Peter Bull Hove

Perception and High-Level Control for Autonomous Drone Missions

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2021

Peter Bull Hove

Perception and High-Level Control for Autonomous Drone Missions

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2021

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

Abstract

An autonomous quadcopter system is proposed in this thesis, enabling a quadcopter to perform autonomous missions starting and ending on a specified landing platform.

A pose estimation algorithm is developed for estimating the quadcopter's pose in relation to a specified landing platform. It uses a multi-sensor Kalman Filter for fusing measurements from a traditional computer vision method pose estimation, a deep learning-based computer vision method for pose estimation, IMU data, barometric pressure data and GPS measurements.

A PID-controller is used for low-level control of the quadcopter, providing velocity commands to an on-board quadcopter controller based on the errors between the estimated- and the desired pose.

A mission control system is proposed for high-level quadcopter control, used for completing autonomous quadcopter missions. The control system is based on a finite state machine approach, where a mission is constituted of sequential execution of actions defined by a set of states, where the states represent movement- and camera actions.

The autonomous quadcopter system is applied to a Parrot AR.Drone 2.0 and is tested extensively both in simulations and experimentally. GPS is only available in simulations.

Encouraging results were seen in simulations where the quadcopter system accurately estimated the quadcopter's pose and used this pose for autonomous take-off and landing missions and autonomous long-range missions.

Experimental results with the Parrot AR.Drone 2.0 demonstrated successful autonomous take-off and landing mission on a landing platform, as well as being able to hover above the landing platform. Autonomous hovering and landing were successfully demonstrated on the landing platform mounted on the DNV GL ReVolt marine vessel.

Experimental results were subject to more sensor noise, a longer communication delay and more disturbances than simulation results making the quadcopter flight response less stable than in simulations.

Acknowledgements

Producing this thesis would have been substantially more difficult if not for the guidance and support I have received from many wonderful people.

I want to thank my supervisor Anastasios Lekkas for invaluable guidance and feedback along the way. He has been instrumental in the process, originally proposing the idea for the thesis, and providing me with all the equipment and help I required.

Furthermore, I would like to thank Thomas Sundvoll. He finished his master thesis in 2020 about computer vision pose estimation for quadcopters, and my work builds upon the foundation laid down by him.

Next, I would like to thank Tom-Arne Pedersen and Eldar Sandanger, who has provided me with access to the ReVolt marine vessel for experimentation.

In the end, I would like to thank my mother and father, my friends and Elin, and the rest of my family for being who they are, and always being there for me.

Contents

List of Figures

List of Tables

Preface

This thesis is written in 2021 in Trondheim as the culminating thesis of my Master's degree in Cybernetics and Robotics at NTNU.

The topic of this thesis is motivating to me because I have always been interested in automation. Automation has been on the mind of humans for thousands of years. Humans learned to reduce labor by utilizing animal power for transport and agriculture. Later humans learned to use water power for milling wheat into flour, using wind for moving ships and harvesting energy from heat by inventing the steam engine. Computers, robotics and artificial intelligence are modern tools that can reduce the labor required by human beings. Automation using these tools can free up time and capacity for people, such that they may continue innovating and to have more time to spend on other things.

The goal of this project and subsequent thesis is to create and present a quadcopter system with a sufficient level of autonomy to perform simple autonomous missions. The project is part of a cooperation with DNV, who have an ongoing project concerning an autonomous marine vessel called the ReVolt. Cooperation between NTNU and DNV sparked the idea of the quadcopter system performing autonomous missions starting and ending on the ReVolt.

My supervisor during this project has been Anastasios Lekkas. He has been supportive of me the whole duration of the project and has been providing me with invaluable feedback and guidance along the way.

The work in this thesis builds upon a basis of work by Thomas Sundvoll [1] who, in his master thesis, proposed a computer vision pose- (position and orientation) estimation system for quadcopters based on traditional computer vision techniques. This thesis utilizes the following contributions from Thomas Sundvoll's work:

- Setup of a computer simulation environment used extensively in the development of this quadcopter system.
- A physical landing platform which is the start- and end-point for most experiments performed in this thesis.
- A computer vision module based on traditional computer vision techniques for detecting the landing platform using a camera mounted on a quadcopter.
- A PID controller for quadcopter control which provided the basis for the further developed PID controlled presented in this thesis.
- A Moving Median Averaging filter for filtering sensor output.

This thesis also builds upon one of my previous projects, where I developed a deep-learning based computer vision pose estimator [2] and combined that with Sundvoll's [1] pose estimator. The specific contributions from that project that are used in this thesis are

- a dataset consisting of images of a landing platform captured from the simulation environment, which in this thesis is used as basis for a further developed dataset.
- a deep-learning based computer vision pose estimator for quadcopters which was further improved and adapted in this thesis.

Anastasios Lekkas, and The Department of Engineering Cybernetics has generously provided me with

- A Parrot AR.Drone 2.0 quadcopter.
- OptiPlex 7040 Computer with Intel Core i7-6700 CPU and 32 GB RAM.
- Komplett Khameleon P9 Pro Laptop Computer with a GeForce RTX 2070 GPU, Core i7-9750H CPU and 32GB RAM

which were components that were instrumental for the development of the quadcopter system proposed in this thesis.

DNV, represented by Tom-Arne Pedersen and Eldar Sandanger have generously provided me with

- a 3D Model of ReVolt vessel used in simulations of the quadcopter
- access to the ReVolt vessel for experiments with the quadcopter system
- video documentation of the experimental quadcopter missions on the ReVolt

The work in this thesis has benefited from utilization of a number of open-source software and and free software tools including

- CVAT computer vision annotation tool
- Python 2.7 as well as numerous python packages, including but not limited to matplotlib, numpy, scipy, rospy and openCV
- ROS and Gazebo by Open Robotics
- Google Colaboratory training GPUs
- The open-source ROS package "Darknet For ROS" by Leggedrobotics [3]
- Roboflow data augmentation software [4].

All images, plots and figures are generated by the author unless otherwise stated.

Chapter 1

Introduction

1.1 Background and Motivation

In recent years autonomous robots and artificial intelligence (AI) and have been used in sectors such as surveillance, delivery, military, farming and security [5] [6]. New milestones are being reached every year within the field of robotics development and AI, which is laying the foundation for further innovation within the field of autonomous robots.

Autonomous robots may offer several advantages compared to their manually controlled counterparts and human personnel performing the same task. Admittedly, many autonomous systems are not par with manually controlled systems. Autonomous systems require a lot of research and development to overtake the skill of humans. Nevertheless, more and more autonomous systems have reached that level, showing that autonomous systems can have the capacity to outperform humans [7]. [8]. Manually controlled robots generally require trained operative personnel which can be costly. Human personnel may require training to perform a task well, and for certain applications there may be a lack of skilled operators. Human operators may still be in the loop, supervising the autonomous systems. A well developed autonomous robot system may:

- make it possible for human operators to supervise multiple autonomous systems, increasing the efficiency of the personnel
- be cheap to operate
- have the capacity of split-second reactions because of efficient computation speed and may therefore be more safe and reliable than their human counterparts [9].
- have the capacity to operate 24 hours a day, seven days a week, much more than any human capacity.

Quadcopters, also called drones, are a type of micro aerial vehicle (MAV) consisting of four rotor blades mounted in an X-shape pattern. Drones have a frame connecting the rotor blades and a body in the center of the frame. Quadcopters benefit from being generally low-cost, easy to operate, and have impressive and delicate maneuverability capabilities, including vertical take-off and landing.

Both professionals and hobbyist currently use quadcopters for video and photo capture. In addition, they are used in the industry for inspection- and cleaning of hard-to-reach- and dangerous locations, package delivery, automatic inspection missions, search missions for people lost at sea or in the mountains, and defence- and military applications [10] [11] [12].

1.2 Objectives and Contributions

This thesis aims to work towards an autonomous quadcopter system, able to perform simple missions without human intervention.

The contributions in this thesis are specifically

- Creating a dataset consisting of labeled images of a landing platform, containing images from both a simulated landing platform and a physical landing platform, for use in supervised DNN training.
- Further development of a computer vision (CV) pose estimation algorithm proposed initially by the author in a previous thesis [2].
- A sensor fusion algorithm based on a Kalman Filter approach for combining pose estimates from multiple computer vision methods, as well as GPS data, barometric pressure data, IMU- and magnetometer measurements.
- A mission control system for implementation- and execution of autonomous quadcopter missions, including take-off, flight maneuvers, photo capture and landing.
- An autonomous quadcopter system combining the sensor fusion pose estimation system with the mission control system.
- Experiments with- and simulations of the autonomous quadcopter system to test the robustness and performance of the quadcopter system. Experiments were also performed by taking off from and landing on the DNV ReVolt marine vessel.

1.3 Previous Work

The problem of autonomous control of quadcopters has been a topic of research that has received much interest.

This work builds upon the foundation laid by the author's project thesis, conducted in 2020 [2]. The project thesis deals with the topic of position- and orientation estimation of quadcopters using a combination of traditional-based and deep learning-based computer vision methods. A deep-learning based computer vision method, proposed in the author's project thesis, is used as a component of the autonomous quadcopter system proposed in this master thesis.

This work also builds upon the foundation laid by Sundvoll [1], who proposed a quadcopter pose-estimation algorithm using a traditional computer vision (TCV) method. This TCV method is used as a component of the quadcopter system proposed in this thesis. The pose estimate is a pose relative to the landing platform and requires that the landing platform be visible in the camera frame to produce estimates. This algorithm was tested using a quadcopter simulation and experimentally with a Parrot AR.Drone 2.0. The algorithm produced precise- and accurate pose estimates for the quadcopter in the simulated environment. This approach, however, lacked robustness as it failed to deliver estimates if the whole landing platform was not in the camera frame. When applied real-world quadcopter system it was not able to produce accurate estimates.

Much of the work on quadcopters presents computer-vision-based pose estimation techniques due to encouraging results with such approaches and cameras being cheap and lightweight, making them suitable for quadcopter applications. Some approaches in literature present traditional computer vision-based pose estimation solutions, while others present deep learning-based methods.

Other approaches are proposed for pose estimation using TCV methods. A popular method uses predefined computer vision detectable markers called ArUCO markers [13] [14]. The location, skewness, and orientation of these markers can be used to calculate the pose of the camera relative to these markers. Approaches using ArUCO markers has been shown in by Sani et al. and Carriera et al. to be sufficiently precise to enable automated landing of quadcopters in small-scale simulations and experiments.

Visual Simultaneous Localization and Mapping (vSLAM) [15] is a TCV method that was used in [16] for estimating pose of micro aerial vehicle (MAV) in an indoor landing scenario, by landing on target images on the floor. Pose estimation for hovering and control of the AR.Drone has been performed using vSLAM [17].

Deep learning (DL) approaches to computer vision (CV) have become popular due to recent research and improvements of DL methods. Bounding box prediction algorithms such as YOLOv4 [18] are used for CV object tracking of a number of different objects, and has been proven to be efficient and precise [19] [20]. Real-time bounding box detection algorithms have applied to live data from quadcopter camera [21]. Other DL-based methods such as reinforcement learning have been applied for autonomous quadcopter control [22] using external camera video of the quadcopter. A specialized DNN structures has been created for quadcopter flight in different scenarios, including TrailNet [23]. TrailNet is a DNN for estimating quadcopter pose relative to forest trails.

Sensor fusion, where sensor data from multiple sensors is combined into one estimate, has been applied to pose estimation of quadcopters. Kalman Filter (KF) sensor fusion produces smooth and precise pose estimates, verified by Brockers et al. [24], and Sani et al. [13]. Both used a Kalman Filter for fusing visual data and inertial IMU sensors for quadcopter pose estimates.

An Error State Kalman Filter (ESKF) has also been applied for mobile robot localization [25]. In that application it replaced the need for a complex dynamic model in KF localization applications.

Zhang et al. demonstrated fusing a dynamic system model with a bounding box detection algorithm using a Kalman Filter [19]. Their algorithm was applied to golf ball tracking.

Alantise et al. applied an Extended Kalman filter to a small 4-wheeled robot to fuse IMU data with visual data gathered by point matching between camera frames [26]. GPS data, sonar- and inertial measurements were fused using a KF approach for quadcopter control by Gustavsson [27].

Quadcopters performing autonomous missions require a mission control system for navigating and controlling the quadcopter during the mission. A Finite State Machine (FSM) approach has been used in the development of quadcopter mission systems. Rabbath et al. [28] developed a mission system for payload delivery. Their approach used multiple cooperative drones for delivering a payload. The quadcopter mission control was implemented as a series of states that, when executed sequentially, constitute a mission. Ghallabi et al. [29] implemented quadcopter control and navigation as an FSM, where the quadcopter states were hovering, altitude control, path tracking or yaw control.

Several mission control systems have been developed for quadcopters. Luo et al. [30] created a system for intelligent control and navigation of an indoor quadcopter and performed a payload drop-off mission with a combination of manual control and automatic landing. Haque et al. parcel developed a delivery quadcopter system where Google Maps' pedestrian path planning software was used for path planning. The drone was able to follow this path using GPS measurements [10]. Bernardini et al. [12] propose a Searching and Tracking (SaT) system using the Parrot AR.Drone 2.0, with extensive planning capabilities, making decisions based on remaining battery capacity, position uncertainty metrics, and more. A return-to-home quadcopter system is developed by Nguyen et al. [31], where the destination is set using GPS coordinates captured during the start of a quadcopter mission.

1.4 Outline

This thesis is outlined such that theory essential for understanding the implementation and results is presented in Chapter 2. The experimental setup, software, and how to replicate these experiments are outlined in Chapter 3. The methodology and implementation of the quadcopter system are proposed in Chapter 4. Results from both simulations and experiments are presented in Chapter 5 and discussed in Chapter 6. The thesis is concluded in Chapter 7 in addition to a discussion of possible future work for continuing this project.

Chapter 2

Theory

2.1 Quadcopter Dynamics

A quadcopter is a 4-rotor aircraft with rotors mounted in a X-shaped pattern. An illustration of a quadcopter is displayed in Figure 2.1. A separate motor powers each rotor, and each rotor can be controlled individually. Quadcopter flight is controlled by applying a variable voltage each engine. The propeller layout permits precise control of speed, position and rotation making it possible to control the quadcopter in 6 degrees of freedom (DoF) using only the four stationary motors.

The lift that is generated from each motor varies from aircraft to aircraft. From literature, [32] it is known that the force generated by a propeller is linearly dependent on the square of the angular velocity of the propeller. Thus, the force generated by motor i, is defined as

$$F_i = b\omega_i^2$$

for $i = 0, .., 4$, where b is a propeller-speed-to-lift constant which is dependent of propeller size and shape.

The torque which each motor applies to the quadrotor is the force from the motor, multiplied by the length of the arm from the motor to the center of mass (CoM) of the quadcopter. Defining the length from quadrotor CoM to the motors as l, being equal for all motors, then motor torque for motor i, τ_i is defined as

$$\tau_i = F_i l$$

for $i = 0, .., 4$.

The vector sum of all forces acting upon the quadcopter controls the acceleration of a quadcopter. The force vector of the sum of all motor forces defines the direction of the acceleration caused by the motors.

Assuming the motors are mounted an symmetrically around the center of the quadcopter, then the force vector depends solely on the pitch and roll of the quadcopter, pointing directly upwards when pitch and roll are zero. If the quadcopter pitches forward, the force vector pitches forward as well. The vertical component of the motor force vector provides lift, while the horizontal component of the force vector provides a horizontal acceleration in the x-axis. The horizontal component of the vector is equal to the sine of pitch multiplied by the sum of motor forces. Similarly, if the quadcopter rolls, then the quadcopter will be subject to an acceleration in the y-axis, equal to the sine of roll multiplied by the sum of motor forces. The lift experienced by the quadcopter is equal to the upward component of the force vector and is therefore equal to the force vector multiplied by the cosine of pitch and the cosine of roll. In sum, quadcopter control is achieved by controlling pitch and roll orientations while at the same time controlling the sum of all motor forces.

Pitch is controlled by the force differential between motors 3 and 1 and roll is controlled by the force differential between motors 4 and 2. Yaw is controlled by torque in the yaw direction caused by spinning the motors, called $\tau_{\psi,i}$ for motor i. The rotors are mounted such that they are spinning in opposite directions. Yaw is controlled by the yaw torque differential between the motor pairs 1, 3 and 2, 4.

