrm{cm}$	$21.36\,\mathrm{cm}$
Total	**17.58 cm**	**20.27 cm**	**19.71 cm**	**14.20 cm**

Deep learning based computer vision

The results from the DNN-CV position estimation system can be seen in Fig. 5.5b. Similarly to the TCV system, the estimates are only available from around a $1\,\mathrm{m}$ altitude. Here both the x- and y-axis follow the ground truth well, but the z-axis suffers from a large offset. The offsets in the estimates come from the bounding boxes detected by the YOLO detector not aligning correctly with the helipad edges. This effect is most pronounced in the z-axis, where the detected bounding box is significantly larger than the actual helipad, causing the estimates to assume the drone is closer to the helipad than it is.

This is reflected in the RMSE values seen in Table 5.4, where the x- and y-axis have RMSE values of around $8\,\mathrm{cm}$ and $6\,\mathrm{cm}$ respectively, while the z-axis has a value of $33\,\mathrm{cm}$. This results in an overall RMSE of $20.27\,\mathrm{cm}$, which is slightly higher than for the TCV system. However, most of the the error comes from the large z-offset, meaning the horizontal position estimation is significantly more accurate in the DNN-CV system than the TCV system.

Kalman filter

The results from using the KF with only the constant velocity (CV) model can be seen in Fig. 5.5c. Here, only the velocity measurements from the drone are used to correct the CV model, so this system is, in effect, a pure prediction system. The results can therefore be assumed to have the same general shape as the ground truth but with some offsets. This is indeed what can be observed from Fig. 5.5c. The shape of all the axes follows the ground truth well and even manages to catch the small spikes that happen each time the drone initiates a movement, which was not detected by the TCV or DNN-CV systems. With no corrective measurements, the covariance of the filter is also expected to grow continuously, as is observed in Fig. 5.5d.

Another issue that comes from a lack of corrective measurements is that the drone only estimates the helipad position based on its own velocity. This means that the estimate can be reasonable in a case such as Fig. 5.5c where the helipad is stationary, but if the helipad moves, the filter has no way of knowing, and therefore the estimate will be off. However, due to its accuracy over a short time frame, it can be a viable model for the KF despite its simplicity. The RMSE values for the KF with only the CV model in Table 5.4 can be seen to be solid in both horizontal axes, while larger in the z-axis.

The results from adding both the TCV and DNN-CV measurements as corrections can be seen in Fig. 5.5e. The RMSE values in Table 5.4 show that the complete filter performs the best out of all the methods, with a total RMSE of $14.20\,\mathrm{cm}$. This is better than any of the other systems by themselves, and the main takeaway here is that the horizontal estimate is affected negatively by the large biases from the TCV system, but that the TCV system gives some improvement to the vertical estimate.

Due to the velocity predictions, the filter can be seen to follow the spikes in position. Fig. 5.5f also shows that the covariance is growing initially until there are available measurements from the TCV and DNN-CV systems, but then remains constant due to the periodical arrival of new measurements.

5.2 Guidance and control

This section will present the results from the velocity controller and guidance system. Here, results from both the simulator and real-world experiments are shown, as the simulator was used to verify that the various control methods were correctly implemented and tuned.

5.2.1 Velocity control

For the velocity controller, both the model-based and the PID controller are tested. The same velocity reference trajectory is used to compare both methods, and the same experiments are validated both in simulation and real-world experiments. As this is a velocity controller meant to be used inside a guidance system, the essential part is not following the reference trajectory precisely but showing robust performance.

Model-based controller

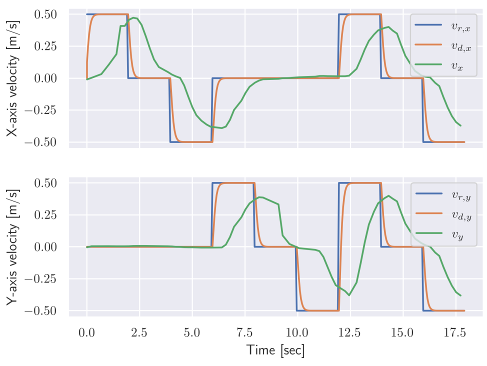
The results from testing the linear drag model model-based controller can be seen in Fig. 5.6. The velocity in both simulation and real-world experiments, seen in Fig. 5.6a and Fig. 5.6c respectively, show a similar response. The response is significantly delayed compared to the reference trajectory, resulting in neither of the responses reaching the complete reference velocity before it changes.

The generated attitudes references can be seen in Fig. 5.6b and Fig. 5.6d. The actual attitude follows the reference well for both simulation and real-world experiments, apart from a time delay also here. This delay comes from the time delay of the velocity measurements from the Anafi. The generated attitude references stay within their bounds of $\pm 5^\circ$ and show a controlled response.

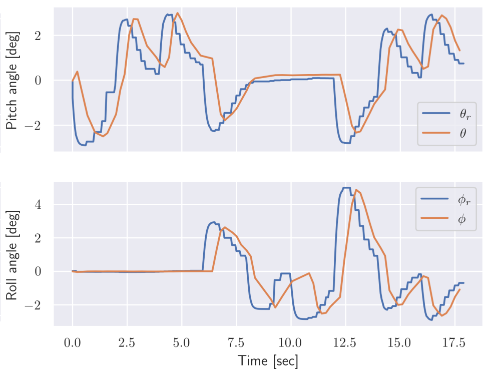
Overall, the controller based on the linear drag model can be seen to work pretty well. The computed attitude references are relatively smooth, and the subsequent output velocity somewhat manages to reach the desired values.

PID controller

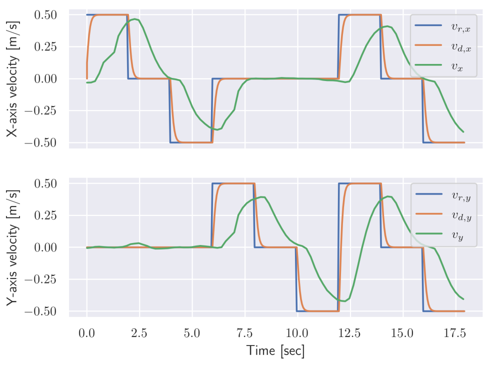
Similarly, the results from using the PID velocity controller can be seen in Fig. 5.7. Compared to the model-based controller, the velocity response is slightly faster, meaning the



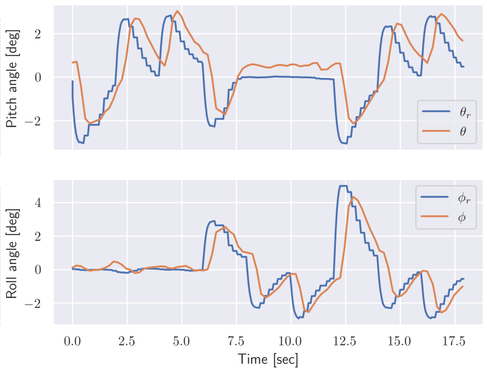
(a) Simulation velocity vs. reference.



(b) Simulation attitude vs. reference.



(c) Real-world velocity vs. reference.



(d) Real-world attitude vs. reference.

Figure 5.6: Figure showing the velocity and attitude response for the model-based velocity controller in simulation and real-world experiments.

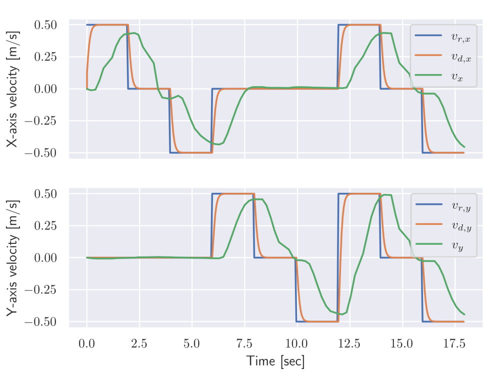
drone is closer to reaching the velocity setpoint. As the drone setup is the same as for the model-based controller, the response also here suffers from time delays as seen in Fig. 5.7a and Fig. 5.7c.

The generated attitudes references seen in Fig. 5.7b and Fig. 5.7d are also here smooth and never saturated, but with the same delays as in the model-based controller.

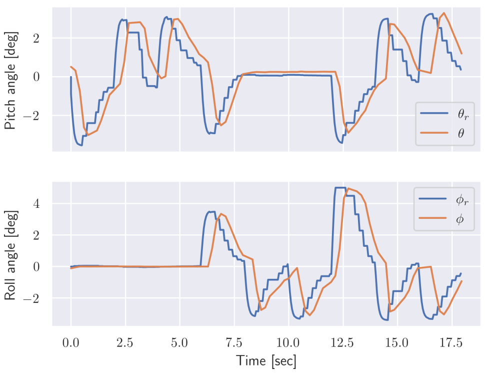
Overall, the PID velocity controller works well, with a relatively fast velocity response. Due to the faster velocity response, this controller was chosen as the controller used in all other experiments.

5.2.2 Guidance and position control

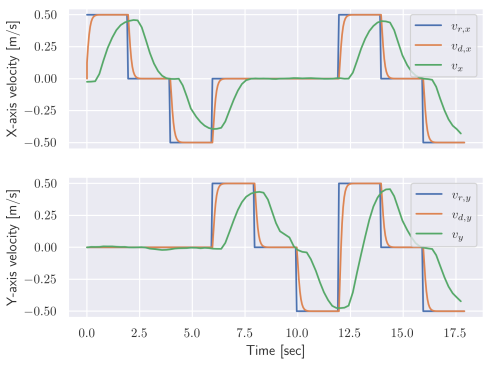
The output of the Kalman filter with all measurements is used as feedback when aligning the drone with the helipad for testing the pure pursuit (PP) target tracking and PID point stabilization methods. The output of the perception system $\boldsymbol{p}_h^b$ can be interpreted as the position error when trying to align horizontally with the helipad.



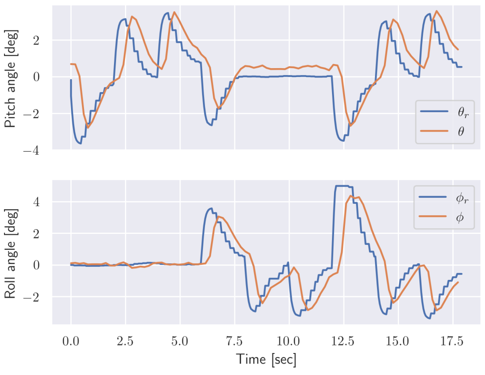
(a) Simulation velocity vs. reference.



(b) Simulation attitude vs. reference.



(c) Real-world velocity vs. reference.



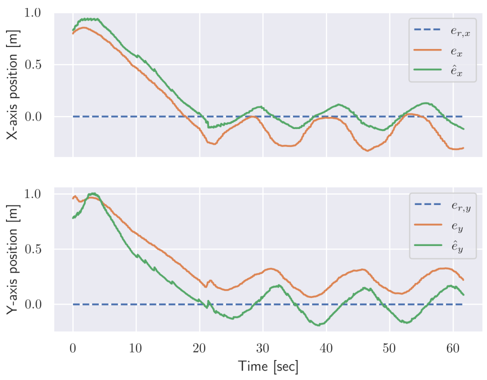
(d) Real-world attitude vs. reference.

Figure 5.7: Figure showing the velocity and attitude response for the PID velocity controller in simulation and real-world experiments.

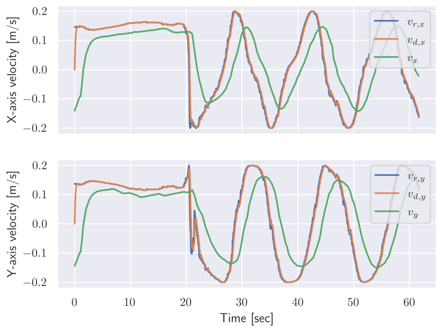
Pure pursuit

The results from using the PP guidance law in the simulator can be seen in Fig. 5.8. In Fig. 5.8a the estimated position error is plotted alongside the ground truth error, and an offset, as discussed in the perception results above, is present. The position error can be seen to decrease rapidly to zero as the drone approaches the helipad but then oscillate around the reference indefinitely. These results are using a value $\kappa = 0.2$ for the tuning parameter in the PP guidance law, and different values were tried. A smaller value resulted in less, but still persistent, oscillations and a slow approach speed to the helipad. Similarly, a larger value resulted in a fast approach and bigger oscillations. Due to these oscillations, the PP guidance law was not tested in real-world experiments.

The poor performance is to be expected as the PP guidance law is based on just arriving at or hitting a target moving at speed. In these tests, the helipad was stationary, and the goal was to remain constantly above it, so the method is not well suited. It might fare better if the helipad was moving, but as this is not possible to test in the simulator and not feasible in real-world experiments indoors due to the risk of crashing, this has not been investigated.



(a) Simulation position error vs. ground truth.



(b) Simulation velocity vs. reference.

Figure 5.8: Figure showing the estimated and ground truth position error and velocity response for the pure pursuit guidance law in simulation. Drone never stabilizes above the helipad.

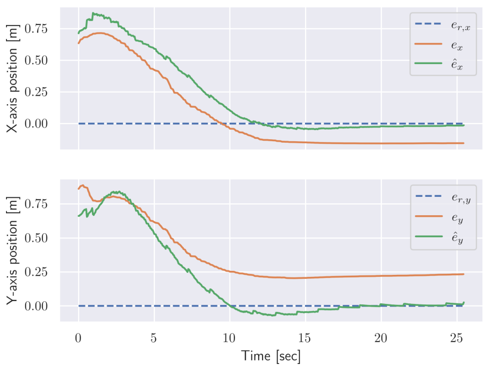
PID controller

The PID point stabilization guidance law fared better, as can be seen by the results in Fig. 5.9. The position error for the simulated results can be seen in Fig. 5.9a, and here the error can be seen to converge smoothly to zero (although not quite to zero due to the offset from the perception system). This controlled response is also reflected in the velocity references, which can be seen in Fig. 5.9b to be saturated at first, but then flatten off smoothly to zero.

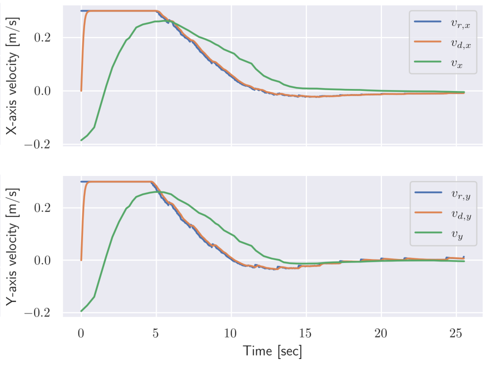
These promising results were carried over into real-world experiments, where the position error can be seen in Fig. 5.9c. Here, the offset between the perception output and the ground truth position error is smaller than in the simulator because the Kalman filter is tuned based on real-world data. The response can be seen to not be as smooth as in the simulator, with some small oscillations around the reference, reflected in the velocity references in Fig. 5.9d as well. The controller gain was attempted lowered to remove the oscillations, but this resulted in a too slow controller that could not follow the helipad once it was moving. Nevertheless, the oscillations are minor, and the overall response is considered adequate for tracking the helipad. This method was therefore chosen to be used in further experiments.