The dynamic equations representing quadcopter movement based on motor force and orientation are presented by Kim et al. [32] and are:

$$\begin{aligned}
m\ddot{x} &= -\sum_{i=1}^{4} F_i \sin\theta \\
m\ddot{y} &= -\sum_{i=1}^{4} F_i \sin\phi\cos\theta \\
m(\ddot{z}-g) &= -\sum_{i=1}^{4} F_i \cos\phi\cos\theta \\
I_{xx}\ddot{\phi} + \dot{I}_{xx}\dot{\phi} &= \tau_4 - \tau_2 \\
I_{yy}\ddot{\theta} + \dot{I}_{yy}\dot{\theta} &= \tau_1 - \tau_3 \\
I_{zz}\ddot{\psi} + \dot{I}_{zz}\dot{\psi} &= \tau_{\psi,2} + \tau_{\psi,4} - \tau_{\psi,1} - \tau_{\psi,3}
\end{aligned}$$

where:

- $\ddot{x}$, $\ddot{y}$, $\ddot{z}$ are accelerations in body frame x-, y- and z-axis.
- θ is pitch angle-, ϕ is roll angle- and ψ is yaw angle of the quadcopter. These angles are defined as the angles between the body frame axes to the inertial frame axes.
- I_{xx}, I_{yy}, I_{zz} are the moments of inertia of rotations around the x, y, and z axis in the body frame respectively.
- $\tau_{\psi,i}$ is the torque in yaw of rotor i around the CoM of the quadcopter due to inertial forces of the spinning propellers.
- g is the gravitational constant,
- m is quadcopter mass

In order to simplify the dynamic model, the following control inputs are chosen:

$$\begin{aligned}
u_1 &= F_1 + F_2 + F_3 + F_4 \\
u_2 &= \tau_4 - \tau_2 \\
u_3 &= \tau_1 - \tau_3 \\
u_4 &= \tau_{\psi,2} + \tau_{\psi,4} - \tau_{\psi,1} - \tau_{\psi,3}
\end{aligned}$$

The moment of inertia is constant for quadcopter's whose frame is fixed, meaning that $\dot{I}_{xx} = \dot{I}_{yy} = \dot{I}_{zz} = 0$. By inserting the control inputs, and assuming the quadcopter body to be a fixed frame, the quadcopter equations are reduced to

$$\begin{aligned} m\ddot{x} &= -u_1 \sin\theta \\ m\ddot{y} &= -u_1 \sin\phi\cos\theta \\ m(\ddot{z} - g) &= -u_1 \cos\phi\cos\theta \\ I_{xx}\ddot{\phi} &= u_2 \\ I_{yy}\ddot{\theta} &= u_3 \\ I_{zz}\ddot{\psi} &= u_4 \end{aligned}$$

This dynamic model does not account for any flight aerodynamics other than propeller thrust, such as drag or air pressure differentials caused by the spinning rotor blades. These aerodynamic forces are presumed to be negligible at close to hover, although while moving at higher speeds, this model will be less accurate. Although the wind may be a significant disturbance when flying in an outdoor environment, it is not accounted for in this model. The experiments in this thesis do not involve high-speed quadcopter maneuvers and are mostly performed in indoor environments; therefore, the model is deemed adequate for this thesis.

2.1.1 Quadcopter Body Frame

Body frame coordinates are represented with axes fixed to elements of a mobile body, contrary to coordinate frames represented by axes fixed to the earth. A number of such fixed coordinate systems exists. Fossen presents [33] the following Earth-fixed coordinate frames:

- **ECI**: Earth-Centered Inertial frame, which is an inertial frame where Newton's laws of motion apply.
- **ECEF**: Earth-Centered Earth-Fixed frame in which the axes are rotating with the earth.

The origin of a body frame is located in the center of mass of the body [33] such that the position of the body, expressed in body frame, is zero. Instead, one may represent the location of the body relative to an external origin expressed using body frame axes, resulting in a convenient representation of body frame position for quadcopter control. This representation of position, expressed in body frame axes, is used for the rest of this thesis and is called body frame position.

A figure displaying a quadcopter with body frame axes and orientations can be seen in 2.1, where

- altitude is body frame z-axis.
- x-axis is defined as forward for the quadcopter, and y-axis is to the left for the quadcopter.
- pitch, roll, and yaw are rotations about body frame y-, x- and z-axes respectively.

The body frame axes of some quadcopters, including the AR.Drone 2.0, are rotated 45 degrees in yaw compared to the body frame presented in Figure 2.1, such

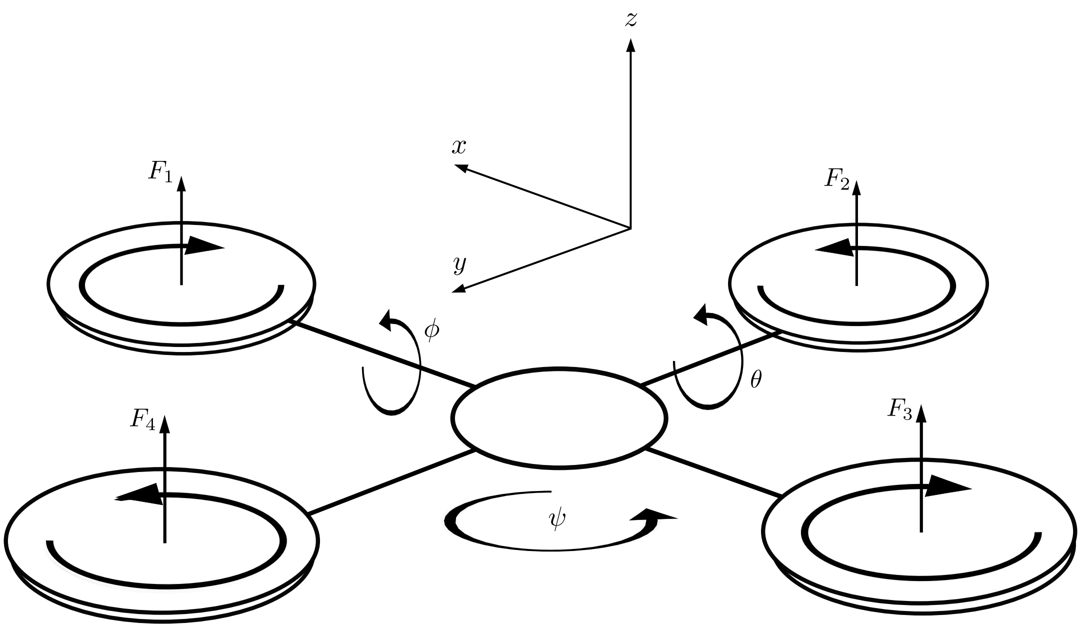


FIGURE 2.1: Quadcopter with body frame axes displayed.

that the x- and y-axes pass between two motors instead of crossing through the motors. The internal quadcopter control equations differ slightly with such a body frame. However, the internal controller of the Ar.Drone 2.0 is regarded as a black box in this thesis. Consequently, the navigation, pose estimation and control, which are the focus of this thesis, are not affected by such a change.

2.2 Kalman Filter

The Kalman Filter is an algorithm for producing an estimate of a state and the uncertainty of the estimate. It combines measurements over time, as well as prior knowledge about the system dynamics, in order to produce a statistically optimal estimate of the system state. Since it combines system dynamics with sensor measurements, it is more robust to sensor failure than a system that relies solely on sensor measurements for state estimation. It will often produce estimates at a higher frequency than the output frequency of the sensors because it extrapolates knowledge about system dynamics to predict the system state in between sensor measurements [34].

There exist multiple variations of Kalman Filters which are designed for different applications. The Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) are used for non-linear state estimation, and an Error-State Kalman Filter is used for estimation of states in which a dynamic model of the system is not available [35]. In this thesis, the standard (vanilla) Kalman Filter is used and is discussed further in the section below.

2.2.1 Vanilla Kalman Filter

A discrete time linear system can be represented by the state space model:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{2.1}$$
$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \tag{2.2}$$

where k ∈ ℕ and k > 0. The vector $\mathbf{y}_k$ is a measurement vector, which is the available sensor output in the system, and $\mathbf{x}_k$ is the state vector of the system. The vector $\mathbf{u}_k$ is a vector of control signals which are applied to the system. The dynamics of the system are described by the system matrices **A** and **B**, while **C** is a measurement matrix which represents the relationship between the state vector $\mathbf{x}_k$ and $\mathbf{y}_k$, apart from noise. The vectors $\mathbf{w}_k$ and $\mathbf{v}_k$ are process noise and measurement noise respectively. They are assumed to be white noise, meaning that they are normally distributed noise processes with zero-mean and stationary covariances **Q** and **R**, s.t. COV($\mathbf{w}_k$) = COV(**w**) = **Q**, and COV($\mathbf{v}_k$) = COV(**v**) = **R**, where:;

$$\mathbf{w}_k \sim N(\mathbf{0}, \mathbf{Q}) \tag{2.3}$$

$$\mathbf{v}_k \sim N(\mathbf{0}, \mathbf{R}) \tag{2.4}$$

The Kalman Filter system is producing an estimate $\hat{\mathbf{x}}$ which is a statistically optimal estimate of the true state **x**. It performs this estimate in two steps. The prediction step, also called the time update step, is performed by using knowledge about the discrete-time state-space system to propagate the current estimate forward in time. Both the current estimate $\hat{\mathbf{x}}_{k-1}$, and the current control signal $\mathbf{u}_{k-1}$ are inserted into Equation (2.1) in order to predict the state in the next time step. Thus, the update step is performed by:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \tag{2.5}$$

where $\hat{\mathbf{x}}_k^-$ is the a priori state estimate. The a priori state estimate is a state estimate which has not been updated by sensor measurement.

Measurement updates of the a priori state estimates are performed when sensor measurements are available. The difference between the sensor measurement $\mathbf{y}_k$ is and the measurement matrix, multiplied with the a priori state estimate $\hat{\mathbf{x}}_k^-$, is called the innovation and is used for updating the estimate. The innovation is multiplied with a Kalman Gain, $\mathbf{K}_k$, and added to the a priori state estimate in order to produce the a posteori state estimate $\hat{\mathbf{x}}_k$. Thus the update step of the Kalman Filter is

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{C}\hat{\mathbf{x}}_k^-) \tag{2.6}$$

A priori estimation error and a posteori estimation errors are defined as

$$\mathbf{e}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^- \tag{2.7}$$

$$\mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k \tag{2.8}$$

where a priori estimation error is estimation error before the estimation is updated using sensor measurement, and a posteori estimation error is estimation error after measurement update. Thus, these errors signify how far the estimate is from the true state.

The covariances of the a priori- and a posteori estimation error are therefore

$$\mathbf{P}_k^- = E[\mathbf{e}_k^-, (\mathbf{e}_k^-)^\top] \tag{2.9}$$

$$\mathbf{P}_k = E[\mathbf{e}_k, \mathbf{e}_k^\top] \tag{2.10}$$

where $\mathbf{P}_k^-, \mathbf{P}_k \in \mathbb{R}_{n\times n}$

In order for the Kalman Filter to be a statistically optimal state estimator, the update step of the Kalman Filter needs an optimal update rule, meaning that the Kalman Gain is defined such that the covariance of the a posteori estimation error is minimized. The Kalman Gain that satisfies these requirements is derived in theory

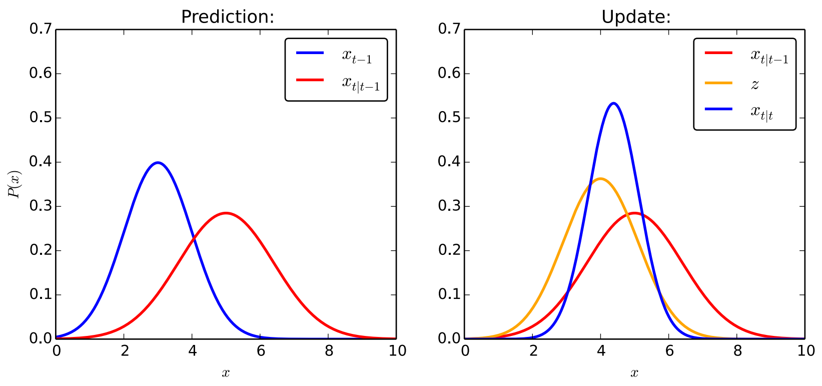


FIGURE 2.2: Kalman Filter update and prediction. Prediction increases variance, while updates reduce variance.

[35] to be

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{C}^\top (\mathbf{C}\mathbf{P}_k^- \mathbf{C}^\top + \mathbf{R})^{-1} \tag{2.11}$$

The estimation error covariances are predicted and updated as well. The prediction rule is

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^\top + \mathbf{Q} \tag{2.12}$$

and updated according to the update rule

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{C})\mathbf{P}_k^-. \tag{2.13}$$

A plot of the Kalman Filter update and prediction steps can be seen in Figure 2.2, where a one-dimensional state estimate x is plotted versus the and uncertainty of the estimate. A prediction may move the mean of the estimate and also increases the uncertainty of the estimate. An update may move the mean and, at the same time, decrease the uncertainty of the estimate.

In sum, the Kalman Filter is an optimal algorithm for combining knowledge about the system dynamics and sensor input to produce a state estimate. However, this requires an approximation of the system dynamics, which may not be trivial to produce.

2.3 Computer Vision

Autonomous robots and computer systems require a mapping of the environment they are supposed to operate. Good sensing- and perception systems are required for autonomous systems to operate in the real world. Computer Vision (CV) is a type of sensor system where cameras are used as sensors, and computer algorithms process the images in order to extract the desired information from the images.

DL-based computer vision systems (DLCV) have dominated the field of computer vision in recent years. However, traditional computer vision (TCV) systems are still favored for specific applications of computer vision. Both TCV and DLCV have unique strengths and weaknesses, and one should choose which one to use depending on the specific application.

2.3.1 Traditional Computer Vision

The field of digital image processing and computer vision has been dominated by mostly TCV systems. These systems have been used for vision systems for robotic applications, processing- and alterations of digital images, extracting information from images, and locating objects and features in images.

The TCV algorithm used in this thesis is developed by Sundvoll [1]. It is in this thesis used as part of a larger system without significant modifications. Therefore, this section does not go deep into the theory behind TCV, instead presenting an overview of the field.

TCV systems are primarily based on three steps. The first step is feature extraction, in which features of interest are extracted from the image. Features of interest are primarily corners and edges and other pixel points which are contrasting the surrounding pixels. These points of interest contain the most relevant information in the image. Thus, reducing an image to solely the points of interest is used to keep the most relevant structural information in an image while reducing the number of pixels required to represent this information. Several different algorithms are available for feature extraction, such as Canny edge detector [36] and Harris edge- and corner detector [37]. An illustration of edge detection using Canny edge detection can be seen in Figure 2.3.

The next step of TCV systems is feature description. Feature description is the process in which descriptors are assigned to each of the detected points of interest. These descriptors are a matrix of values representing the points of interest and the surrounding pixels. Such a descriptor is used to match similar points together, often for tracking objects across multiple image frames. Speeded-Up Robust Features (SURF) [38] is a much-used feature detector and descriptor algorithm, which is fast and robust, as well as being invariant to image translation and rotation.

The last step is defined as classification, where points of interest and descriptors are used for the application's intended purpose. Visual Simultaneous Localization and Mapping (VSLAM) [39] is a TCV algorithm where points of interest are matched between image frames in order to create a 3d-point cloud map of the environment, as well as tracking the camera's trajectory throughout the map. Stereo vision systems match feature points together in order to create a single view using multiple cameras. In order to detect specified objects in images, the descriptors are compared with descriptors of the objects being searched for.

2.3.2 Deep Learning-Based computer vision

Recent leaps in hardware processing technology, as well as much research and interest in the field of DL, has caused many new milestones within the field to be reached. The spread of DL-based technology has been increasing and therefore has also the interest in DL-based computer vision increased. DL-based computer vision systems are being used for facial recognition software [40], self-driving cars and other mobile robot systems [41], industry inspection and quality control [42], and numerous other applications.

DL is a branch of computer science based upon a network structure called Deep Neural Networks (DNNs). By training these deep networks with input data labelled with output values, the network can be trained to predict the desired output for the correct input data. Deep Neural Networks consists of many nodes, also called neurons, that are interconnected in a network. Each node contains one numerical value.