5.3 Mission planning and execution

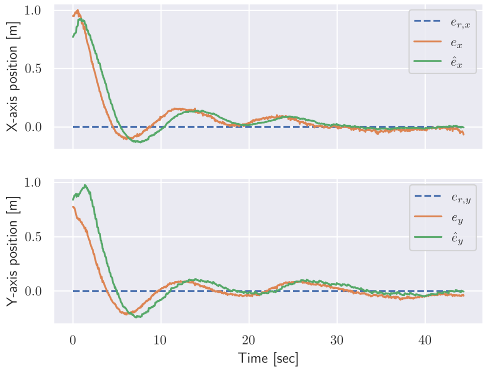
This section will present the results from testing the complete system with perception, guidance and control, and mission planning and execution. First, testing inside where ground truth measurements are available will be performed to see if the overall system manages to execute Mission 1 (Takeoff - Track - Land) while the platform is moving. Then, the overall system will be tested in its intended environment outside where the



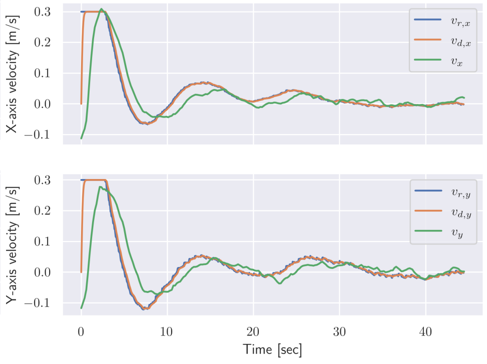
(a) Simulation position error vs. ground truth.



(b) Simulation velocity vs. reference.



(c) Real-world position error vs. ground truth.



(d) Real-world velocity vs. reference.

Figure 5.9: Figure showing the estimated and ground truth position error and velocity response for the PID setpoint controller guidance law in simulation and real-world experiments. Drone stabilizes in both environments, with some more oscillations in the real-world experiments.

helipad is attached to the DNV ReVolt vessel. During these experiments, both Mission 1 and Mission 2 (Takeoff - Move12 - Search2 - Move21 - Track - Land) will be tested, both while the ReVolt is on land and at sea.

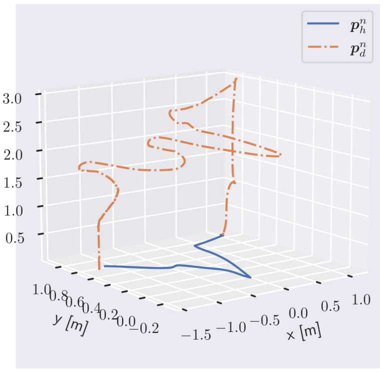
5.3.1 Landing on a moving platform

First, to test the guidance system's ability to land on a moving platform, two tests were designed. In both tests, a rope was attached to the helipad and it was pulled along the floor. While the helipad was moving, the drone was tasked with landing autonomously on it following Mission 1. The first experiment tests the drone's ability to react to fast and short movements of the helipad, while the second tests the drone's ability to land on the helipad when it is moving slowly and constantly.

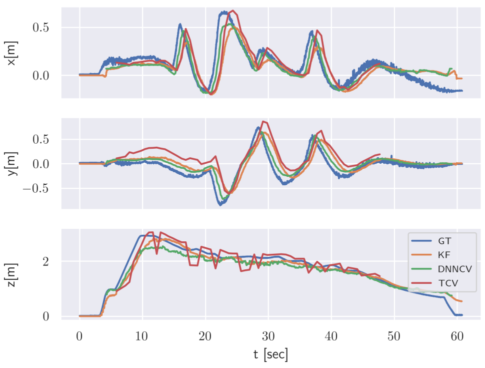
In the first experiment, the helipad is moved fast and discretely in random directions for short periods and then left stationary a bit before moving again. The helipad is moved like this until the drone is close to landing, and then the helipad is left stationary. The results from this experiment can be seen in Fig. 5.10, where the 3D trajectory of the drone and helipad can be seen in Fig. 5.10a. As can be seen, the drone is able to follow the helipad while descending, with some overshoot at certain areas. Nevertheless, the drone manages to land autonomously on the platform close to the center. All the outputs of the perception system during this experiment can be seen in Fig. 5.10b. The overall accuracy is decent, and it can be seen that the system manages to observe the fast movements and correct for them. The z-position can be seen to flatten out at certain points, corresponding to when the helipad moves sharply. The reason for this is that the drone pauses the descent until realigned horizontally. The TCV system is seen to suffer more from misdetections in this case when the helipad and drone are moving, and this is due to more corner misidentifications.

In the second experiment, the helipad was moved along the floor at a slow, constant velocity in only one direction, as seen in Fig. 5.11a. In this experiment, the helipad also moved during the entire mission, even during the drone landing. As seen in Fig. 5.11b, the drone oscillated a bit around the helipad center, but descended at a constant pace and managed to land right in the middle of the helipad as seen in Fig. 5.11a.

Overall, these two experiments demonstrate the guidance system's ability to land the drone on a moving target, given that the movement is either slow or that the drone is given time to catch up.



(a) Helipad and drone trajectory.



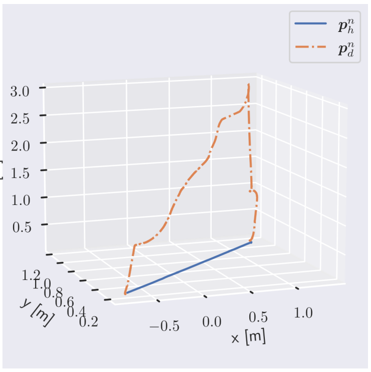
(b) Perception output during landing.

Figure 5.10: Figure showing the 3D trajectory and perception output of the drone while landing on discretely moving helipad (stationary during the final landing). Drone shows some oscillations when realigning with the helipad.

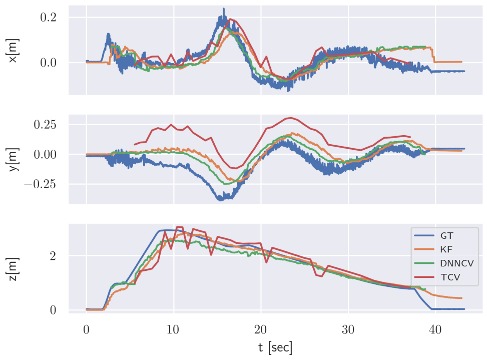
5.3.2 Landing on the DNV ReVolt

The final experiments done in this thesis were executing Mission 1 and 2 with the helipad attached to the DNV ReVolt. This is the most realistic testing scenario concerning the ultimate goal of performing autonomous search and rescue (SAR) missions at sea.

The system was first tested with the ReVolt resting on a boat trailer on land. This allowed the ReVolt to be pulled along to test landing on a moving target similar to the tests



(a) Helipad and drone trajectory.



(b) Perception output during landing.

Figure 5.11: Figure showing the 3D trajectory and perception output of the drone while landing on continuously moving helipad (moving during the final landing). Drone shows minor oscillations, and descends continuously.

presented above. At sea, for practical reasons, the ReVolt was kept in a shallow spot where its motion could be controlled by hand and using ropes. Due to the limited space of this shallow spot, the ReVolt could not be moved horizontally, but the vessel could be made to roll from side to side to better replicate the real-world motion the vessel would experience while at sea.

The land and sea experiments were done on different days, but the weather conditions were similar, with cloudy weather and temperatures around 10 °C. There was occasionally more sun and wind during the land testing, but not enough to significantly affect the results.

A total of 10 tests were carried out for the land tests and a total of 6 at sea. Although not enough to make definitive claims about the system, these tests give an indication of its overall performance. The results from all the tests have been summarized in terms of the number of successful landings in Table 5.5, and the results from each test will be discussed below.

Land experiments

The drone managed to land on the ReVolt in each experiment when the boat trailer was stationary. One example of such a landing can be seen in Fig. 5.12 where the drone took off and landed back on the helipad. The output of the perception system shows that the drone oscillates slightly around the helipad center but manages to find it and lands close to the center.

Mission 2, where the drone flew away a small distance and then back, also worked well when the ReVolt was stationary. The drone was flown 2 m away from and back to the ReVolt, and the relative position control on the Anafi proved reliable enough that the drone could see the helipad once back, thereby enabling the tracking and landing to work.

However, when the ReVolt was moving outside in these experiments, the promising indoor results from the drone lab could not be replicated. As seen in Table 5.5, only 1 of the 5 total experiments where the ReVolt was moving resulted in a successful landing.

Table 5.5: Rate of successful landings on the ReVolt for the land and sea experiments.