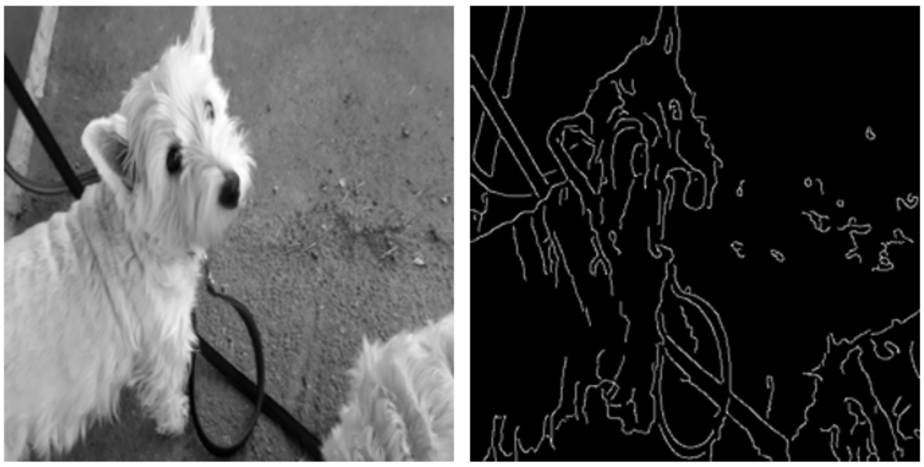


FIGURE 2.3: Canny edge detector algorithm applied to a picture of a dog. Color and fill information is removed, while edges are detected and kept.

Nodes are separated into a structure called layers, and these layers are organized in a serial pattern.

Fully Connected Neural Networks

In a Fully Connected Neural Network (FCNN), every node is connected to every node in the previous- and the following layers. A graphical structure of an FCNN can be seen in Figure 2.4. The first-, and leftmost layer of the network is the input layer. Input data is input as the values of the nodes in the first layer. Then the values are propagated through the network until it reaches the last layer, which is the output layer. Neuron values are propagated through the connections between the neurons. These connections are called weights and consist of a numerical value as well. The value of a neuron is propagated to the connected neuron in the next layer by multiplying the neuron value by the weight value. Thus, the value of a neuron in an FCNN will be a weighted sum of the previous layer neuron values. Additionally, an individual bias is added to every neuron.

The value of neuron j in layer k is defined as:

$$z_j^k = \sum_{i=0}^{n} w_{ij}^k a_i^{k-1} + b_j^k \tag{2.14}$$

$$a_j^k = f(z_j^k) \tag{2.15}$$

where $w_{ij}^k \in \mathbb{R}$ is the weight connecting neuron i in layer $k-1$ and neuron j in layer k. $a_i^{k-1} \in \mathbb{R}$ is the value of neuron i in layer $k-1$, and $b_j^k \in \mathbb{R}$ is the individual bias for neuron j in layer k. $f : \mathbb{R} \to \mathbb{R}$ is an activation function which is used to get neuron values contained within a desired range. These weights between the nodes are what maps the network into to the output. Changing these weights will result in a different output for the same input. The network model parameters is denoted by $\boldsymbol{\theta}$, which is a matrix that contains all the weights and biases in the network, such

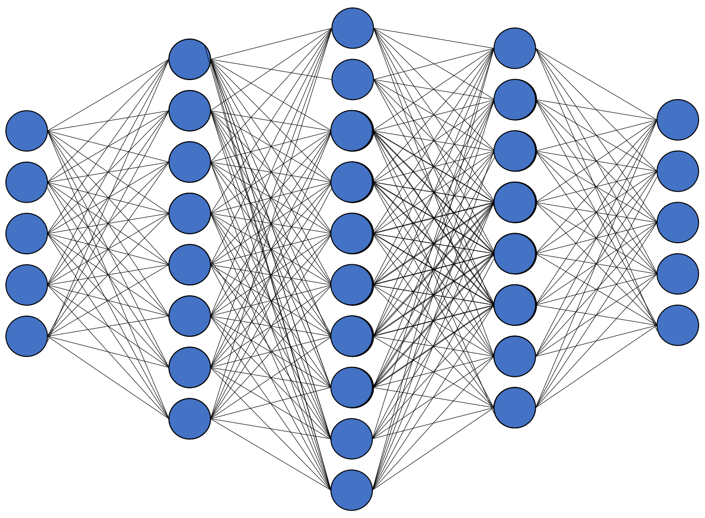


FIGURE 2.4: Graphical structure of a small FCNN. Every node is connected to every node in the previous- and next layer. Input data is applied in the leftmost layer, and then propagated through the network, to the last layer, which is the output values of the network.

that $\boldsymbol{\theta} = [\mathbf{w}^0, \mathbf{w}^1, ..., \mathbf{w}^n, \mathbf{b}]$, where $\mathbf{w}^k$ is the weight matrix for layer $k = 0, 1, ..., n$ and $\mathbf{b}$ is the network biases.

The output of the network is the resulting values in the last- and rightmost layer.

The number of neurons in an FCNN can span from a couple thousand in small networks to many million in large networks. The more complex the analysed data structures, the more neurons are required in the network.

Training Neural Networks

In order for a DNN to be able to produce the desired output for a given input it has to be trained. In supervised machine learning DNNs are trained by being presented with input data, and a corresponding desired output, called labeled data. The labels are what the trainer desires the DNN to be able to predict for the given data. When training begins the training data is propagated through input through the network layers. Thus the network produces output values for the given input values. This is called a forward pass. This output from the network is compared to the desired output for the corresponding input data. An error metric is calculated, called loss, which signifies how accurate the network was in predicting the desired output value. The learning algorithm seeks to minimize loss. In order to do this it modifies the weights and biases in the network in such a way that loss is reduced for the given data. This loss is also called cost, and is denoted by C.

In order to minimize loss the training algorithm has to calculate the partial derivative of loss given input data, with respect to the model parameters. Then, using a update rule, the network updates the current model parameters, and continues training using the new model parameters. One such update rule is called gradient descent minimization, and is commonly used. With gradient descent update rule the model parameters is updated by

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \alpha \frac{\delta C(\boldsymbol{\theta}, \mathbf{z}^k)}{\delta \boldsymbol{\theta}}$$

where $\mathbf{z}^k \in \mathbb{R}$ is a vector of the output of the last layer in the network and $\alpha \in \mathbb{R}$ is predefined learning rate which specifies how fast the network is to converge. The partial derivative $\frac{\delta C(\boldsymbol{\theta}, z^k)}{\delta \boldsymbol{\theta}}$ is numerically difficult to calculate, and is instead approximated using a technique called backpropagation.

This training is performed over a large number of iterations, and loss is gradually decreased over time. After training, the network is tested on another labeled data set, called a test set, which has not been used for training. Using this test set, the final accuracy of the trained model is determined.

For a DNN to achieve a high level of accuracy it requires a dataset which is of high quality. The quality of a dataset is dependent on the number of data samples, the precision of the labels, whether the labeled data is varied and whether the data resembles the data which the final model is to be used on. Data augmentation is a technique for augmenting and improving the quality of the dataset. Data augmentation is performed by duplicating parts of the data in order to increase the size of the data set, and subjecting the data to modifications and distortions in order to increase the variety of the data set, and to make the data more robust to changes in input data.

Convolutional Neural Networks

When working with DNNs, an FCNN is not well suited to capture information in images. Images are coded as a matrix of pixel values, where the location of the pixels in relation to the others is an essential part of the information. In an FCNN, the neurons in the first hidden layer are connected to every pixel in the input image. A one-pixel translation of the pixels in an image will not alter the image information much. However, it will significantly change how the image is propagated through an FCNN because the pixels will be passed through different weights. Therefore an FCNN does not capture all relative spatial information in images.

A Convolutional Neural Network (CNN) provides a solution to this and is, therefore, the preferred choice working with image data. A CNN consists of three different types of layers; convolutional layers, pooling layers, and fully connected layers.

Convolutional layers are layers where data, represented as matrices, is subjected to convolutional mappings. A convolutional mapping is an operation where data values are combined with other adjacent values and passed on to the next layer. The values are passed on to the next layer by taking the dot product of a moving grid in the value matrix with a trained matrix of weights called a kernel. An example of a convolution operation using an example data matrix A and kernel K can be seen in Figure 2.5.

Pooling layers are a type of layers in a CNN used to reduce the dimension of the value matrices without losing important data. A maxPool is the most commonly used pooling technique, and a 2x2 maxPool layer will propagate the highest value in a 2x2 grid onto the next layer, thus halving the matrix dimensions. An example of a maxPool can be seen in Figure 2.6 where a 4x4 matrix is reduced to 2x2 using a 2x2 maxPool. Other pooling techniques are also used, such as minPool and average-Pooling.

The structure of a CNN usually consists of a series of pairs of one convolutional layer and one pooling layer. After a series of convolutional- and pooling pairs, CNNs contain one or more fully connected layers.

Bounding box (BB) classification algorithms are based on the CNN structure. BB-classification algorithms detect whether certain classes are present in an input image while also predicting the location of these classes. An example of an output image

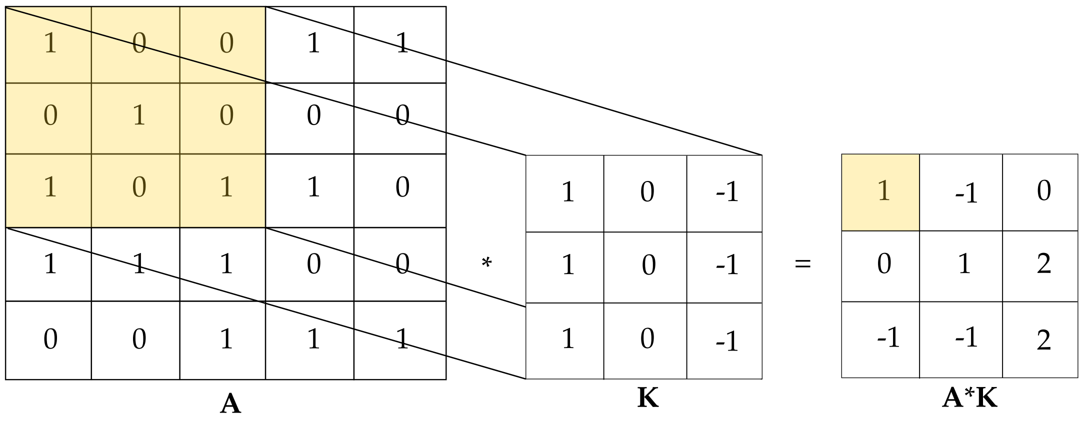


FIGURE 2.5: An example of a convolution operation, with a 3x3 convolution filter K

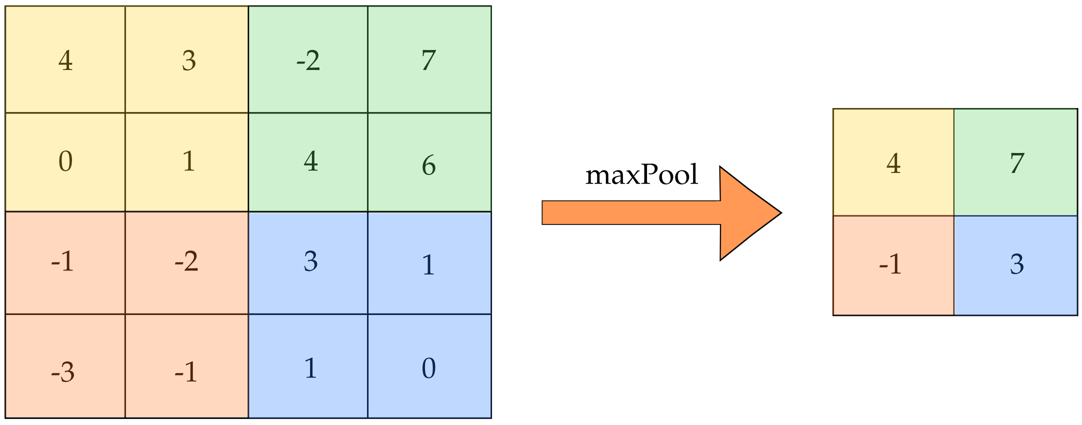


FIGURE 2.6: 2x2 MaxPool function, where the dimensions of a matrix are reduced by stripping away non-max values within a grid. Here a 2x2 maxPool function is applied to a 4x4 matrix, and the resulting matrix consists of the largest values within the respective colored grids.

from a bounding box prediction algorithm is seen in Figure 2.7, which is output from a YOLO bounding box prediction algorithm. BB-prediction algorithms are used for surveillance applications, item detection and localization and pose estimation for autonomous control. There are multiple different kinds of bounding box prediction algorithms including YOLOv3 [43] and YOLOv4 [18], Faster R-CNN [44], EfficientDet [45].

Other computer vision techniques that use a CNN structure are image segmentation [46] and direct methods. Direct methods use an end-to-end DNN from input image to desired output, which is being developed for autonomous automobile control [47] [48].

YOLO

You Only Look Once (YOLO) [43] is a real-time bounding box (BB) prediction algorithm created by Redmon et al. It is a fast algorithm compared to comparable prediction algorithms while at the same time achieving a high mean average precision (mAP). The inference times and corresponding mAP of YOLOv3 and YOLOv4 compared to other BB-prediction algorithms can be seen in Figure 2.8. Because YOLO

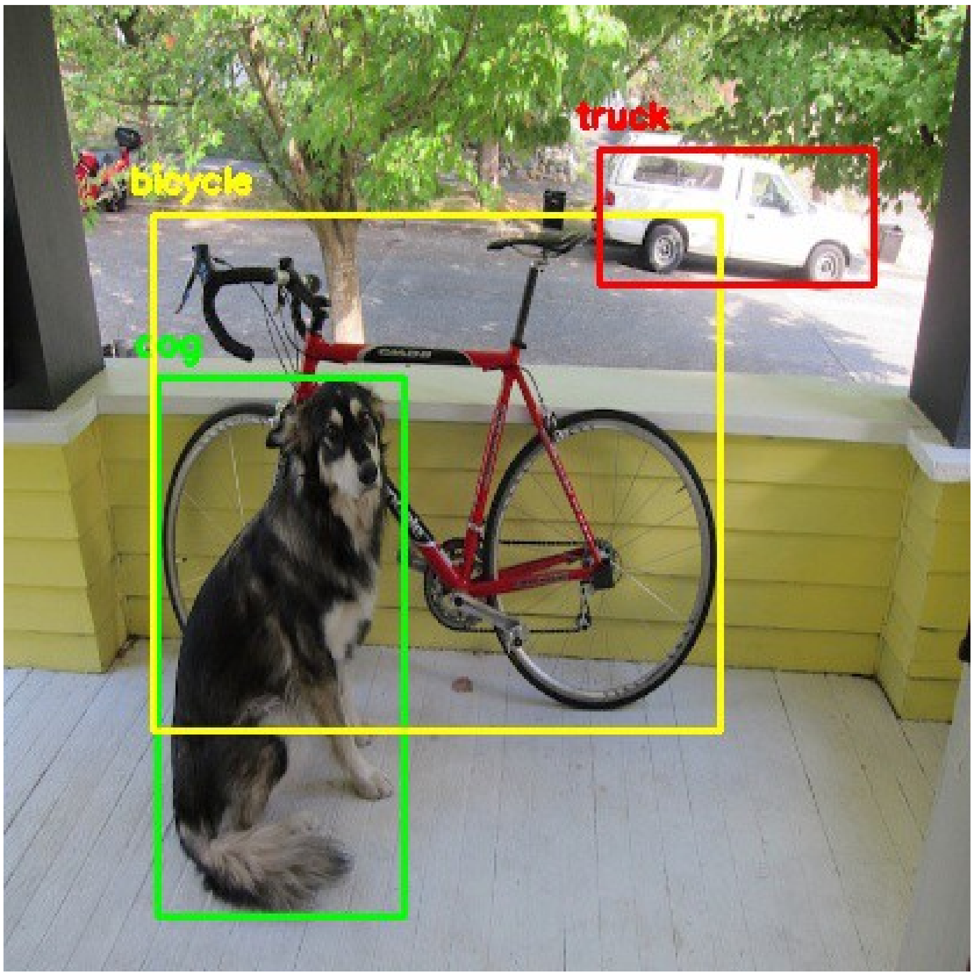


FIGURE 2.7: Image with bounding boxes predicted by YOLO. Image collected from [49]. Three classes are identified; dog, bicycle and truck, and bounding boxes are drawn around the predicted location of the classes.