Land experiments	
Mission type	**Successful landings**
Mission 1, ReVolt stationary	4/4
Mission 2, ReVolt stationary	1/1
Total - ReVolt stationary	**5/5**
Mission 1, ReVolt moving continuously	0/4
Mission 1, ReVolt moving discretely	1/1
Total - ReVolt moving	**1/5**

Sea experiments	
Mission type	**Successful landings**
Mission 1, ReVolt stationary	1/1
Mission 2, ReVolt stationary	2/2
Total - ReVolt stationary	**3/3**
Mission 1, ReVolt rolling	0/2
Mission 2, ReVolt rolling	1/1
Total - ReVolt rolling	**1/3**

The successful landing was also after the helipad stopped moving for the final part of the descent, meaning the drone was never able to follow the continuously moving ReVolt well enough to manage to land.

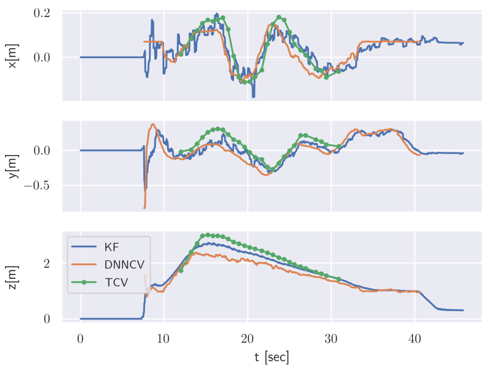
It was found that the drone always lagged behind the moving ReVolt by about $1\,\mathrm{m}$ and was never able to stay horizontally aligned until the ReVolt came to a stop. Misdetections in the DNN-CV perception system were also present, making the drone unable to realign itself once the helipad was not in the image as it detected other parts of the ReVolt as the helipad.

It was attempted to change the landing system to only descend once horizontally aligned, and pause the descent until realigned if the horizontal error grew above a certain threshold. Using this resulted in the drone never descending due to it always lagging a distance behind the moving helipad.

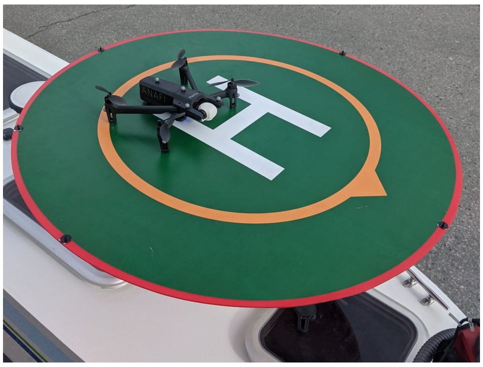
Another discovery was that the drone was sometimes unable to initiate a landing due to incorrect altitude measurements. It was found that the vertical DNN-CV measurements flattened out around $0.7\,\mathrm{m}$, meaning that the overall vertical estimate remained constant even while the drone was descending past this point. This caused the drone to descend close to the helipad but not initiate its internal landing command as it thought it was higher up. To avoid this, the Kalman filter was changed to only use the horizontal DNN-CV measurements for correction when the drone was below $0.7\,\mathrm{m}$, and predict the final part of the descent using the velocity measurements.

Sea experiments

Similar results were found when testing the system at sea. The drone was able to land on the ReVolt during all the experiments where the vessel was kept stationary. As in the land experiments, it was found that both Mission 1 and 2 gave equally good landings when the



(a) Perception output.



(b) Helipad landing result.

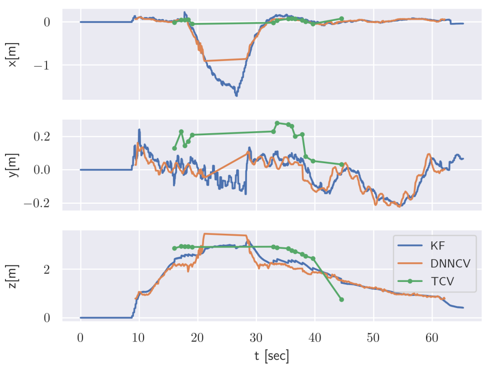
Figure 5.12: Figure showing the perception output and resulting landing on the helipad from executing Mission 1 while the ReVolt was stationary on land. All perception systems produce similar estimates and the drone lands continuously.

drone was stationary, as the tracking system was able to rediscover and track the helipad in Mission 2.

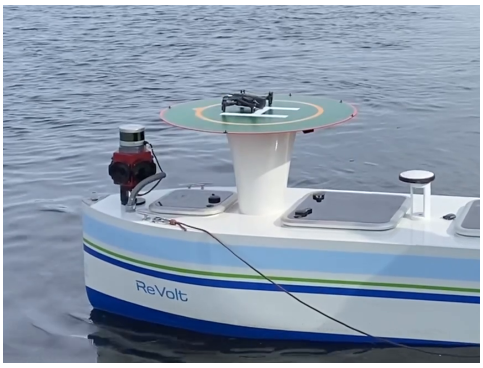
However, the introduction of rolling motion into the ReVolt caused problems. In one of the experiments, the drone could never start descending because of a large offset in the perception data. In the other test of Mission 1, the drone started descending until close to the helipad but lost track right before landing and could not regain it due to misdetections from the perception system.

In the very last experiment, Mission 2 was tested with rolling motion, and this time it worked, as can be seen by the perception output and resulting landing seen in Fig. 5.13. The perception system can be seen to estimate the position well when the helipad is not in view, as the Kalman filter estimate and DNN-CV estimate align well when the helipad reappears in view. The TCV system can be seen to suffer a lot in this scenario, with only very infrequent measurements available and a significant bias in the y-axis compared to the DNN-CV measurements.

Overall, the landing can be seen to be successful as the drone lands very close to the helipad center. However, as this still only accounts for 1/3 of the experiments where the ReVolt was rolling, the system does not fare well when the helipad experiences significant motion.



(a) Perception output.



(b) Helipad landing result.

Figure 5.13: Figure showing the perception output and resulting landing on the helipad from executing Mission 2 with the ReVolt at sea an rolling. The TCV system fails to produce reliable estimates, but the perception system is saved by accurate measurements from the DNN-CV system and predictions from the velocity measurements.

6 Discussion

This chapter discusses the results from the perception, control, and mission planning and execution systems. The results and observations from the real-world experiments with the DNV ReVolt and in the drone lab are also discussed, with some closing remarks connecting the results to the underlying goal of autonomous search and rescue missions.

6.1 Perception

6.1.1 Traditional computer vision

The traditional computer vision (TCV) system was demonstrated to work to a limited degree as it was able to produce pose estimates in real-world experiments. However, the main problems with the system were that

- overall runtime was slow and inconsistent
- correct corner identification was sparse when testing outside
- position estimates had offsets in all axes

The circle detection was the main factor affecting the runtime of the TCV system, as this had a median run time $117\,\mathrm{ms}$ in addition to using multiple seconds on certain images. The reason for this slow runtime could be the parameters used in the Hough circle detector, where the range of pixels to detect circles in was set to $[50, 500]$. Although this allows for segmentation at different altitudes, it also drastically increases the search space for the detector. To narrow down this search space, an expected range for the circle radius could be determined from the current altitude estimate of the drone. This would, however, not make the system completely independent from the state estimation of the drone, which could be a problem. To still have an independent system, a different circle detector could be used, such as a YOLO detector for seeing the helipad, similar to what is already used in the DNN-CV system. This detector would be faster, and the detection time would be independent of the size of the helipad in the image without requiring any prior knowledge.

The 5 known corners that were the final result of the corner detection pipeline were only rarely found when flying outside in real-life situations. The pipeline suffers from errors propagating from one step to the next, as corner detection fails when helipad segmentation fails, causing the corner identification to fail as well. The corner identification also introduced its own errors by frequently misidentifying the top corners of the ”H” when the drone was far away and sometimes producing a valid result even when the corners found from the corner detector were not the correct ones. The algorithm could be made more robust by introducing more consistency checks between the distances of the points and using different metrics to determine the top two corners of the ”H”. One way to completely bypass this pipeline is by using fiducial markers such as AprilTags [44] that are easy to detect and identify. These could be placed inside the ”H” on the helipad and thereby retain the classic helipad design when viewed from far away. Another benefit of this is that the pose estimation would be available at lower altitudes, as it is now only available above a certain altitude due to the large size of the helipad features.

The cause of the offsets in the position estimates was not found. It was attempted to compensate for the offsets manually, but they were found to be varying when performing different types of tests with different maneuvers, so this had no effect. The cause of these offsets will have to be investigated more.

6.1.2 Deep neural network-based computer vision

The DNN-CV position estimation worked better than the TCV estimation system, due to faster and more frequent detections from the YOLO detector. The main problems with this system were

- offsets due to incorrect bounding box sizes
- false detections of the helipad
- incorrect altitude estimation at low altitudes

The offsets in the estimates were most pronounced in the z-axis. They were due to the YOLO detector estimating too large bounding boxes for the helipad, thereby making the helipad appear closer than it is. These errors could be rooted in the labeling process of the images used to train the detector. Therefore, the detector could be retrained by using new training data that can be labeled automatically to ensure the labels are accurate.

Another advantage of retraining the detector is reducing the number of false detections. During the real-world experiments, it was found that the detector detected parts of the ReVolt as the ”H” and helipad perimeter. This caused the drone to be unable to regain tracking once the helipad was out of view as the DNN-CV system claimed the helipad was still below the drone. The detector is trained on images from the AR.Drone 2.0, which has severely lower image quality the Anafi, meaning worse detection results are to be expected when used on the Anafi.

The detector was also not able to estimate the drone’s altitude once it was below around $0.7\,\mathrm{m}$. This was due to the YOLO detector only being able to create bounding boxes the size of the image, meaning if the helipad went outside the edge of the image, the detector would report a bounding box corresponding to the case when the helipad was inside the image view. This means the altitude estimates remained constant as the drone descended past this threshold, leading to these estimates having to be removed as corrections in the Kalman filter. A solution to this would be using the DNN-CV system only for approximate position estimation when the drone is sufficiently far away and then turning off the estimates in favor of more suitable methods once the drone is close to the helipad.

6.1.3 Kalman filter

The Kalman filter (KF) was able to fuse the TCV, DNN-CV and drone velocity measurements to produce a combined estimate with a reasonable estimated uncertainty. The predictions using the constant velocity (CV) model and velocity measurements from the drone were accurate over short periods, while the position measurements from the TCV and DNN-CV systems were able to correct the predictions. There were, however, some factors limiting the performance of the filter, the main ones being

- inaccurate position measurements from TCV and DNN-CV
- time delayed velocity measurements from the Anafi
- only local navigation reliant on having the helipad in view

The underlying reason for inaccuracies in the KF was the poor position measurements from the TCV and DNN-CV systems. The large biases in these measurements propagated into the KF as well, and as the filter has no estimation of these biases, the final estimates were inaccurate. These biases could be estimated if the filter had additional accurate position measurements, but it is believed that a better solution is to remove the errors in the measurements themselves.

Time-delayed velocity measurements from the Anafi also proved troublesome for the filter, as the velocity reported by the Anafi could be as much as 1 second delayed compared to the ground truth. This meant that during a change in velocity, the estimates with a shorter delay from the DNN-CV system would be counteracted by the CV model as no change in velocity had been recorded yet. This issue was most significant when tracking the helipad for landing, as this involved many small changes in the velocity. After completing the experiments, it has been noted that the Anafi sends some telemetry data, including velocity measurements, along with the images streamed from the drone. This data then arrives at the same frequency as the images at close to 30 Hz, and based on a quick inspection seems to have less of a delay than the measurements arriving at 5 Hz. Updating the drone interface to use these measurements instead would improve the prediction accuracy due to an increased amount of measurements and improve the time delay consistency of the measurements in the filter, as all measurements are now based on data from the same image.

The final limiting factor of the filter is that the navigation is only local and dependent on having the helipad in view to get corrections. The disadvantage of this is that the drone cannot fly far away from the helipad without losing it due to the predictions from the CV model not being accurate over extended periods of time. Solutions to this would be introducing additional states in the filter for navigation in, e.g. the NED frame, using GPS measurements as corrections. Another issue is that any movement of the helipad while outside of the drone's camera view is not detected and would lead to the drone losing track once it returns to where it thinks the helipad is. Here the filter could be additionally expanded by tracking the NED position of the helipad as well and incorporating some mechanism of correcting these estimates from external sources.

6.2 Guidance and control

6.2.1 Velocity control

Velocity control on the Anafi worked well in simulation and real-world experiments using both the model-based open-loop method and the PID method, where the PID method was ultimately chosen as it provided a slightly faster and smoother response. However, the main limitations to the velocity controller on the Anafi were

- transmission delays
- delayed velocity measurements feedback

Transmission delays mean the drone's response is delayed compared to the reference. This makes aggressive trajectory tracking with temporal constraints infeasible for the Anafi if the trajectory changes too quickly. For trajectory tracking, slow and controlled maneuvers should instead be used. Another option is to use path following where no time constraints are placed on the drone to reach the given positions.

The large delays in the velocity measurements used as feedback for the PID controller are also undesired as they could cause instability in the control system. The chance of instability increases the more aggressive maneuvers are attempted, meaning that the drone should be limited to slow maneuvers to mitigate this problem. In any case, the large measurement delays are still undesired and should be reduced by updating the drone interface as described above.

6.2.2 Guidance

Using the pure pursuit (PP) target tracking guidance law proved infeasible for stabilizing above the helipad, so the point stabilization PID guidance law was used instead as it showed promising results. This system of successive loop closure with the velocity controller stabilized the drone around the helipad center, although with some oscillations when testing in real-world experiments. The main problems with the guidance system were

- oscillations when stabilizing
- constantly trailing behind when helipad moving

Horizontal tracking of the helipad worked well when it was stationary or when the drone was given time to catch up with it. However, significant oscillations were observed when first approaching the helipad, an effect that was more profound when the helipad was moving and stopping. A contributing factor to this could be the time delay of the position estimate from the Kalman filter used to calculate the next reference. Reducing this would likely increase the performance of the system. Reducing the gains on the controller was tried, but this resulted in a response that was too slow. The integral and derivative parts of the controller were not used due to the good response the system showed in the experiments when using just a P-controller. Investigating the use of the integral and derivative terms could help improve the performance in real-world experiments with wind and helipad movement.

The guidance system was able to land the drone on a moving platform inside in the drone lab, but not outside on the ReVolt where the drone trailed too far behind the helipad and never caught up with it. The results indicate that this caused by the guidance system, as the perception system is constantly estimating an offset that the guidance system is never able to close. This can be due to the P-controller used, and adding integral effect might improve the tracking. Another option is using guidance laws based on following moving objects that may be better suited. The PP guidance law might have been able to catch up with the helipad, but the oscillations would still be present when trying to land, so this would still be infeasible. Another option could be to investigate the line-of-sight (LOS) guidance law, where the drone attempts to align itself with the helipad. Using a PID controller for altitude control would still be feasible given that the helipad only moves horizontally.

6.3 Mission planning and execution

6.3.1 Graphplan

The Graphplan algorithm was able to generate action sequences for all three missions designed in this thesis. The algorithm quickly generated the plans and was well suited to generate a flight plan prior to takeoff.

Although the third mission was a bit more complicated and required some resupplying, all three missions were, in fairness, simple missions where using the Graphplan algorithm is strictly speaking unnecessary. However, the results show that it is possible to define a domain and create missions for an autonomous UAV, where the Graphplan algorithm would be able to generate a plan regardless of the mission complexity. The algorithm could, e.g. be particularly useful in the case where there are multiple UAVs involved in a search and rescue (SAR) mission, similar to its demonstrated use in coordinating two warehouse robots picking up cargo in [59].

The algorithm could then find a mission plan that coordinates the actions of different UAVs such that they collectively solve a complex mission. An example of this could be that when first arriving at a scene, multiple UAVs could be coordinated to search a large area to increase the situational awareness of the responders. Then once more information about the situation is known, the goals could be updated accordingly, and the UAVs could be coordinated to best serve the needs of this specific mission.