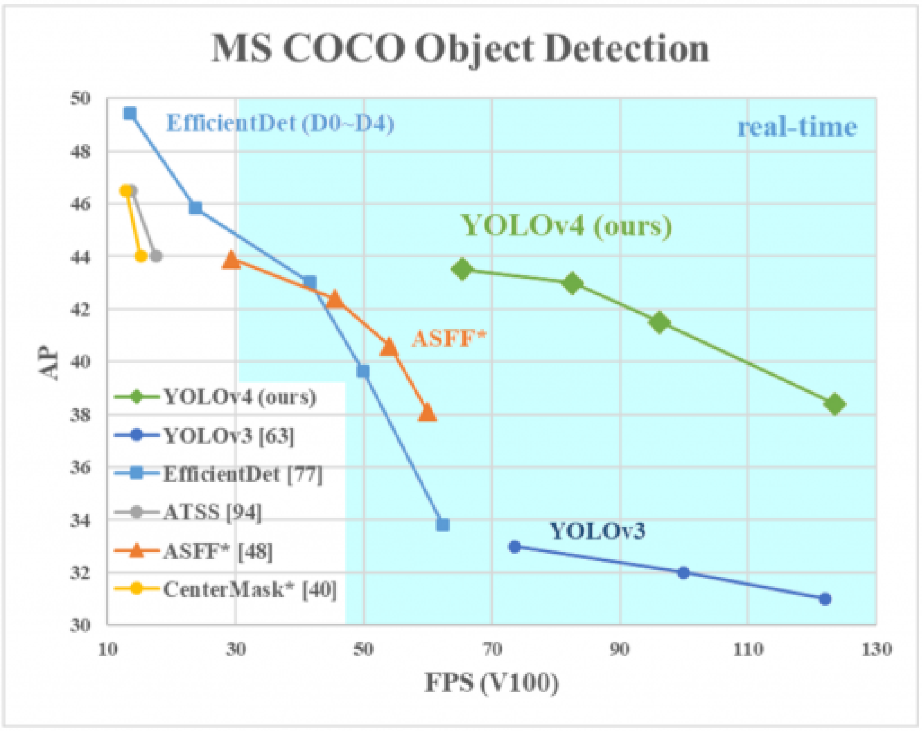


FIGURE 2.8: Inference times and corresponding mean-average-precision compared between different bounding box detection algorithms, while being tested on the MS COCO dataset [50]. Image collected from [18]

can achieve both a high inference rate and a high mAP, it is one of the most used real-time bounding box prediction algorithms.

There exists multiple versions of YOLO, as well as multiple iterations of these models. YOLOv4 is the most recent update to YOLO, although an unofficial YOLOv5 has been released by Glenn Jocher [51]. YOLO-tiny [52] is a scaled-down version of the YOLO network, which has a higher inference rate at the cost of a lower mAP. Scaled YOLO [53] is a YOLO version that can be scaled up and down, depending on preferences between high mAP or high inference rate.

The YOLO algorithms are known to have some weaknesses. YOLO is known to struggle with the detection of small objects [20]. Another weakness of the YOLO algorithms is that bounding boxes tend to have both small translational shifts as well as minor changes in bounding box aspect ratio between frames making the boxes seem to 'jitter' [54].

The output from YOLO is a list of bounding boxes that are detected in a given image frame. If no objects are detected in the image frame, then an empty list is output. A bounding box is a data type that consists of:

- coordinates: xmin, ymin, xmax, ymax
- bounding box probability
- object class

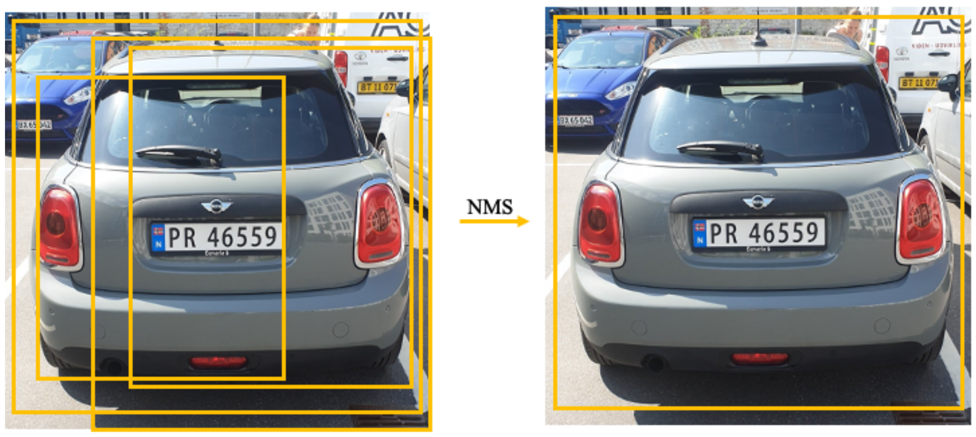


FIGURE 2.9: NMS removes overlapping bounding boxes of the same class

where xmin, ymin, xmax, ymax defines the bounding box location in the camera frame. Probability is a confidence score for the algorithm, which signifies how confident the algorithm is in the prediction. Class is the type of object that the algorithm predicts the object to be.

YOLO can achieve a high inference rate because it performs *one* inference on an image in order to predict both bounding box location as well as the class type and confidence score, hence the name *You Only Look Once*. YOLO performs a forward pass through the network by dividing the picture into a grid of S x S cells, where S is predetermined. Every cell in the grid will predict a number of bounding boxes, between 3 and 5, centered in the cell. Each bounding box will find an object and encapsulate it. These bounding boxes have different pre-set aspect ratios in order to detect objects with different shapes.

YOLO predicts objects of different scales by performing forward pass multiple times in parallel at different scales, thus using different numbers for S. A large number for S will result in the image being divided into a large number of cells. These cells will then predict bounding boxes, but A confidence score is predicted for every bounding box prediction.

Every frame spawns a large number of bounding box predictions, as every cell produces multiple bounding boxes. In order to reduce this to the relevant matches, YOLO performs two filtering techniques. A confidence threshold is applied to all the bounding boxes, which removes all bounding boxes with low confidence scores. This threshold is a user-tunable number between zero and one. YOLO also performs Non-Max-Suppression (NMS), which removes duplicate bounding boxes for the same object in an image. It removes bounding boxes located in an area occupied by a bounding box of the same class and a higher confidence score. An example of NMS is shown in Figure 2.9 where NMS removes overlapping bounding boxes of the same class.

YOLO is based on a CNN structure called Darknet [43] by Redmon et al. YOLOv3 is based on a is a CNN backbone structure called Darknet-53 with 53 layers. The different YOLO versions are modifications of the same Darknet structure, where the Tiny-YOLO versions have fewer- and smaller layers to increase inference frequency, and the scaled-up versions have more- and larger layers to achieve a higher mAP. YOLOv5 is not based on a Darknet structure.

2.3.3 Comparing DLCV and TCV methods

DL-based computer vision techniques (DLCV) and traditional computer vision (TCV) techniques are different in implementation and use. DLCV has in recent years pushed the limits for what is possible to do with computer vision. However, Walsh et al. argue that DL has not made TCV techniques obsolete [55].

A DLCV system is created by training a DL model with a dataset of images. Such a training dataset is required to be sufficiently large and of high enough quality for the model to achieve the desired level of accuracy. Such a dataset may not be available and may be costly or difficult to generate. In recent years public datasets such as Microsoft COCO [50] have made it possible to pre-train the models on public datasets and subsequently performing transfer learning by continuing training on a specific dataset. Data augmentation techniques[56] are techniques for making datasets larger and improving the quality of the datasets. Such methods have made it easier and less expensive to generate proper datasets for DLCV. TCV methods do not require datasets for training and may therefore be the method of choice if no dataset is available.

Training DLCV models does not require expert knowledge about the specific computer vision problem, rather general knowledge about training DLCV models. Therefore such implementations can be simple to implement and improve, as long as proper datasets are available. TCV methods, however, do require expert knowledge about the specific computer vision problem. Implementing such solutions requires that the creator is skilled in extracting the useful information from the images and processing that knowledge into usable output, which may be time-consuming if the CV problem is complex.

DLCV methods can not be better than the dataset it is trained on, as it is not able to learn to predict the output better than what it has been trained to do. It can also be challenging to perform iterative improvements of DLCV models, as the requirements for improving such models are improved datasets or further training with changes in model parameters. Due to the black-box nature of DL methods, it is not trivial to know what backbone changes result in improved models. Thus improvements may require training a number of different models, and then to choose the best one. Training DL models is time consuming and requires large computing powers. When a TCV method is created it may be easier tweak the final model than it is to tweak a DL model. A TCV model does not require a new training period, nor a new dataset, meaning that iterative improvement may be easier.

DLCV methods, when run on a Graphical Processing Unit (GPU), are able to process images very quickly, with achieving inference rates in the hundreds of hertz. This may be possible on simple TCV tasks as well, however TCV is generally slower for complex CV tasks.

TCV may be quite easy to implement and sufficient for simple CV tasks, but may prove insufficient when applied to more complex tasks, such as facial recognition and product inspection. TCV still dominates in areas such as visual Simultaneous Localization and Mapping (vSLAM) [39]. DLCV methods excel is extracting information from complex data structures, and classifying them. DLCV solutions such as TrackNet [57] are able to perform object tracking by performing object tracking over time by using input from multiple image frames at the same time. Implementing such solutions using TCV methods would be extremely complex.

DL solutions are generally regarded black box solutions, meaning that the inner processes of the solutions are either unknown or not understood. Thus, it is viewed as a *black box* in which one inputs data and receives an output while not knowing how the solution produced that output. Consequently, one cannot guarantee how the solution will perform in the future, which may cause a safety verification problem. Solutions to the black box problem of AI are being developed, such as explainable AI solutions [58] which may be imperative for wide-scale adoption of AI solutions. TCV solutions are not black box solutions. They are programmed explicitly, and the output is therefore predictable and can be understood from the given input.

2.4 Inertial Measurement Unit

An inertial measurement unit (IMU) is a sensor that measures specific force- as well as angular velocity experienced by the sensor [59]. When placed on a body, it will measure the specific force and the angular velocity of the body, represented in the body frame.

An IMU measures the specific force using a 3-axis accelerometer, which measures acceleration and measures angular velocity using a 3-axis gyroscope. Together, these sensors compose a 6-DoF IMU. There also exists IMUs that utilize a magnetometer in order to measure the orientation of the body.

The measured specific force on the body$\tilde{\mathbf{f}}^b$ and the measured angular velocity on the body $\tilde{\boldsymbol{\omega}}^b$ are given by the equations:

$$\tilde{\mathbf{f}}^b = \mathbf{f}^b + \mathbf{b}_f^b + \mathbf{n}_f \tag{2.16}$$

$$\tilde{\boldsymbol{\omega}}^b = \boldsymbol{\omega}^b + \mathbf{b}_\omega^b + \mathbf{n}_\omega \tag{2.17}$$

where $\tilde{\mathbf{f}}^b$ is measured specific force in the body frame, $\mathbf{f}^b$ is true specific force experienced by the body and $\mathbf{b}_f^b$ and $\mathbf{n}_f$ are sensor bias and noise for accelerometer measurements. Likewise for the gyroscope measurements, $\tilde{\boldsymbol{\omega}}^b$ is measured angular velocity, $\boldsymbol{\omega}^b$ is true angular velocity experienced by the object and $\mathbf{b}_\omega^b$ and$\mathbf{n}_\omega$ are gyroscope bias and noise.

IMUs generally have a high sampling rate. Consumer-grade IMUs generally have a sampling rate in the range of 100-1000Hz, while higher-grade IMUs may have sampling rates of 10kHz and more.

Consumer-grade IMUs generally have significant measurement noise and biases. Due to these measurement errors in the system, an inertial navigation system based upon the integration of IMU measurements may become significantly inaccurate within seconds. Such a system will therefore require frequent corrections using other sensors. Despite this, such an INS system may be a useful component in combination with other sensors.

2.5 Finite State Machine

A Finite State Machine (FSM) is a programming technique for developing programs that are to perform different actions at different points in time [60]. An FSM is defined by a set of finite pre-defined and discrete states. While running, the program will at any time be in one of the states and can change from one state to another in response to certain conditions being met or in response to some inputs.

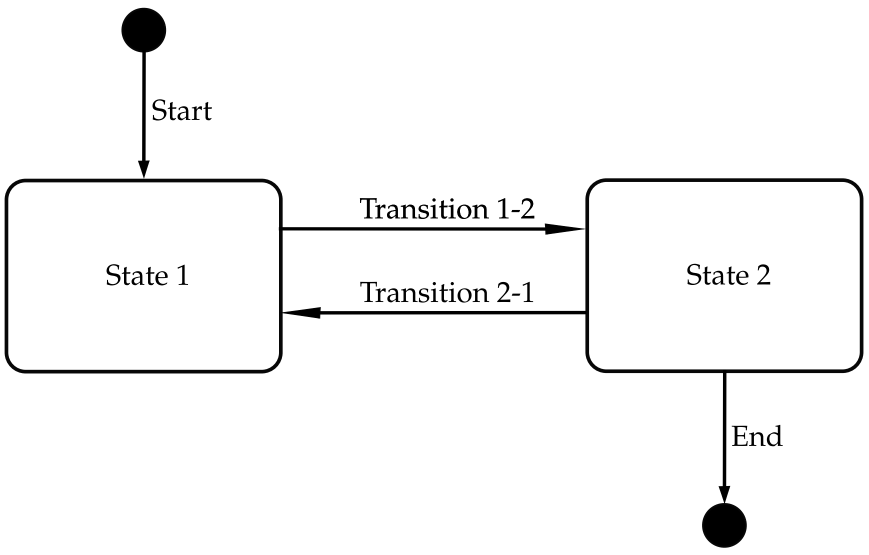


FIGURE 2.10: A two sate state-machine

While in a state, an FSM will perform the specified functionality for that state while also checking for conditions that should trigger a state change. Each state can transition to some other states, and each state can have different conditions for transitioning.

The states and transitions of an FSM can be represented as a graph, where the states are nodes in the graph, and the transitions between the states are the edges of the graph. A graphical representation of an example state machine consisting of two states is displayed in Figure 2.10. The black circles represent the start and end state. The controller for an object like a toaster could be implemented with such a two-state FSM, where the states are *IDLE* and *TOASTING*, and transitions occurring when a button is pressed and when a toasting timer is finished.

FSMs are used in robotics development, control systems and embedded systems and can also be used for control of systems that are to perform a number of actions sequentially. Desired actions are then represented as separate states, and switching to the next state is triggered by conditions signifying that the current action is completed.

Chapter 3

Experimental Setup and Environment

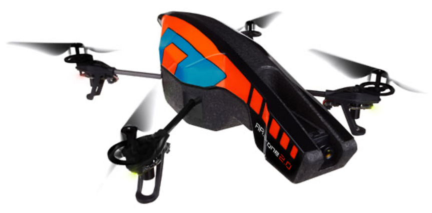
3.1 Parrot AR Drone 2.0

The Parrot AR.Drone 2.0 is a quadcopter produced by the French company Parrot in 2010. The AR.Drone 2.0 is a low-cost commercially available quadcopter.

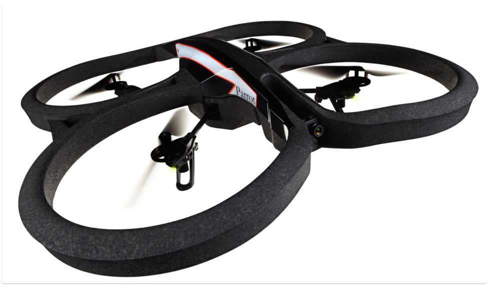
An indoor hull is available for the quadcopter, which provides a protective shield around the propellers. This hull makes the drone capable of surviving small bumps and crashes into walls and furniture, thus facilitating iterative testing during development. Other quadcopters are known to be less resilient to crashes, breaking upon impact. The indoor hull increases drag and makes the quadcopter less resilient to wind disturbance. Wind and high-speed flight where drag is significant is usually not a factor indoors, and therefore the indoor hull is used when performing indoor experiments. For outdoors use, the force of wind is significant and therefore the indoor hull can be replaced with an outdoor hull, which does not have the protective shields. The drone weighs 420g with the protective hull and 380g with the outdoor hull. The drone is 59x59cm equipped with the indoor hull. A figure showing the Parrot AR.Drone 2.0 with both indoor- and outdoor hulls can be seen in Figure 3.1.

The battery capacity of the AR.Drone is quite limited, enabling about twelve minutes of flight time on a full charge. Twelve minutes of flight time is sufficient for the experiments performed in this thesis.

The technology behind the AR.Drone 2.0 is not up-to-date with the current state-of-the-art quadcopters on the market. Quadcopter technology has advanced in the recent decade, and it may therefore be argued that development and research using this drone is less fruitful than using more recent and updated drones. Naturally, newer drones on the market are equipped with better sensors, have more stable on-board control algorithms, and faster- and more accurate communication systems.



(A) Outdoor hull



(B) Indoor hull

FIGURE 3.1: The Parrot Ar.Drone 2.0 with outdoor- and indoor hulls.

Although the AR.Drone is lacking in these aspects, it does provide the following advantages for research and development of quadcopter systems:

- There exist open-source drivers for communication with- and control of the AR.Drone
- There exists open-source computer simulation systems featuring the AR.Drone, facilitating for extensive testing of the quadcopter system.
- It is cheap and easy to replace if it breaks during testing.
- It features a protective foam shield in order to be resilient to collisions.
- Several studies have been performed using the AR.Drone, and therefore it is simple to perform a comparative analysis of the performance of the quadcopter system.