6.3.2 Mission execution

The mission executor managed to execute the action sequences from the Graphplan algorithm well. The system of defining standardized actions for the drone to perform proved to work well and makes it possible for the drone to handle complex missions with the same simple mission executor.

However, the one action that saw problems was when descending to land on the helipad. A necessary improvement made was to have the drone stop descending once the horizontal error was above a certain threshold, where descending did not resume until realigned. This increased the chance of the system regaining alignment with the helipad, but an even more robust system would still be desirable. E.g. a system that could evaluate the quality of the perception system output would be beneficiary, as the drone could ascend and restart the landing if the perception output was unreliable. This would help avoid missing the helipad due to the false position estimates discussed above that originate from false detections of the helipad, as happened in the experiments where the drone trailed the helipad and descended past it during the final part of the landing.

6.4 Real-world experiments

Despite the issues with the different subsystems mentioned above, the overall system was able to successfully complete

- autonomous landings on the ReVolt while stationary at sea and land
- autonomous landing indoors on a moving platform

Achieving the overall thesis goal of landing the Anafi on the ReVolt proved that the system was robust enough to work in the intended environment. However, the system did have issues landing on the ReVolt while it was moving on land or rolling in the water. While moving on land, the issue lies with the guidance system not being able to follow the ReVolt and always lagging behind. The system did manage to land on the platform while it was moving when tested indoors, but this could be due to fewer disturbances and the helipad moving at a slower speed than the ReVolt.

Based on this knowledge, some conclusions can be drawn concerning the objective of investigating what type of control system is suited for landing the Anafi reliably on the helipad. The experiments indicate that using a PID controller for the underlying velocity control of the system is a viable option and something that would likely perform even better with less measurement delay on the velocity measurements used for feedback. The experiments also show that using a PID controller for position control is a viable option for landing on the platform when it is fully or nearly stationary, even when tested outside and at sea with environmental disturbances. In the case of a non-stationary helipad, however, a different guidance law capable of following the helipad without trailing it is required for the drone to land.

Conclusions can also be made about the objective of investigating what type of perception system is suitable for estimating the Anafi's position relative to the helipad. The complete perception system was successful in the sense that it produced position estimates for the drone frequently and accurately enough to allow it to land on the helipad. The position estimates from the TCV and DNN-CV systems provided the Kalman filter with position corrections, while the constant velocity model with velocity measurements was able to accurately predict the motion of the Anafi during short periods when the helipad was not in view. This indicates that using computer vision techniques in a Kalman filter is a viable method of estimating the position of a UAV relative to a helipad of known appearance. The Kalman filter design implemented is, however, accurate only when the helipad is parallel to the ground. This is reflected in the experiments at sea, where the ReVolt was rolling from side to side, causing the filter to estimate that the horizontal error was too large to start descending in two of the three experiments. This suggests that if the UAV is landing on a helipad with more complicated dynamics than just translational motion, the helipad dynamics should be included in the filter to give accurate position estimates of the UAV relative to the helipad.

The investigation into whether AI planning could be used for mission planning of autonomous UAVs in SAR missions has been done on a more conceptual level than the other two objectives. The Graphplan algorithm was demonstrated to be able to compute action plans for the different mission goals defined in this thesis, and the two first missions were tested successfully in real-world experiments on the ReVolt. However, as stated above, the benefit of using such an algorithm for mission planning might be more significant for a more complicated system with more advanced missions or multiple UAVs. Based on the results from using Graphplan in this thesis, it can be concluded that AI planning can be used for mission planning of a single UAV. However, further research into the topic is necessary as the low mission and domain complexity in this thesis prevents truly evaluating the potential of such methods for this application.

6.5 Towards use in search and rescue

The results and contributions of this thesis will now be related to the underlying theme of marine SAR missions using an autonomous UAV with a boat as a base of operations. The work in this thesis provides contributions toward the part of the problem related to landing the drone once the mission is finished, but multiple other subsystems must be implemented for the system to be able to have any meaningful contributions in a SAR system. The base systems, in addition to the ones implemented in this thesis, needed to carry out a search mission are

- local navigation for travelling larger distances
- path planning for effectively searching an area
- detection for finding victims during the search

Local navigation is necessary for the drone to be able to fly to a search area as well as return to the boat. For this GPS sensors for position measurement and communication with the boat as described above can be incorporated into the Kalman filter.

When arriving at the search area, an efficient search plan must be generated. The literature on the subject should be investigated to find the most efficient patterns that maximize the chance of finding victims fast.

Detection of victims can be done by using a standard camera with object detection based on, e.g. the same YOLO detector as used in this thesis. The detector could be trained on images of floating objects, where one such large dataset was recently released

[74]. It can also be beneficiary to use thermal imaging when discovering bodies, and this could similarly be paired with some form of object detection.

It is also important that the drone used is properly equipped to handle the mission. It has to be able to handle harsh environmental conditions such as severe wind and rain, as these often accompany such missions. To this end, the Anafi drone is not suited as it is not waterproof and not designed for flight in particularly windy conditions. It also has a short battery life, meaning that it would not be able to carry out a substantial search before having to replace its battery. It is also beneficial for the drone to carry a payload in the form of a lifebuoy that can be dropped at command, another feature the Anafi lacks.

The existing system implemented in this thesis would also have to be improved for the system to be used in SAR missions. The perception system would need improved accuracy and decreased delay and be robust enough to work with potentially limited visibility and varying lighting conditions. The control system must be robust enough to land on the boat while at sea with potentially harsh conditions, contrary to the experiments in this thesis where the motion of the ReVolt was calm and controlled due to manually moving it by hand. Mission planning and execution is also an important area that would have to be robust with failsafe mechanisms and multiple scenarios planned due to the uncertainty of such missions.

Overall, this thesis explores the foundations of some of the fundamental systems needed in such SAR systems, but more development and experimentation would need to be done for these to be used in an actual SAR mission.

7 Conclusion and future work

This chapter concludes the thesis by summarizing the findings from the results and discussion chapters before outlining potential future work based on the results obtained in this thesis.

7.1 Conclusion

This thesis has investigated methods within the fields of perception, control, and mission planning with the end goal of having a complete system capable of landing an unmanned aerial vehicle (UAV) autonomously on a boat. The experiments were carried out using a Parrot Anafi quadcopter drone which landed on a helipad attached to the DNV ReVolt vessel.

The perception system proposed was composed of two different computer vision position estimation techniques combined in a model-based Kalman filter. The first computer vision system developed was based on traditional computer vision (TCV) techniques, including corner detection and homography-based pose estimation, and was a continuation of the work done in the specialization project preceding this thesis [3]. Hyperparameter optimization (HPO) was in this thesis introduced to make the corner detection more reliable. It was used to find the best parameters for a Hough circle detector that segmented out the helipad and a corner detector that detected the corners and ”H” on the helipad. Using HPO was found to improve the rate at which the known corners of the helipad ”H” and arrow were found and identified in the image, but the system struggled due to varying runtimes of the circle detector and few correct identifications when testing outside. It was suggested to use a deep neural network (DNN) based circle detector instead to improve detection accuracy and decrease runtime, as well as using fiducial markers placed on the helipad to make the known points easier to detect and identify. The homography-based pose estimation system showed promising results when the known points were found but suffered from offsets in all axes.

The other computer vision position estimation technique used a deep neural network-based computer vision (DNN-CV) method from [1]. This method is based on determining the drone position based on detecting the helipad with the YOLO object detector and was shown to produce frequent estimates in indoor and outdoor real-world experiments. However, the method was found to include offsets, particularly in the z-axis position estimates, due to the detector estimating the size of the helipad bounding boxes to be too large. Another issue was false detections of the helipad, causing the drone to lose track of the helipad if it was not in view of the camera. It was suggested that both of these issues could be mitigated by retraining the YOLO detector on new images from the Parrot Anafi, as the current detector is only trained on lower-resolution images from the AR.Drone 2.0 used by [1]. A final issue was found to be that system was unable to estimate the altitude when more than the entire helipad was in the camera view, as the bounding boxes were unable to estimate the actual size of the helipad. It was therefore concluded that the DNN-CV system should be used for position estimation only when the drone is farther away from the helipad and not during the final parts of the landing.

The TCV and DNN-CV systems were combined with velocity measurements from the drone in a model-based Kalman filter using the constant velocity (CV) model for predictions. The performance of the filter was found to be sufficiently accurate to be used in feedback to land the drone on the helipad, and the overall accuracy was better than for the TCV and DNN-CV systems by themselves. The main cause of inaccuracies in the filter was the inaccurate measurements from the TCV and DNN-CV systems as well as time-delayed velocity measurements used to update the CV model. Although the CV model could accurately predict the motion of the drone on its own over short periods of time, the large delay from the velocity measurements caused conflict with the less delayed DNN-CV measurements. This was most notable when there was a significant change in the drone's velocity, such as at the start of a maneuver. It was concluded that using velocity measurements accompanying the images streamed from the drone would mitigate this issue as these measurements appear to be less time delayed. One of the limitations of the implemented filter was that the estimation was limited to having the helipad in view most of the time and requiring the helipad to be stationary while not in view of the camera. To allow the drone to travel farther away from the helipad and still return, it was suggested to extend the position estimates to also include the drone position in the NED frame using GPS measurements. It was also suggested to estimate the helipad position in the NED frame as well with methods for updating this estimate when the helipad is not in view of the camera.

The overall control system proposed was a system based on individual controllers for attitude, velocity, and position, combined using successive loop closure. The attitude controller was internal on the Anafi, while methods for the velocity and position controller were developed and compared.

For velocity control, the goal was to generate the attitude reference needed to achieve a given velocity reference. The first method tested for this was an open-loop model-based solution for calculating the reference based on the forces acting on the drone. The velocity response from using this method was pretty good but a little slow. A PID controller was also tested to generate this attitude reference, and this showed similar results to the model-based method, but with a slightly faster and smoother response. The PID method was therefore chosen. Regardless of the method used, the main issues with the velocity controller were transmission and measurement delays. The transmission delays meant the drone motion was delayed compared to the commanded references. Therefore, it was concluded that target tracking with aggressive maneuvering should be avoided as the delays could cause instability. A similar argument was made regarding the delays in the feedback measurements, as these could also cause instability. These measurement delays could also be reduced by updating the drone interface to use the more reliable velocity measurements accompanying the images sent from the Anafi.

For position control, the goal was to generate the velocity references needed to align the Anafi with the helipad horizontally and descend the drone towards the helipad for landing. Two methods were investigated for the horizontal tracking of the helipad, namely, the pure pursuit (PP) guidance law and point stabilization using a PID controller. The PP guidance law was found to oscillate around the helipad center without stabilizing and was therefore not used. The PID method proved to be able to align the drone with the helipad well when the helipad was fully or nearly stationary. It suffered from some oscillations when stabilizing in the real-world experiments, and a contributing factor to this was identified to be the time delay of the position estimates from the Kalman filter. The PID controller was also not able to track the helipad well when it was moving at significant speeds, as the drone always trailed behind the helipad. For vertical control, a PID controller was used, and this was found to regulate the altitude well during landings.

It was investigated if the use of AI planning could be used for the mission planning of such an autonomous UAV. Specifically, the AI planning algorithm Graphplan was used to generate action sequences to solve three missions of different complexity based on a set of defined domain variables specific to the problem at hand. The algorithm was able to compute valid action sequences solving all three missions, thereby demonstrating that the use of this algorithm is feasible in such a scenario. Using such a system of generating standardized actions to execute was found to be beneficial as it allowed for a simple mission executor that could handle missions of any complexity. It was, however, discussed that the missions defined were not complicated enough to fully evaluate the potential of using this algorithm in this domain. The potential upsides of using the algorithm could be better evaluated by having more complicated missions with possibly multiple UAVs. It was identified that further research is necessary to make any definite conclusions about the effectiveness of AI planning for autonomous UAVs for SAR missions.

Landing the Anafi on the helipad using the complete system of perception, control, and mission planning and execution was tested inside in the drone lab, where motion capture ground truth was available, and outside landing on the DNV ReVolt vessel while on land and at sea. The drone was able to land on the helipad successfully in all cases where the helipad was stationary. Inside the drone was also able to land on the helipad while continuously moving slowly, but these results could not be replicated when tested outside, as here the drone constantly trailed behind the helipad and was therefore not able to land on it. This was concluded to be because of the guidance system only being a point stabilization system and therefore not being able to follow a moving helipad. It was therefore concluded that the successive loop closure using a PID position controller and velocity controller used in this thesis is sufficient for landing the drone reliably at a stationary helipad. However, better-suited guidance laws would have to be used to allow the drone to land on a helipad with significant movement. The perception system was able to determine the drone's position in the experiments when the helipad was parallel to the ground but struggled when the helipad was mounted on the ReVolt while rolling at sea. It was therefore concluded that the current perception system using computer vision in this Kalman filter is adequate for estimating the drone's position relative to the helipad while landing if the helipad is parallel to the ground. However, the filter would have to be modified to include the dynamics of the helipad to handle landings at sea reliably.

The results of the thesis were discussed in relation to the overall use case of using autonomous UAVs in SAR missions. It was found that additional capabilities were needed in addition to the ones in this thesis for the system to have a meaningful contribution to SAR missions. These included navigating in larger areas, path planning for effective search, and detection of victims in the water. It was also concluded that the Anafi UAV used in this thesis is not suited for such missions due to its short battery life, lack of water and heavy wind resistance, and inability to carry supplies such as a lifebuoy which could be delivered to victims in the water. The systems implemented in this thesis were also found to need further improvements to be reliable enough to use in SAR missions. Both the perception and control systems would have to be more robust to handle harsh conditions, and the mission planner would have to be able to handle mission uncertainty.

7.2 Future work

This section will now briefly discuss the future work that the author has identified as potential areas of improvement for the project. The future work will be split into perception, control, and mission planning, as these are the areas investigated in this thesis, as well as some ideas to other areas of improvement. The ideas are summarized in Table 7.1.

For perception, the shortcomings of the TCV and DNN-CV systems should be addressed. The Kalman filter should be extended to model the dynamics of the helipad as well to aid in landings at sea, while also incorporating GPS measurements to be able to navigate greater areas around the helipad.

For control, the most significant improvement would be finding a new guidance law that allows the drone to follow the helipad without trailing some distance behind, allowing the drone to properly land on a moving platform.

Future work regarding mission planning involves further evaluating whether AI planning algorithms such as Graphplan are suitable for this type of application, which could be done by increasing the mission complexity. In addition to this, it could be advantageous to have a robust mission planner able to handle multiple mission scenarios, so developing a mission controller capable of dynamically adapting the mission as it unfolds is a promising area of future research.

For the anafi_ros interface, a necessary improvement is to use the velocity and attitude measurements accompanying the images rather than the ones currently used. For the system to be used in SAR missions, the work in this thesis would have to be ported to a waterproof drone, as well as creating a complete search system, including path planning to search an area and object detection to detect victims in the water. Finally, some experiments that could be useful to attempt are landing the drone on the ReVolt while floating and moving freely at sea, as the ReVolt was held in place manually during the experiments in this thesis.

Table 7.1: Possible areas of future research.

Perception	
TCV	• Use DNN-based object detector to segment helipad. • Test using fiducial tags instead of corners as known points in homography pose estimation. • Find cause of offsets in estimates.
DNN-CV	• Retrain detector based on images from the Anafi to avoid false detections. • Remove estimates at low altitudes.
KF	• Extend states to estimate drone attitude and the motion of the helipad as well. • Estimate position in NED frame using GPS sensor.
Guidance and control	
Guidance	• Experiment with different guidance laws to be able to track the helipad while moving.
Mission planning	
General	• Increase mission complexity to properly evaluate Graphplan potential. • Make mission planner adaptive.
Other	
Drone interface	• Use attitude and velocity measurements accompanying images.
Search and rescue	• Port results to waterproof drone. • Implement path planning to search for victims. • Implement object detection of victims.
Experiments	• Test landing on ReVolt while floating freely at sea. • Test landing on ReVolt while moving freely at sea.