Consequently, the AR.Drone 2.0 was selected for the development of the quadcopter system in this thesis. Future research and further iterations of this system should be carried out using up-to-date quadcopter systems.

There is a significant delay in communication with the Parrot AR.Drone 2.0. This communication delay is tested and discussed by Engel. et al. [17], where they estimate the delay from control signals are sent from the control device and until the AR.Drone starts performing the control actions to be 60ms. Engel et al. also estimate there is a 130ms delay from sensor information is received by the AR.Drone and until it is received by the control device. Adding control device processing time, the total delay in a feedback control loop may surpass 200ms.

3.1.1 Sensors

The AR.Drone is equipped with multiple sensors for state estimation. These include a barometric pressure sensor for the estimation of altitude. The drone does not translate the barinetric data into altitude, rather sending raw pressure readings in Pascal.

Furthermore, the AR.Drone 2.0 is equipped with two cameras, a front-facing camera and a bottom facing camera. The front-facing camera is able to output video at two user-selectable levels of resolution: 720p (1280x720) and 360p (640x360) at 30fps. The bottom-facing camera of the AR.Drone captures video at a resolution of 240p (320x240) at 60 fps. This resolution is then scaled up so that the resolution of the output video is 360p. Both cameras are rolling shutter, meaning that the images may be severely distorted when the quadcopter is moving [61].Thus, there is a significant amount of motion blur from the bottom facing camera when the quadcopter pitches or rolls.

The AR.Drone 2.0 also features a 200Hz IMU which consists of a 3-axis accelerometer as well as a 3-axis gyroscope. It also contains a 3-DOF magnetometer with an accuracy of 6 degrees, as reported in the documentation [62]. These components together make the drone good at providing pitch and roll estimates. These estimates do not seem to drift over time, making the quadcopter able to stay controlled in the air for a substantial amount of time. Engel et al. report that that the AR.Drone onboard estimates of yaw of the were subject to drift at a rate of about 60 degrees per minute.

It is possible to mount an external GPS sensor, called a Flight Recorder, on the AR.Drone 2.0. A flight recorder is not available for the experiments in this thesis. It is, however, available in simulations.

3.1.2 Communication

The drone can be user-controlled by smartphone using the official Parrot AR.Drone app. Computer control is possible by using a ROS driver called ardrone_autonomy by AutonomyLab [63].

The quadcopter communicates by setting up an ad-hoc Wi-Fi network to which the control device connects. Communication is then performed by the quadcopter and control device sending messages over this Wi-Fi network. This communication is streamlined over four different channels. These are called:

- Navdata channel
- Video Feed channel
- Command channel
- Control channel

The Navdata channel is a channel where the quadcopter sends all navigation data, including battery percentage, barometer and IMU sensor readings, estimates of quadcopter orientation and current internal quadcopter state. Output from the quadcopter cameras is sent over the Video Feed channels. This is restricted to video output from only one of the cameras simultaneously. The command channel is for the control device to specify flight commands. The control channel is for changing quadcopter settings, such as toggling which camera provides the video feed.

3.1.3 Control

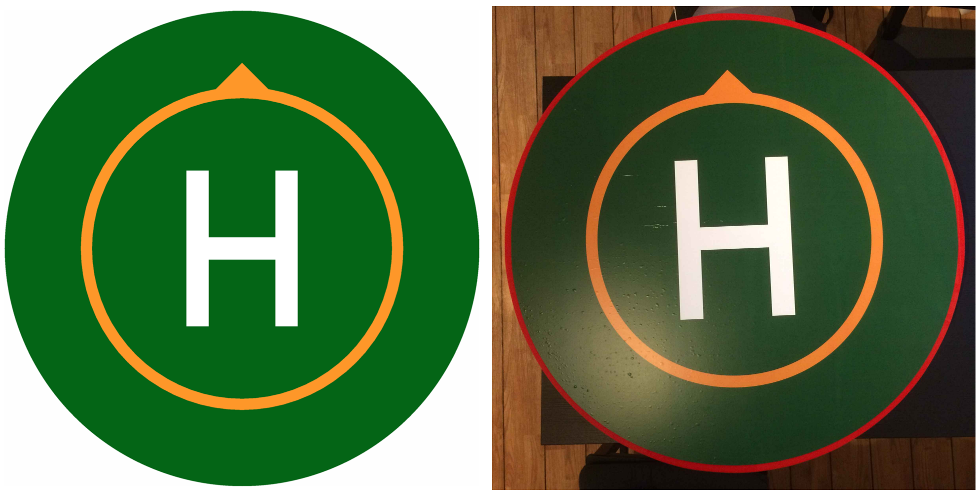
The Parrot AR.Drone 2.0 is equipped with an on-board low-level flight controller. This controller uses the IMU and gyroscope to control the drone in order to execute external flight commands. Flight commands are sent as desired velocities in the three linear directions and desired velocity in yaw. For control, only higher-level velocity input is required. The quadcopter can execute these commands by having an on-board estimation of attitude using the IMU and the magnetometer [61]. The on-board control system is not open-source and is therefore treated in this thesis as a black box system.

The AR.Drone will attempt to hover stationary in the air if it loses connection with the control device or does not receive any velocity commands.

3.2 Landing platform

The experiments of this thesis involve the Parrot AR.Drone 2.0 taking off from-, hovering over-, and landing on a specified landing platform. This landing platform was designed and constructed by Sundvoll [1]. The landing platform is a green circular platform. It features a large capital H, which is easily recognizable and typical for helicopter landing platforms. The landing platform also features an orange circle outside of the H and an orange arrow to specify the orientation of the landing platform.

The features of the landing platform are designed to be distinct and easily detectable for a computer vision system. The colors of the landing platform are contrasting colors, with the orange circle and -arrow contrasting the green background. In addition, the white H is in contrast to the green background, thus making the



(A) Computer model (B) Physical version

FIGURE 3.2: The quadcopter landing platform

features distinct and easy to recognize from above, making it easier for a computer vision algorithm to locate the landing platform in the image.

One might argue that an even more distinct landing platform design could have been chosen. For example, such a design could include additional larger features or blinking lights to more easily detect the platform for a computer vision system. While this is true, the design of this landing platform is meant to be visually pleasing to human beings while at the same time being detectable for a computer vision system. Moreover, a design that mimics the design of traditional helicopter landing platforms makes it evident to the uninformed observer that the platform is a landing platform. Ultimately, the design of the helipad is proven to be sufficiently distinct for computer vision systems to detect it [1] [2], while at the same time having a visually pleasing design.

The physical landing platform has a radius of 40cm, sufficient for hosting the AR.Drone 2.0. The white H is located in the center of the platform and spans 1/3 of the platform radius on the long side of the H and 1/4 of the platform on the short side. The orange circle, where the arrow is located, has a radius of 25 cm. A computer model of the landing platform can be seen in Figure 3.2a, and the physical landing platform can be seen in Figure 3.2b.

3.3 Robot Operating System

The Robot Operating System (ROS) is an operating system for programming robots. It is free and open-source, and is created by Open Robotics. It is extensively used in robotics development [64].

ROS, rather than offering a graphical user interface, offers a well-structured communication structure for development.

ROS is designed for modular implementation of software. This means that the functionality of the program is split into separate independent modules. Each module should contain everything that is required in order to execute one part of the system functionality.

ROS modules are called nodes. Nodes communicate by sending messages and receiving messages from other nodes. Such messages are sent over ROS communication channels called topics. Nodes may publish information to topics, and they

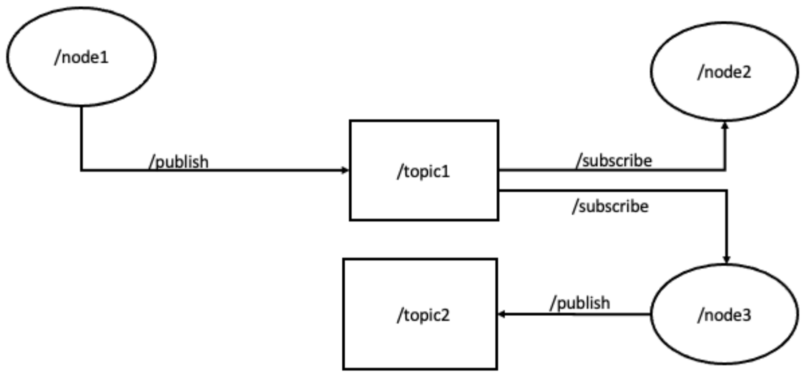


FIGURE 3.3: An example of a ROS structural network, with nodes and topics, publishers and subscribers.

may subscribe to topics, thereby receiving information that has been published to those topics.

It is possible to visualize the data flow in a ROS program as a communication graph of a ROS program, using arrows between nodes and topics to show data flow direction. An example of a ROS communication graph can be seen in Figure 3.3 where nodes are represented as circles and topics are squares.

Control of hardware is possible through ROS by implementing hardware drivers as ROS nodes. These ROS nodes may subscribe to topics for control signals and publish sensor messages to topics for sensor data.

There are several advantages with ROS-based application development. The modular structure makes iterative development and testing simple. ROS is open-source, and there is much available open-source code and hardware drivers available for use. Due to the modular nature of development with ROS, it encourages re-use and sharing of code. ROS offers development in several languages, but ROS messages are standardized between languages, meaning that different modules in the same application can be written in different languages. Another advantage of ROS is that due to the modular nature of the program, one can easily change sensor modules for data reading modules. Therefore, the program can be tested on dummy data just as easily as being tested with real sensor data.

Nodes in ROS are launched using the command line in terminal. It is possible to create .launch files in ROS, which are files written in .xlm, which contain information about launching ROS nodes. Therefore, one can write one .launch file that will launch a large number of ROS nodes using one terminal command. Creating separate .launch files for separate experiments and settings can run different experiments and tests simply and efficiently.

3.4 Gazebo Simulation Environment

Some of the quadcopter system performance tests are carried out in in a simulation environment called Gazebo, using Gazebo version 7.0.0. Gazebo is a computer simulation system for robotics development. It is developed by Open Robotics, and

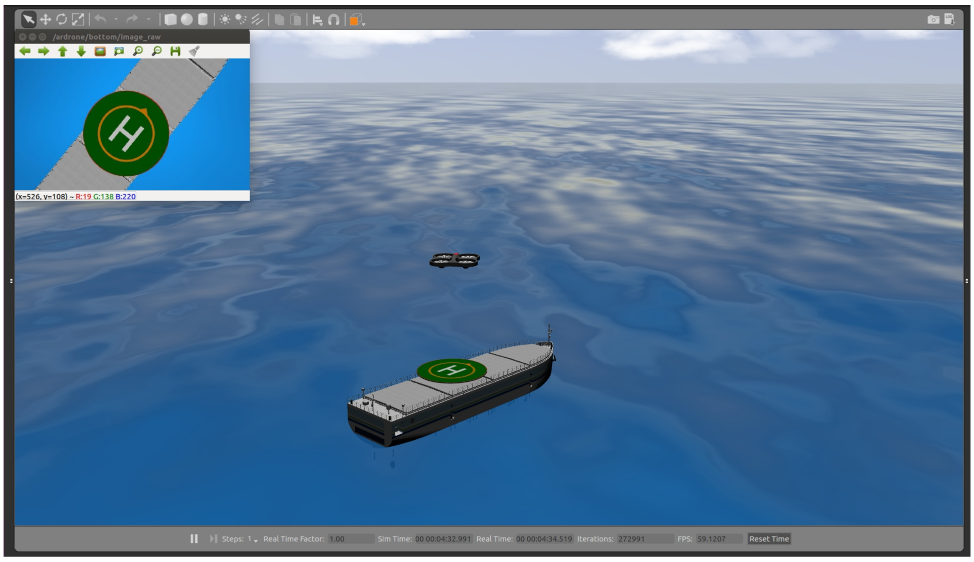


FIGURE 3.4: The Parrot AR.Drone 2.0, landing platform and ReVolt-ship in the Gazebo simulation environment. The camera view from the bottom facing camera can be seen in the top left of the simulation.

is open-source. Gazebo provides a robust physics engine as well as high-quality graphics and is integrated for use with ROS.

A simulation of the Parrot AR.Drone 2.0 is available in the ROS package *tum_simulator*, for use Gazebo. It approximates the physical response of the real AR.Drone 2.0, and features the same cameras and IMU as the physical quadcopter.

A 3D model of the landing platform, created by Sundvoll [1] is located upon a computer model of the ReVolt marine vessel, provided by DNV. Sundvoll combined these computer models with a simulated ocean model in order to create a marine simulation environment. An image of the simulation environment can be seen in Figure 3.4. The drone's bottom facing camera view is displayed in the top left of the simulation.

Chapter 4

Methodology

This chapter proposes a system for autonomous quadcopter control, including perception, filtering, mission navigation and control. The system aims to meet the objective of the thesis, to work towards an autonomous quadcopter system capable of performing simple missions without human intervention.

4.1 Defining coordinate Frames

The quadcopter system is to navigate relative to the stationary landing platform. Two coordinate frames are introduced for this task, one being the quadcopter body frame, which is defined as presented in Section 2.1.1. This coordinate frame is called body frame for the rest of this thesis.

The second coordinate frame has origin in the center of the landing platform, with axes fixed to the elements of the platform. Thus, this is a body frame for the landing platform. However, since the landing platform is stationary, this landing platform body frame is called the world frame in this thesis.

The world frame is defined by:

- (0,0,0) being located in the center of the landing platform.
- **z**-axis: is pointing directly upwards
- **y**-axis: is pointing opposite of the orange arrow on the landing platform
- **x-axis**: is perpendicular to both z-axis and y-axis.

Quadcopter body frame yaw rotation is defined as zero when the orange arrow on the landing platform is pointing to the right in the camera view.

A rotation matrix $\mathbf{R}_w^b(\phi, \theta, \psi)$ is used for transforming world frame coordinates to body frame coordinates, such that

$$\mathbf{x}^b = \mathbf{R}_w^b(\phi, \theta, \psi)\mathbf{x}^w \tag{4.1}$$

where

$$\mathbf{R}_w^b(\phi, \theta, \psi) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \tag{4.2}$$

where $\mathbf{R}_z(\psi)$, $\mathbf{R}_y(\theta)$, $\mathbf{R}_x(\phi)$ are rotation matrices for rotations about the axis' z, y and x respectively.

The world-to-body coordinate transformation can be approximated to rotation about the z-axis yaw, assuming that the pitch and roll movements of the quadcopter are small. This approximation is possible because of small angle approximation, where for any small angle α it follows that

$$\alpha \approx 0 \Rightarrow \begin{cases} \sin\alpha \approx 0 \\ \cos\alpha \approx 1 \end{cases} \tag{4.3}$$

Utilizing this, the rotation matrix is approximated to

$$\mathbf{R}_w^b(\phi, \theta, \psi) \approx \mathbf{R}_z(\psi) \tag{4.4}$$

Thus, by using this approximation of the rotation matrix, the world-to-body rotation is reduced to

$$\mathbf{x}^b = \mathbf{R}_w^b(\psi)\mathbf{x}^w \tag{4.5}$$

where

$$\mathbf{R}_w^b(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

is a rotation about the z-axis. The inverse of the world-to-body rotation matrix is a body-to-world rotation matrix such as

$$\mathbf{R}_b^w(\psi) = (\mathbf{R}_w^b(\psi))^{-1} \tag{4.7}$$

$$= \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

4.2 System Architecture

The system architecture is displayed in Figure 4.1. The system is constructed of 6 components:

- Quadcopter
- Mission Control
- PID Control
- Kalman Filter
- TCV
- DNN CV.

This system can use a computer-simulated quadcopter or the physical quadcopter.

Mission Control is the guidance and navigation system for the quadcopter. It receives the current body frame pose estimate $\hat{\mathbf{x}}$ as input and produces the current world frame desired pose $\mathbf{x}_r$ for the quadcopter.

PID-control is the quadcopter pose controller, receiving the desired quadcopter pose from Mission Control, and produces velocity references $\mathbf{v}_r$ to which the on-board controller of the quadcopter follows in order to reach the desired pose.

The Kalman Filter fuses sensor measurements to produce one quadcopter pose. A Kalman Filter was chosen for sensor fusion because

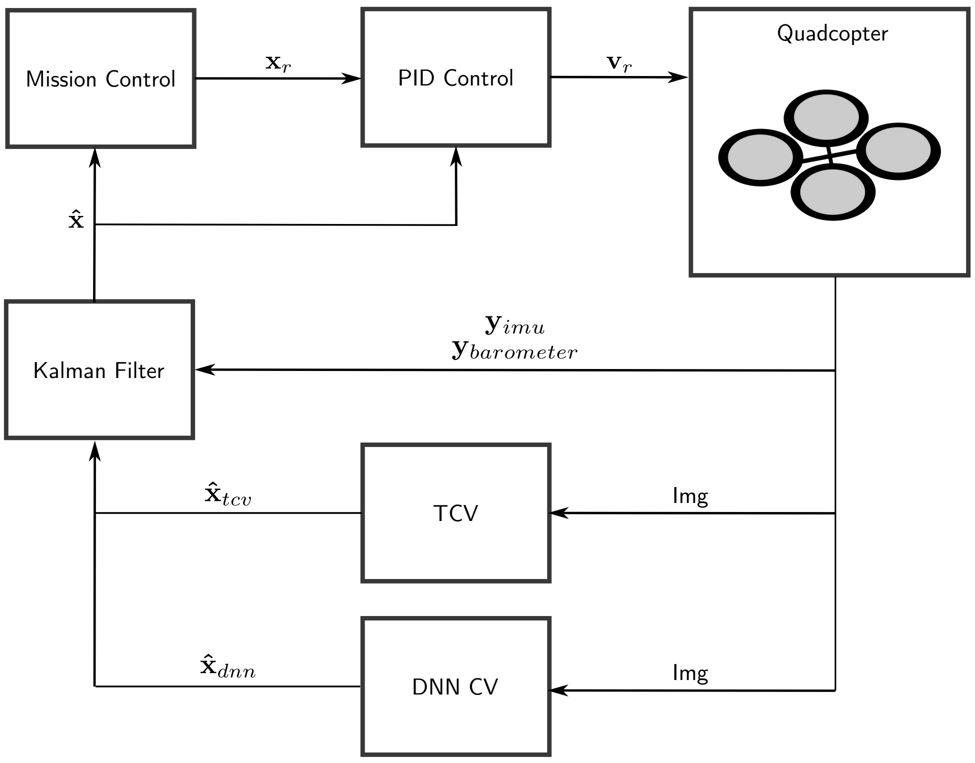


FIGURE 4.1: System Architecture

- It is more robust to sensor failure because it both fuses pose measurements and predicts the pose simultaneously.
- It dynamically adapts to missing- or infrequent sensor measurements by updating the current estimate based on the current uncertainty of the estimate
- It can easily be expanded to include more sensors.
- It can easily be adapted for use with another quadcopter.

The Kalman Filter receives IMU data $\mathbf{y}_{IMU}$, pressure data $\mathbf{y}_{barom}$ and computer vision pose estimates $\hat{\mathbf{x}}_{tcv}$ and $\hat{\mathbf{x}}_{dnn}$, and produces the current body frame pose estimate of the quadcopter $\hat{\mathbf{x}}$.

Two different computer vision models are processing images concurrently and are modeled as two independent sensors:

- **DNN CV**: A Deep Neural Network-based computer vision model, adapted from an earlier thesis by the author [2].
- **TCV**: A Traditional Computer Vision-based computer vision model,

Both computer vision models receive images from the drone's bottom facing camera as input and produce body frame pose estimates relative to the landing platform.

4.3 Computer Vision Pose Estimation

The author proposed, in an earlier thesis, a CV pose estimation algorithm by combining two computer vision methods [2]. The two methods are:

- Pose estimation method by TCV methods by T. Sundvoll [1].

- Pose estimation method using DNN CV methods created by the author [2],

Both methods require detecting the circular landing platform to produce a position estimate, and both require detecting the orange arrow to estimate yaw orientation.

The TCV method is created by Sundvoll [1]. Color segmentations, Canny edge detector [36] and Harris corner detection [37] are used to identify the circular landing platform, the H and the arrow. The center, radius, and yaw of the landing platform are calculated based on these features' location and size.

The DNN CV method proposed by Hove in [2] estimates the center and radius of the landing platform by using the location and sizes of YOLO bounding boxes [43]. The DNN is trained to detect three classes:

- Helipad
- H
- Arrow

Helipad is the complete landing platform, *H* is the white H in the center of the landing platform, and *arrow* is the orange triangle. A model image of the landing platform with labels for the three classes can be seen in Figure 4.2. The center of the landing platform is found by calculating the center of a bounding box of a detected *Helipad*. The radius of the landing platform in pixel coordinates is found by half of the longest side-length of a bounding box surrounding the *Helipad*.

Both methods estimate the pose of the quadcopter by first detecting the landing platform in the camera frame of the bottom facing camera of the quadcopter. Subsequently, the position is calculated by knowing the size- and location of the landing platform in the camera frame and transforming that to body frame coordinates based on known camera intrinsics of the AR.Drone bottom facing camera, as well as the real size of the landing platform. The quadcopter orientation is calculated as the angle between a vector connecting the orange arrow on the landing platform and the center of the landing platform and a horizontal vector. Yaw angle on the landing platform is illustrated in Figure 4.3, where (x_a, y_a) is the center of the bounding box encapsulating *Arrow*, and (x_h, y_h) is the center of the bounding box encapsulating *Helipad*. Further elaboration on the implementation of the DNN CV algorithm is presented in [2], and the implementation of the TCV algorithm is presented in [1].

The TCV method and the DNN CV method are processing the images independently and concurrently. Combining computer vision methods may result in a computer vision system that is more robust and accurate than each method independently.

The TCV method proposed by Sundvoll [1] and the DNN CV method proposed by the author [2] is based upon the assumptions that the landing platform is perfectly horizontally oriented. and that the quadcopter pitch and roll are zero. These assumptions are close to accurate in windless conditions and when quadcopter control is sufficiently slow to avoid large pitch and roll movements. Simulation results from [2] and [1] seem to verify these assumptions.

The DNN CV method is further developed in this thesis by:

- being trained on a larger dataset, achieving a higher mAP
- being trained on a dataset consisting of images of the physical landing platform

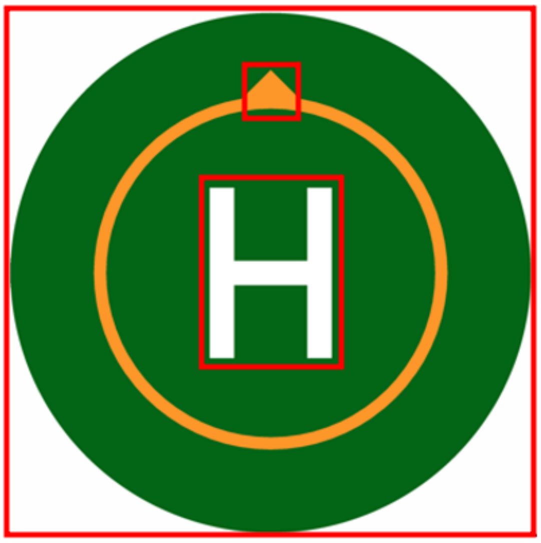


FIGURE 4.2: The landing platform with bounding boxes encapsulating the three classes; *Helipad*, *H* and *Arrow*.

- fixing bugs that were causing yaw estimation errors reported in [2].
- Misdetection checks and filtering

Building Datasets

The DNN CV algorithm proposed by the author in [2] is based on a trained YOLOv4 algorithm detecting the landing platform. For training, the algorithm requires a dataset that contains images of landing platforms, labeled with the classes *Helipad*, *H* and *Arrow*.

The author collected a dataset with 1083 images of the simulated landing platform for a project thesis [2]. The YOLOv4 model proposed in the project thesis was trained with that dataset and achieved a mAP of 86.9%. That project thesis dataset formed the basis for a further developed dataset used for training a YOLOv4 model in this thesis.

The experiments in this thesis are performed in two different visual environments; a simulation environment and an indoor environment. Two different datasets are created in order to train one model for each environment.

The Simulation dataset, called SimSet, consists solely of images of the landing platform from the simulations, at altitudes between 0.5m and 20m. The images are captured with the AR.Drone's bottom-facing camera by manually controlling the quadcopter over the landing platform in the simulation.

The Real-World dataset, called RealSet, consists of images of the physical landing platform. These images are captured by manually controlling the quadcopter 0.5 to 2.5m over the landing platform while capturing video at 5 fps using the AR.Drone's bottom facing camera. In order to make the dataset as general as possible, the video is captured with three different backgrounds:

- indoor with a wood floor background and indoor light conditions.
- outdoor with a grass background, in sunny conditions.
- outdoor with a wood floor background, in sunny conditions.

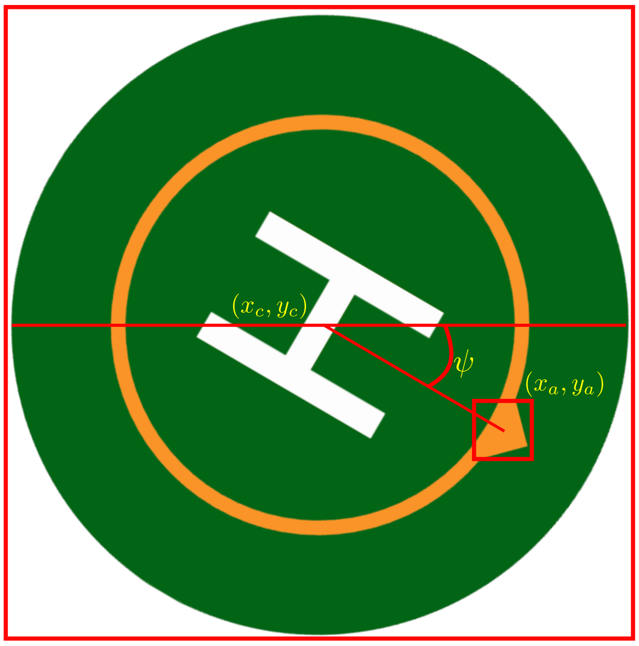


FIGURE 4.3: Calculating yaw as the angle between a flat line in the camera frame and the straight line between the centers of the Helipad and the Arrow.

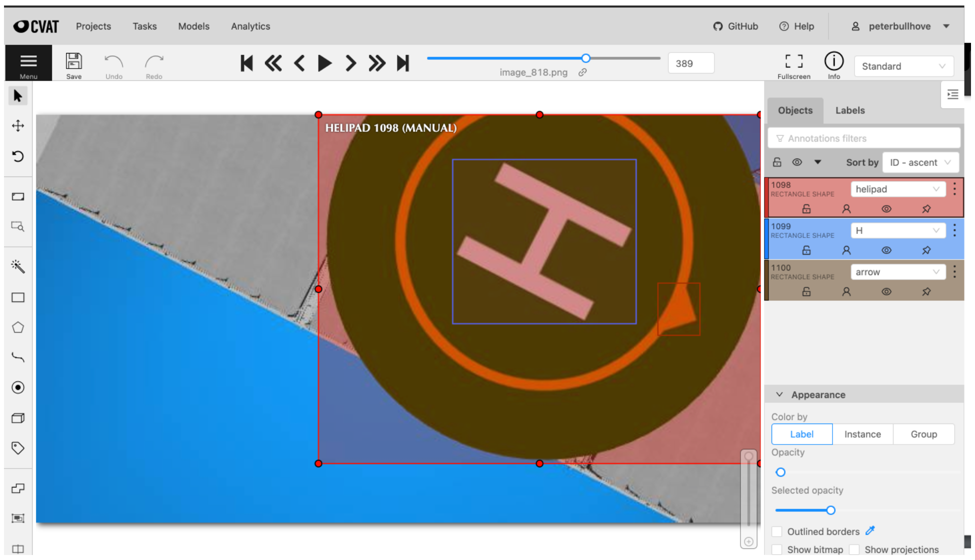


FIGURE 4.4: The labeling process in cvat.

The SimSet consists of 2182 images, including the initial 1083. The RealSet consists of 1321 images.

The two datasets are labeled using an online labeling tool called cvat [65]. The images are labeled by manually drawing bounding boxes around the classes in the images, specifying which class each box belongs to. An image of the labeling process can be seen in Figure 4.4.

YOLO requires images to be square, with side lengths are pixel multiples of 32. Higher resolution images result in more accurate but significantly slower detection. The image resolutions that was chosen for the datasets in this thesis were 416x416, where sufficient detail is present in the image for accurate detections. At the same time, the inference speed is sufficiently high for real-time control. The bottom-facing camera of the AR.Drone 2.0 is 320x240, so any higher resolution than that does not increase detection accuracy.

The datasets are partitioned into training sets, validation sets, and test sets, required to assess the DNN training. The training sets are used to train the model, while the validation sets are datasets that the model uses to evaluate performance during training. The test sets are datasets that have not been seen during training and can therefore be used to test the final model performance. The training set consists of 70% of the images, while the validation set and the test set consists of 20% and 10%, respectively.

Data augmentation is performed in order to increase the size of the datasets further, as well as to make the model more general robust. Augmentation is performed using an online data augmentation tool called Roboflow [4]. A table showing all augmentations that are performed on the datasets can be seen in Section 4.3.1. Augmentation is only performed on the training set, so after augmentation, the training set contains 88% of the images, the validation set contains 8% of the images, and the testing set contains 4% of the images in the dataset.

The augmentations that are performed on the SimSet are chosen to provide the best performance in simulations. The simulation environment is a stable visual environment, where the lighting or background behind the landing platform never changed, and thus no color augmentations are chosen. The augmentations that are

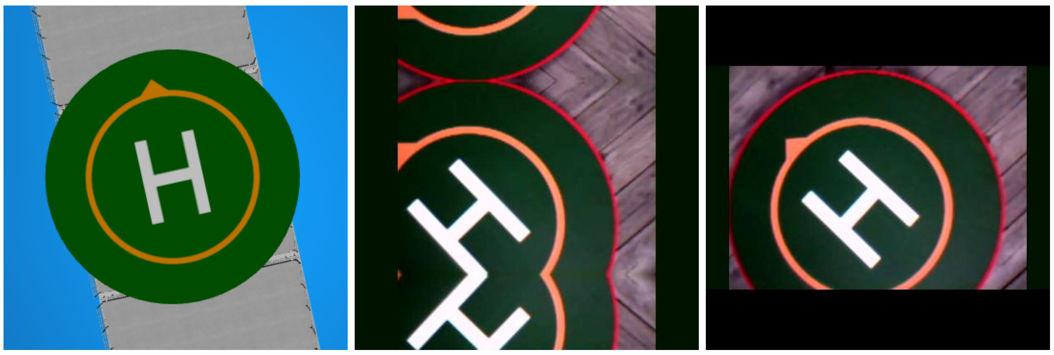

(A) SimSet (B) MirrorSet (C) RealSet

FIGURE 4.5: An image from each of the three datasets.

chosen for the SimSet are image rotations and image flipping in order for the algorithm to be invariant to yaw rotation. The SimSet is augmented to be 5229 images.

The RealSet is augmented by image flipping as well as image shear, grayscale and exposure. The RealSet is augmented to be 3171 images.

A third dataset is created by resizing RealSet in a different way, called mirror edges, which resize the images to 416x416 by mirroring the edges, so the image becomes square and subsequently scaling the resolution. Edge mirroring causes some images to contain multiple elements of the same classes and the model may get more training detecting the classes this way. However, the mirroring may mirror the middle of the landing platform, resulting in the mirrored image containing two side-by-side half landing platforms, which looks like one landing platform. The mirroring may cause the YOLO detection algorithm to struggle because the mirrored landing platform should be detected as two separate landing platforms. The MirrorSet does not accurately represent the final detection scenario, and the model may not perform well on the test set. Whether the increased number of detection cases caused by the mirroring will make up for the different training images will be seen after training.

An image from each of the three datasets is displayed in Figure 4.5.

All three datasets are pretty balanced. Each landing platform contains one type of each class. Some images do not contain the *Arrow* because the camera view does not contain the complete landing platform, but not a sufficient amount to cause dataset imbalance problems.

Training YOLOv4 CV Models

A YOLOv4 model is trained for each of the three datasets. Training is performed using Google Colaboratory's free GPU [66], using a YOLOv4 training script provided by Roboflow [4] [67]. A table of training data, training parameters and final mAP for the three datasets can is presented in Section 4.3.1.

Training a model for the SimSet is performed using transfer learning with initial weights being the officially released YOLOv4 ConvNet weights from [68]. Training of the MirrorSet and RealSet is performed using transfer learning using a fully trained SimSet model. Transfer learning reduces the number of training images required, which is helpful since the gathered datasets are small. Transfer learning from the SimSet model should be effective since the simulated landing platform is quite similar in shape, size and color to the physical landing platform. In addition, RealSet and MirrorSet consist of only 3171 images, which is quite low to train a model without transfer learning.