Bibliography

[1] 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.

[2] T. Sundvoll, “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.

[3] M. Falang, “Ros interface and perception system for autonomous quadcopter,” Specialization project, O. S. Bragstads Plass 2D, 7034 Trondheim, 2021.

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

[5] J. Redmon, “Darknet: Open source neural networks in c,” http://pjreddie.com/darknet/, 2013–2016.

[6] L. Takayama, W. Ju, and C. Nass, “Beyond dirty, dangerous and dull: What everyday people think robots should do,” in *2008 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2008, pp. 25–32.

[7] R. Geirhos, D. H. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, “Comparing deep neural networks against humans: object recognition when the signal gets weaker,” *arXiv preprint arXiv:1706.06969*, 2017.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[9] A. Adams, T. Schmidt, C. Newgard, C. Federiuk, M. Christie, S. Scorvo, and M. De-Freest, “Search is a time-critical event: When search and rescue missions may become futile,” *Wilderness & environmental medicine*, vol. 18, pp. 95–101, 02 2007.

[10] 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.

[11] 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. [Online]. Available: https://doi.org/10.1080/19475705.2016.1238852

[12] A. Gautam, S. P.B, and S. Saripalli, “A survey of autonomous landing techniques for uavs,” 05 2014, pp. 1210–1218.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

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

[15] A. L. Salih, M. Moghavvemi, H. A. F. Mohamed, and K. S. Gaeid, “Modelling and pid controller design for a quadrotor unmanned air vehicle,” in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, vol. 1, 2010, pp. 1–5.

[16] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, “Model predictive control for micro aerial vehicles: A survey,” *CoRR*, vol. abs/2011.11104, 2020. [Online]. Available: https://arxiv.org/abs/2011.11104

[17] A. L. Blum and M. L. Furst, “Fast planning through planning graph analysis,” *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[18] L. H. Henriksen, “Hovering control of a quadrotor using monocular images and deep reinforcement learning,” Master’s thesis, Norwegian University of Science and Technology, O. S. Bragstads Plass 2D, 7034 Trondheim, 2019.

[19] A. Rodríguez Ramos, C. Sampedro Pérez, H. Bavle, P. de la Puente, and P. Campoy, “A deep reinforcement learning strategy for uav autonomous landing on a moving platform,” *Journal of Intelligent & Robotic Systems*, vol. 93, 02 2019.

[20] O. Araar, N. Aouf, and I. Vitanov, “Vision based autonomous landing of multirotor uav on moving platform,” *Journal of Intelligent & Robotic Systems*, vol. 85, 02 2017.

[21] M. Monajjemi, “ardrone_autonomy,” https://ardrone-autonomy.readthedocs.io/en/latest/, accessed: 2020-11-16.

[22] A. L. Salih, M. Moghavvemi, H. A. F. Mohamed, and K. S. Gaeid, "Modelling and pid controller design for a quadrotor unmanned air vehicle," in *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, vol. 1, 2010, pp. 1–5.

[23] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, vol. 22, p. 22, 2011.

[24] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *ArXiv*, vol. abs/2109.04210, 2021.

[25] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 298–304.

[26] S. Grzonka, G. Grisetti, and W. Burgard, "A fully autonomous indoor quadrotor," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 90–100, 2012.

[27] A. Mohammadi, Y. Feng, C. Zhang, S. Rawashdeh, and S. Baek, "Vision-based autonomous landing using an mpc-controlled micro uav on a moving platform," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 771–780.

[28] P. Doherty and P. Rudol, "A uav search and rescue scenario with human body detection and geolocalization," in *AI 2007: Advances in Artificial Intelligence*, M. A. Orgun and J. Thornton, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–13.

[29] S. P. Yeong, L. M. King, and S. S. Dol, "A review on marine search and rescue operations using unmanned aerial vehicles," 2015.

[30] G. De Cubber, D. Doroftei, D. Serrano, K. Chintamani, R. Sabino, and S. Ourevitch, "The eu-icarus project: Developing assistive robotic tools for search and rescue operations," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013, pp. 1–4.

[31] "The first responder of the future," https://ingenious-first-responders.eu/ingenious-project/, accessed: 2022-05-29.

[32] "Ingenious," https://www.sintef.no/en/projects/2019/ingenious/, accessed: 2022-05-29.

[33] J. Kim, M.-S. Kang, and S. Park, "Accurate modeling and robust hovering control for a quad–rotor vtol aircraft," *Journal of Intelligent and Robotic Systems*, vol. 57, pp. 9–26, 01 2010.

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

[35] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003.

[36] M. W. Spong, S. Hutchinson, M. Vidyasagar *et al.*, *Robot modeling and control*. Wiley New York, 2006, vol. 3.

[37] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft. Theory and Practice*. New Jersey, United States of America: Princeton University Press, 2012.

[38] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," 08 2007.

[39] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3277–3282.

[40] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[41] 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. [Online]. Available: http://arxiv.org/abs/1910.13796

[42] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: A hands-on survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 12, pp. 2633–2651, 2016.

[43] D. A. Forsyth and J. Ponce, *Computer Vision - A Modern Approach, Second Edition.*, 2nd ed. Prentice Hall, 2012.

[44] E. Olson, "Apriltag: A robust and flexible visual fiducial system," 06 2011, pp. 3400 – 3407.

[45] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *CoRR*, vol. abs/1502.00956, 2015. [Online]. Available: http://arxiv.org/abs/1502.00956

[46] C. Harris, M. Stephens *et al.*, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[47] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[48] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[49] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[50] S. J. K. Pedersen, “Circular hough transform,” *Aalborg University, Vision, Graphics, and Interactive Systems*, vol. 123, no. 6, 2007.

[51] M. A. Nielsen, “Neural networks and deep learning,” 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[52] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[53] M. Claesen and B. De Moor, “Hyperparameter search in machine learning,” 2015. [Online]. Available: https://arxiv.org/abs/1502.02127

[54] E. Brekke, *Fundamentals of Sensor Fusion. Target tracking, navigation and SLAM.*, 3rd ed., 2020.

[55] M. Breivik and T. Fossen, *Guidance Laws for Autonomous Underwater Vehicles*, 01 2009.

[56] H. K. Khalil, *Nonlinear systems third edition*. Patience Hall, 2002, vol. 115.

[57] B. Foss and T. A. N. Heirung, *Merging Optimization and Control*, 03 2016.

[58] J. G. Balchen, T. Andresen, and B. Foss, *Reguleringsteknikk*, 6th ed. Institutt for teknisk kybernetikk, NTNU, 2016.

[59] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[60] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.

[61] R. E. Fikes and N. J. Nilsson, “Strips: A new approach to the application of theorem proving to problem solving,” *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[62] D. S. Weld, “Recent advances in ai planning,” *AI magazine*, vol. 20, no. 2, pp. 93–93, 1999.

[63] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “Pddl-the planning domain definition language,” 1998.

[64] *ANAFI White Paper*, Parrot, v1.4.

[65] “Olympe documentation,” https://developer.parrot.com/docs/olympe/, accessed: 2022-05-23.

[66] *ANAFI Guide*, Parrot, v1.4.

[67] “New eu-regulations,” https://luftfartstilsynet.no/en/drones/new-eu-regulations/, accessed: 2021-11-16.

[68] "Parrot sphinx guide book," https://developer.parrot.com/docs/sphinx/masterindex.html, accessed: 2022-05-23.

[69] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[70] P. A. Nogueira, "Motion capture fundamentals a critical and comparative analysis on real-world applications," 2012.

[71] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[73] F. van Breugel, J. N. Kutz, and B. W. Brunton, "Numerical differentiation of noisy data: A unifying multi-objective optimization framework," 2020. [Online]. Available: https://arxiv.org/abs/2009.01911

[74] J. Gasienica-Jozkowy, M. Knapik, and B. Cyganek, "An ensemble deep learning method with optimized weights for drone-based water rescue and surveillance," *Integrated Computer-Aided Engineering*, vol. 28, no. 3, pp. 221–235, 2021.