Training is stopped when mAP plateaus to avoid overfitting. The training parameters used for training and the final mAP of the models are displayed in Section 4.3.1.

The final mAP of the RealSet model is 98.11% which is quite high. The mAP of the MirrorSet not as good, reaching mAP of 93.56%, and the RealSet model is therefore chosen for the DNN CV. By analyzing output images from the model trained on the MirrorSet, the model was prone to detect multiple overlaying bounding boxes over the same objects in the image and sometimes detect the objects as multiple smaller objects instead of one large object. The mirroring created problems because it resulted in a dataset that was not similar to the test set resulting in a model unable to produce accurate detections.

The mAP of the SimSet was 89.46%, quite a bit less than the mAP of the RealSet. The lower mAP of the SimSet may be surprising, as the visual environment of the SimSet is stable and less varying than the RealSet. The main challenge with SimSet was detecting the *Arrow* and the *H*. Many of the images in the SimSet were taken from a very high altitude, causing the landing platform to be very small in the images. Subsequently, the *Arrow* and the *H*, which are smaller parts of the landing platform, were tiny. Kumar B. reports in [5] that YOLO struggles to detect small objects, and these training results seem to reflect those results. RealSet consists of images captured from between 0.5m and 2.5m in altitude and are similar to the actual experimental images.

4.3.1 Computer Vision Processing Delay

The computer vision algorithms are computationally demanding. The DNN CV algorithm runs on the GPU and CPU of the computer and performs inference at 45-50 fps at an image resolution of 416x416. The TCV algorithm produces estimates at a rate of 10-12 fps. When running both of these computer vision algorithms concurrently, there is a computer power bottleneck. When running the CV methods concurrently, the TCV algorithm performs inference at 5-6 fps and the DNN CV at 22-26 fps. These results can be seen in Table 4.1. A decrease in estimate frequency is not desirable. The resulting estimation frequency is arguably sufficient for real-time pose estimation because the quadcopter system is slow to react. The combined estimation frequency is between 27-31 fps, which is quite a lot higher than the original pose estimation frequency of the TCV algorithm.

A more demanding problem with the decreased estimation frequency with the TCV algorithm is the time delay between when the computer receives an image and when the TCV produces an estimate from that image. That time delay, called a processing delay, is equal to $\frac{1}{f_{\text{cv}}}$, where f_{cv} is the inference frequency of the CV method. The processing delay for the TCV algorithm, while running the DNN algorithm concurrently, is calculated to be between 166ms to 200ms. Such a delay can be quite significant while running a real-time control system. For the DNN CV algorithm, the processing delay is between 38 ms and 45ms. The total control loop delay is the sum of both processing delay and communication delay. Communication delay is reported in Chapter 3 to be approximately 130ms for the real quadcopter and negligible in the simulations.

	Running Separately	**Running Concurrently**
DnnCV	45-50 fps	22-26 fps
TCV	10-12 fps	5-6 fps
Total	45-50 fps or 10-12 fps	27-31 fps

TABLE 4.1: CV inference frequencies. Running both computer vision algorithms concurrently reduces frequencies

	SimSet	**MirrorSet**	**RealSet**
Number of Images:	2182	1321	1321
Augmented to:	5229	3171	3171
Augmentations:			
Resize by:	Black edges	Mirror edges	Center crop
Horizontal flip:	Yes	Yes	Yes
Vertical flip:	Yes	No	No
$\pm$ 15 deg rotations:	Yes	No	No
$\pm$ 90 deg rotations:	Yes	No	No
$\pm$ 2 deg shear:	No	Yes	Yes
Grayscale:	No	25% of images	25% of images
Exposure:	No	$\pm$ 25%	$\pm$ 25%
Transfer Learning from:	YOLOv4 ConvNet weights by [68]	SimSet	SimSet
Training Parameters:			
Batch size:	64	64	64
Img size:	416x416	416x416	416x416
Momentum:	0.949	0.949	0.949
Weight decay:	0.0005	0.0005	0.0005
Learning rate:	0.001	0.001	0.001
Num training Epochs:	9000	8000	4000
Final mAP:			
Helipad:	98.25%	93.26%	98.69%
H:	90.71%	90.91%	96.29%
Arrow:	79.42%	93.26%	99.36%
Total:	89.46%	93.56%	98.11%

TABLE 4.2: YOLOv4 training on the three different datasets and the resulting mAP

4.4 Sensor fusion using Kalman Filter

The internal state space of this Kalman Filter implementation is defined to be

$$\mathbf{x}_k := (x_k, y_k, z_k, \phi_k, \theta_k, \psi_k, \dot{x}_k, \dot{y}_k, \dot{z}_k) \in \mathbb{R}^9 \tag{4.9}$$

where

- the subscript k signifies the value of the respective state at time-step k
- x,y and z are linear coordinates of the drone position in body frame in relation to the landing platform, represented in metres.
- ϕ, θ and ψ are roll-, pitch- and yaw angle of the quadcopter, representing rotations around body coordinate axis' x,y and z respectively. The angles ϕ and θ are estimated solely by the quadcopter's on-board angle estimation system, and are not filtered further in this implementation.
- $\dot{x}$, $\dot{y}$ and $\dot{z}$ are velocities of the drone in the linear in relation to the landing platform, represented in metres per second.

The state-space does not include any angular velocities as these provide little value in this pose estimation system.

4.4.1 Prediction

Prediction is performed by propagating the current linear velocities one time step into the future. The prediction is implemented as

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} + \delta t \begin{bmatrix} \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix} \tag{4.10}$$

where δt is the time in seconds since the last prediction.

The error covariance is predicted by

$$\mathbf{P}_{k+1} = \mathbf{P}_k + \mathbf{Q} \tag{4.11}$$

where $\mathbf{Q} \succ 0 \in \mathbb{R}^9$ and is prediction uncertainty. Thus, the error covariance is increasing for every prediction. The value of $\mathbf{Q}$ is set to be

$$\mathbf{Q} := \begin{bmatrix} 10^{-3}\mathbf{I}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & 10^{-3}\mathbf{I}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & 10^{-2}\mathbf{I}_{3x3} \end{bmatrix}. \tag{4.12}$$

4.4.2 Update

The quadcopter pose is updated by measurements from the following sensors

- DNN CV
- TCV
- IMU
- Barometer

- GPS (in simulations only)

The sensor uncertainty parameters are displayed Table 4.3, and the individual sensors updates are presented below.

DNN CV and TCV

Both CV pose estimation systems produce estimates of linear position x, y and z, and yaw angle ψ.

The output matrix for the computer vision systems, $\mathbf{C}_{\text{cv}}$, is therefore defined as

$$\mathbf{C}_{\text{cv}} := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{4.13}$$

Barometer

The barometer on the AR.Drone outputs a pressure P_k in Pascal. Pressure at sea level is approximately linearly decreasing with altitude, at a rate of about 11.3 Pascal/m [69]. Pressure is dependent on temperature, weather conditions and absolute altitude above sea level. By capturing the baseline pressure P_0 during takeoff, the altitude is calculated as

$$y_{\text{Barom}} := \frac{P_0 - P_k}{11.3} \tag{4.14}$$

The barometer provides altitude measurements and the output matrix for this sensor is therefore

$$\mathbf{C}_{\text{Barom}} := \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.15}$$

GPS

A GPS is not available for the physical AR.Drone, but is available in simulations. GPS signals are created by adding mock sensor noise ρ to the ground truth position of the drone.

GPS measurements are published at a rate of 2 Hz, and the GPS noise is zero mean and has a standard deviation of 5 cm. Thus the mock GPS data is created by

$$\mathbf{y}_{\text{gps}} := \mathbf{x}_{\text{xyz}}^{gt} + \boldsymbol{\rho} \tag{4.16}$$

where $\mathbf{x}_{\text{lin}}$ is the ground truth position of the quadcopter and $\boldsymbol{\rho} \in \mathbb{R}_3$ is generated normally distributed sensor noise such that $\boldsymbol{\rho} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma})$ where $\sigma = \begin{bmatrix} 0.05 \\ 0.05 \\ 0.05 \end{bmatrix}$.

The GPS provides position measurements and the output matrix for this sensor is therefore

$$\mathbf{C}_{\text{gps}} := \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3x6} \end{bmatrix} \tag{4.17}$$

IMU and Magnetometer

The Kalman Filter state space includes linear velocities and performs position predictions based on these velocities. Velocity is predicted using acceleration data from the accelerometer in the IMU. The accelerometer measures the specific force on the body as well as the acceleration of gravity. The acceleration of gravity is an acceleration vector with a magnitude of 9.81 and a direction pointing down towards the center of the earth.

The accelerometer is fixed to the quadcopter body frame axis. If the quadcopter is laying flat, meaning that pitch an roll are zero, then all the force of gravity is measured by the z-component of the accelerometer. If the quadcopter is pitched or rolled, then the force of gravity is measured as

$$\mathbf{g}(\phi, \theta) := 9.81 \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\theta)\cos(\theta) \end{bmatrix} \tag{4.18}$$

In order to find the specific force of the quadcopter one will have to estimate the gravity vector, and subtract it from the accelerometer data.

The estimated gravity vector at time-step k is therefore calculated by inserting the estimated roll and pitch, $\hat{\theta}$ and $\hat{\phi}$, into Equation (4.18). The resulting estimated gravity vector is therefore

$$\hat{\mathbf{g}}(\hat{\phi}_k, \hat{\theta}_k) = 9.81 \begin{bmatrix} -\sin(\hat{\theta}_k) \\ \cos(\hat{\theta}_k)\sin(\hat{\phi}_k) \\ \cos(\hat{\theta}_k)\cos(\hat{\theta}_k) \end{bmatrix} \tag{4.19}$$

This estimated gravity vector is subtracted from the acceleration data from the IMU to find the specific force of the quadcopter

$$\mathbf{a}_k = \mathbf{a}_{\text{IMU},k} - \hat{\mathbf{g}}_k(\hat{\phi}, \hat{\theta}) \tag{4.20}$$

$$\tag{4.21}$$

This acceleration is then used to predict the current velocity:

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \delta t \mathbf{a}_k \tag{4.22}$$

Roll and pitch are estimated by using the on-board estimates from the AR.Drone

$$\hat{\phi}_k = \hat{\phi}_{\text{AR},k} \tag{4.23}$$

$$\hat{\theta}_k = \hat{\theta}_{\text{AR},k} \tag{4.24}$$

where $\hat{\phi}_{\text{AR},k}$ and $\hat{\theta}_{\text{AR},k}$ are the AR.Drone's on-board estimate for roll and pitch respectively, at time-step k.

Yaw is updated using the AR.Drone on-board estimate for yaw, $\hat{\psi}_{\text{AR},k}$. In order to compensate for magnetometer drift, yaw is updated as the difference between the previous yaw measurement from the AR.Drone and the current. Therefore, a constant bias in yaw measurements from the AR.Drone will not conflict with yaw updates from the computer vision algorithms. Yaw is therefore updated as

$$\hat{\psi}_k = \hat{\psi}_{k-1} + (\hat{\psi}_{\text{AR},k} - \hat{\psi}_{\text{AR},k-1}) \tag{4.25}$$

Velocity Updates Using Differentiated Position Measurements

Velocity is predicted using IMU acceleration measurements. Uncertainty of velocity estimate increases over time without updates since velocity is predicted using integrated sensor measurements. Sensor noise and bias will be integrated and results in an inaccurate velocity estimate.

Velocity updates are implemented in order to deal with uncertain velocity estimates. Velocity updates are based on differentiating the position estimates from the sensors. Differentiating position estimates is done by capturing the time difference δt between the current and last sensor update from a specific sensor. Velocity is calculated as the measured position difference divided by the δt. Thus, for a sensor i providing a measurement at time step k velocity is updated using

$$\mathbf{y}_{\text{vel}} := \frac{\mathbf{y}_k^i - \mathbf{y}_{prev}^i}{\delta t} \tag{4.26}$$

The sensor noise for velocity measurements is assumed higher than the sensor noise for the specific sensor. It is known from literature that differentiating increases high-frequency noise. Therefore velocity updates uncertainty for all sensors $i = 0, 1, ..., n$ is set 10 times that of the sensor, such that

$$\mathbf{R}_{i,\text{vel}} = 10\mathbf{R}_i \tag{4.27}$$

By using $\mathbf{v} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^\top$ the velocity update is implemented as

$$\mathbf{v}_{k|k} = \mathbf{v}_{k|k-1} + \mathbf{K}_{\text{vel},k}(\mathbf{y}_{\text{vel}} - \mathbf{v}_{k|k-1}) \tag{4.28}$$

where $\mathbf{K}_{\text{vel},k}$ is Kalman gain calculated for velocity updates, using the sensor uncertainty for velocity and the components of the error covariance matrix $\mathbf{P}$ relating to the velocity states.

4.4.3 Pre-Filtering

Sensor misdetections may severely impact the system's stability because the Kalman Filter does not remove any misdetections from the sensors. While some of the sensors are generally quite precise, they can have occasional misdetections, updating the pose estimate with a far away measurement from the current estimated position.

A discarding filter is applied to measurements in order to discard misdetections. CV measurements measuring a pose outside of the visual range of the landing platform cannot be correct and are therefore discarded. Estimates that deviate more than 2 meters from the current estimated position are also discarded, as long as the current estimate uncertainty is low.

In order to further remove misdetections and remove unwanted oscillations, a Moving Median Averaging (MMA) filter, developed by Sundvoll [1], is applied.

By specifying an averaging filter size A, and a median filter size M, the total size of the filter is $A + M + 1$, named T. The median filter outputs a list of the A median measurements from last T measurements, and the averaging filter outputs one estimate, which is the average of the median estimates. A more in-depth presentation of the filter is found in [1].

A separate MMA filter is created for the TCV algorithm, DNN CV algorithm and the barometer. M and A are chosen independently for the sensors depending on the amount of sensor noise and misdetections. The GPS measurements are infrequent

	KF Sensor Uncertainty Parameters
TCV	$\mathbf{R}_{tcv} = 0.01^2\mathbf{I}_{4x4}$
DNN CV	$\mathbf{R}_{dnnCV} = 0.01^2\mathbf{I}_{4x4}$
Barometer	$\mathbf{R}_{barom} = 1^2$
GPS	$\mathbf{R}_{gps} = 0.05^2\mathbf{I}_{3x3}$

TABLE 4.3: Sensor uncertainties parameters used in the Kalman Filter. Found through a combination of noise analysis and experimental tuning

	Simulation (M,A)	**Experimental (M,A)**
TCV:	(1,1)	(5,5)
DNN CV:	(1,1)	(5,5)
Barometer:	(1,1)	(200,200)

TABLE 4.4: Median filter Size M and Averaging filter size A for the sensors in simulations and experiments.

and sufficiently accurate such that a filter is be of little value. The IMU data are not filtered using an MMA filter. Instead, unreasonably high accelerations are discarded.

The simulated sensors are less noisy and produce significantly less misdetections than in the experiments, and subsequently the filter sizes in the experiments are chosen to be higher. The MMA filter sizes are displayed in Table 4.4. While the barometer filter sizes may seem high, with an update frequency of 200Hz the filter averages data over one second in time, which is found to be acceptable for altitude control.

4.5 Mission Control System

Mission Control is implemented as a Finite State Machine (FSM). The states defined for the FSM are chosen to be general states that can be combined to constitute several different autonomous quadcopter missions.

The FSM states are:

- **INIT**: Quadcopter sensor calibration, assert that system is ready.
- **TAKEOFF**: Quadcopter takeoff and start PID-control.
- **HOVER**: Hover at stationary setpoint for specified duration.
- **LANDING**: Perform automatic landing on landing platform.
- **MOVING**: Move quadcopter to specified setpoint.
- **PHOTOTWIRL**: Perform four 90 degree yaw rotations, and capture photo after each completed rotation.
- **IDLE**: Quadcopter stationary on the ground, no process running.
- **ERROR**: Error state. Perform slow descent for safe quadcopter retrieval.

Mission Control navigates by sending pose setpoints $\mathbf{x}_r$ to the PID-controller and by sending camera-, landing-, and takeoff commands to the drone. Mission Control continuously checks whether the quadcopter has reached the desired setpoint, subsequently proceeding to the next state.

4.5.1 Safety and error-handling

The ERROR state is for safety and quadcopter retrieval in case the mission fails. Mission Control will switch to the ERROR state if 90 seconds have passed without any state switches, which is a duration that is deemed sufficient for the experiments that are performed in this thesis.

Additionally, Mission Control will switch to the error state if the pose estimates for the quadcopter are outside a specified box of operations. The box of operations is a specified operational area where the quadcopter is allowed to perform, implemented to be within 15m from the landing platform and within 7m over the landing platform. If the quadcopter estimates its position to be outside of this volume, the Mission Control switches to the ERROR state.

4.6 PID-Control

A PID- (proportional, integral, derivative) controller calculates control inputs for the quadcopter. As can be seen in Figure 4.1 the PID-controller of the system receives two inputs, $\mathbf{x}_r \in \mathbb{R}^6$ and $\hat{\mathbf{x}} \in \mathbb{R}^6$. $\mathbf{x}_r$ is the world frame desired pose for the quadcopter, called reference pose, which is the output from the Mission Planning System. $\hat{\mathbf{x}}$ is the estimated quadcopter pose received from the Kalman Filter. The output from the PID-controller is $\mathbf{v}_r \in \mathbb{R}^4$, which is the desired quadcopter velocity in the three linear axes as well as angular velocity in yaw. The on-board controller of the AR.Drone controls the quadcopter motors to reach this velocity reference.

The PID-controller is a modified version of a PID-controller implemented by Sundvoll in his master project [1]. The alterations performed on Sundvoll's PID-controller are:

- an implementation of *movement slicing*, a technique proposed by Sani et al. [13].
- changed PID-controller to body frame coordinates instead of world frame coordinates, required for missions involving yaw rotations.
- tuning of control parameters

The PID-controller calculates the output velocity as a function of the quadcopter pose error. The error is the difference between the pose reference and the estimated pose. The estimated pose is represented in body frame, while the reference pose is represented in world frame. The reference pose is transformed to body frame by

$$\mathbf{x}_r^b = \mathbf{R}_w^b \mathbf{x}_r \tag{4.29}$$

where $\mathbf{R}$ is the world-to-body transformation matrix. The body frame reference pose is updated continuously using the most recent estimated quadcopter yaw.

The pose error $\mathbf{e}$ is defined as

$$\mathbf{e} := \mathbf{x}_r^b - \hat{\mathbf{x}} \qquad \mathbf{e} \in \mathbb{R}^6 \tag{4.30}$$

The velocity control vector $\mathbf{v}_r$ does not have any components relating to pitch or roll. In order for the components of the error state to match the components of the control signals a new error state is defined; $\mathbf{e}^\star \in \mathbb{R}^4$, which is a subset of the states

	Simulation Parameters	Experiment Parameters
$\mathbf{K}_p$	x: 0.7 y: 0.7 z: 0.7 ψ: 0.02	x: 0.1 y: 0.1 z: 0.1 ψ: 0.0
$\mathbf{K}_d$	x: 0.5 y: 0.5 z: 0.5 ψ: 0.0	x: 0.0 y: 0.0 z: 0.1 ψ: 0.0
$\mathbf{K}_i$	x: 0.01 y: 0.01 z: 0.001 ψ: 0.0	x: 0.001 y: 0.001 z: 0.001 ψ: 0.0

TABLE 4.5: PID-control parameters for in simulations and experiments. Found through experimental adaption of the parameters used by Sundvoll [1].

in **e**, such that

$$\mathbf{e}^\star := \begin{bmatrix} x_e \\ y_e \\ z_e \\ \psi_e \end{bmatrix} \tag{4.31}$$

where x_e, y_e, z_e, ψ_e are error components of **e** in the respective dimensions.

Control parameter vectors $\mathbf{K}_p, \mathbf{K}_d, \mathbf{K}_i \in \mathbb{R}^4$ are defined, corresponding to proportional-, derivative- and integrative control parameters respectively. The elements of each control parameter vector are control parameters corresponding to errors in the x-, y- and z- axes and yaw rotation. The control parameters are displayed in Table 4.5. The experimental control parameters are significantly lower than in control parameters for simulations because maneuvers with the physical quadcopter lead to instability.

In a discrete time system, the integral error at time-step k, $\mathbf{e}^\star_{i,k}$, is calculated as the sum of all errors up to the current time-step, such that

$$\mathbf{e}^\star_{i,k} = \sum_{j=0}^{k} \mathbf{e}^\star_j \tag{4.32}$$

and the derivative error $\dot{\mathbf{e}}^\star$ is calculated as the difference between the errors in the two most recent time-steps, divided by the duration between the time-steps, hence

$$\dot{\mathbf{e}}^\star_k = \frac{1}{\delta t}(\mathbf{e}^\star_k - \mathbf{e}^\star_{k-1}) \tag{4.33}$$

$$\tag{4.34}$$

where δt is the duration of time between time-steps.

By combining this, the reference velocity at time-step k is calculated by

$$\mathbf{v}_{r,k} = \mathbf{K}_p \mathbf{e}^\star_k + \mathbf{K}_d \dot{\mathbf{e}}^\star_k + \mathbf{K}_i \mathbf{e}^\star_{i,k}. \tag{4.35}$$

4.6.1 Movement Slicing

Sani et al. report that a standard PID-controller is not sufficient for stable control of the AR.Drone 2.0 with their implementation, where the drone tends to overshoot the target. This overshoot may be caused by time delay in communication with the AR.Drone. Sani et al. also suggest that the drone's inertia and control delay in the internal controller on the AR.Drone 2.0 are factors that contribute to the unsatisfactory results with the standard PID-controller. In response to this they propose a solution called *movement slicing* [13].

Movement slicing is a control technique that involves splitting control into time slots, where some of the time slots are moving slots, where the PID-controller controls the quadcopter, and halting slots, where the quadcopter halts. Movement slicing reduces the adverse effects of overshooting due to communication- and control delays.

Movement slicing is applied to the PID-controller, with moving slots and halting slots of 2 and 2 seconds. During the halting slots, the PID-controller does not apply any control signals to the quadcopter, and halts error integration as well. The onboard controller then controls the drone's velocity to zero. Admittedly, such a technique will invariably result in a slow quadcopter response because the quadcopter is standing still for a significant amount of time. Movement slicing is not deemed necessary in simulations, where communication delays are negligible. However, it is used during experiments with the physical quadcopter because it significantly improves control stability.

4.7 Adjustments To Enable Real-World Experiments

It is seen during experimental testing with the quadcopter that the position predictions using integrated accelerometer data from the IMU are severely inaccurate to the degree of causing instability and quadcopter crashes. Therefore the IMU accelerometer measurements and subsequent position predictions are discarded during the real-world experiments. The failure of the IMU may be due to gravity vector compensation. The specific force is calculated by removing an estimated gravity vector based on estimated pitch and roll. If the estimates for pitch and roll are inaccurate, the compensation of the gravity vector will result in inaccurate acceleration data.

Because of the lack of a GPS and the inaccurate IMU measurements, the system is dependent on frequent computer vision estimates for estimation of quadcopter position in x and y. Computer vision estimates are not reliable when the quadcopter is very close to the landing platform, found from testing to be below 0.5m in altitude. The current landing procedure involves multiple descending setpoints over the landing platform, which the quadcopter is to pass through in a slow and controlled manner. Since positional measurements are not available when low above the landing platform, the quadcopter may not notice if it starts drifting away from the platform. To deal with drifting while landing, the automatic landing is simplified to be a steady descent until the quadcopter hits the ground. This descent is quick, so the quadcopter does not drift during landing.

Yaw control is disregarded in the experiments. During experimental testing, it is seen that yaw maneuvers cause the quadcopter to gain altitude and started drifting. Accurate position control is deemed more important than accurate yaw orientation. Even small control parameters for yaw reduce the overall stability of the quadcopter system. Yaw control is therefore disregarded, thus reducing the control problem from 4 to 3 degrees of freedom.

Chapter 5

Results

A system for autonomous quadcopter control is proposed the previous chapter. In order to assess and validate the performance of the system a number of experiments and simulations are performed. This chapter presents experiments and simulations used to test the system as well as the results from the experiments and simulations.

5.1 Design of Simulation Tests and Experiments

In order to test the performance of the autonomous quadcopter the quadcopter system is subjected to several performance tests. These tests are intended to measure the performance and robustness of the quadcopter autopilot. Tests are performed in both the simulation environment and with the physical quadcopter.

Five tests are designed, three of which are simulated and two with the physical AR.Drone. An overview- and the specifications of the tests are seen in Table 5.1. A further presentation of the tests follows in the sections below.

5.1.1 HoverMission

HoverMission is a test in which the drone performs automatic hovering at a setpoint located 1.5 meters directly above the landing platform with zero yaw. The objective of *HoverMission* is to test

- the accuracy of the pose estimation system in simulations.
- whether the control system can provide sufficiently accurate control of the quadcopter using pose estimates as input in simulations.

5.1.2 LandingMission

The *LandingMission* is a test in which the drone performs an automatic landing operation on the landing platform. An automatic landing operation is an integral part of an autonomous quadcopter mission, and a robust automatic landing is therefore required for a safe and reliable autonomous system. The initial position of the quadcopter is 2.0m, 2.0m and 5.0m in x,y and z respectively and oriented -90 degrees in yaw. This mission consists of two steps; Move to 1.5m centered above the landing platform, then perform an automatic landing operation.

Pose estimates during a landing scenario are challenging since the CV algorithms are less valuable when very close to the landing platform. The landing platform fills the entire camera view, and the current implementation of the algorithms is not accurate close to the landing platform.

GPS is not used for this test to demonstrate that the pose estimation is sufficiently accurate to perform automatic landing without the use of GPS. The physical

Name:	**Environment:**	**Initial pose:**	**Mission state squence:**
HoverMission	Simulation	Manually controlled to ∼ [0,0,1.5,0,0,0]	1. Hover
LandingMission	Simulation	[2,2,5,0,0,-90]	1. Move to [0,0,1.5,0,0,0] 2. Hover (5 sec) 3. Automated Landing 4. Idle
PhotoMission	Simulation	[0,0,0,0,0,0]	1. Init 2. Takeoff 3. Move to [10,0,5,0,0,0] 4. Phototwirl 5. Move to [10,10,5,0,0,0] 6. Phototwirl 7. Move to [0,0,5,0,0,0] 8. Automated Landing 9. Idle
Hover and Land	1. Indoor 2. DNV ReVolt	Manually controlled to ∼ [0,0,1.5,0,0,0]	1. Hover 2. Automated Landing
RealMission	Indoor	[0,0,0,0,0,0]	1. Init 2. Takeoff 3. Hover (5 sec) 4. Automated Landing 5. Idle

TABLE 5.1: Mission experiments and simulations. Quadcopter pose expressed as [x,y,z,ψ,θ,ϕ].

AR.Drone 2.0 is not equipped with a GPS, and therefore it is interesting to demonstrate the system in simulations without the GPS in order to perform fair comparisons. Quadcopter may also be required to operate in GPS-denied environments.

Without a GPS, the landing platform must be in the camera view to get x and y estimate updates. Position predictions are performed with integrated accelerometer data. Predictions based on integration of data are unreliable after a while without updates from other sensors. Therefore, with no GPS, the system is vulnerable if the landing platform is outside of the camera view, as it has no other sensors than the IMU to navigate.

5.1.3 PhotoMission

PhotoMission is a long-range mission that is performed in simulations. The drone is initialized on the landing platform, instructed to fly to a setpoint, and then instructed to perform a 360-degree spin while capturing four images of the area using the front-facing camera. Then it is controlled to move to another location before returning to the landing platform by performing an automatic landing. The quadcopter performs the whole mission without any human intervention. Such a mission can be used for search- and rescue operations and common inspection missions. The exact setpoints are displayed in Table 5.1.

This mission is constructed to test the long-range capability of the quadcopter system and test the capabilities of the quadcopter system when outside of the visual range of the landing platform. Testing the long-range capabilities of the quadcopter system is interesting because that is a test of the quadcopter performance without CV updates. The position is then estimated using GPS, IMU and barometer. Yaw rotation is using magnetometer and gyroscope. The magnetometer is subject to negligible drift in the simulations, producing very accurate yaw measurements. An accurate magnetometer is important because the quadcopter is navigating in body coordinates. If yaw estimates are off, the quadcopter may move in the wrong direction to reach the desired position. While the GPS provides accurate position measurements regardless of any error in yaw estimates, the quadcopter relies on knowing which direction it is facing to move in the right direction. Reported magnetometer drift [17] with the physical drone may pose a significant problem during long-range experiments.

While the simulated quadcopter does have a GPS, a robust computer vision system is still necessary for a robust landing mechanism. The computer vision system is producing pose estimates that are more precise than the GPS measurements, as seen in Table 5.2. Therefore, the computer vision system should be active during long-range missions as long as the missions start and end on the landing platform.

5.1.4 Hover and Land

A Hover and Land mission is performed by the physical AR.Drone 2.0 in an indoor environment. The controller is activated after manually controlling the AR.Drone to a position about 1.5m above the landing platform. After 75 seconds of autonomous hovering, Mission Control initiates an automatic landing on the landing platform.

The objective of this experiment is to

- determine whether the pose estimation algorithm is sufficiently accurate for use in real-time control of the quadcopter

- test the capability of the control system, and whether the quadcopter can attain stable hovering
- test the capability of the automatic landing procedure

This experiment is also performed with the landing platform mounted on the DNV ReVolt.

5.1.5 RealMission

The quadcopter performs an experiment called RealMission in which the quadcopter executes a complete autonomous mission, including takeoff, hovering and landing. The mission is performed in an indoor environment with the drone starting on the landing platform.

This experiment aims to test whether the quadcopter system is capable of performing an autonomous mission, providing a proof of concept of the mission capabilities of the quadcopter system.

5.1.6 Accuracy Measures and Error Metrics

Some accuracy measures and error metrics are defined to assess the system's performance appropriately.

Simulation Accuracy Measures

Ground truth measurements are available for quadcopter simulations. Because of this it is possible to calculate the measurement error of the state estimation algorithm.

$$\mathbf{e}_k = \mathbf{x}_k^{\mathrm{gt}} - \hat{\mathbf{x}}_k \tag{5.1}$$

The estimation error of position can be represented by the Euclidean distance between the ground truth position and the estimated position, which is calculated by

$$\mathbf{e}_{p,k} := \sqrt{(\hat{x}_k - x_k^{gt})^2 + (\hat{y}_k - y_k^{gt})^2 + (\hat{z}_k - z_k^{gt})^2}. \tag{5.2}$$

The pose estimation algorithm proposed in this thesis does not estimate roll or pitch other than reading the on-board estimates from the AR.Drone. Accordingly, the estimates of orientation discussed henceforth are limited to estimates of quadcopter yaw. The orientation error is defined to be

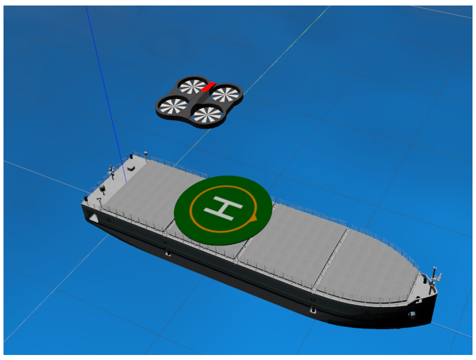
$$\mathbf{e}_{o,k} := \psi_k^{gt} - \hat{\psi}_k \tag{5.3}$$

Round-mean-square error (RMSE) is used for measuring the difference between the estimate and the ground truth over the complete mission. RMSE is calculated by

$$\mathbf{e}_{\mathrm{RMSE}} := \sqrt{\frac{1}{n}\sum_{k=0}^{n} \mathbf{e}_k^\top \mathbf{e}_k} \tag{5.4}$$

Experiment Accuracy Measures

Ground truth measurements are not available when performing experiments using the physical quadcopter. Therefore it is not possible to calculate the estimation error. Other success measures are used instead for determining mission success.


(A) Quadcopter successfully hovering above the landing platform.

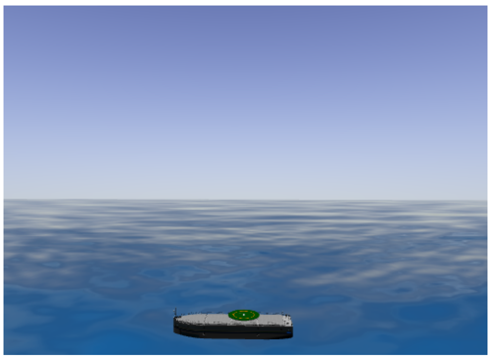

(B) Photo captured using the quadcopter's front-facing camera during *PhotoMission*.

FIGURE 5.1: Images from the simulations

Sensor	RMSE (m)	Mean (m)	Std. dev (m)
DNN CV	0.04207	0.0137	0.0200
TCV	0.0409	-0.0167	0.0166
GPS	0.0887	-0.0049	0.0510
IMU	0.2196	-0.0624	0.1144
Barometer	0.0395	0.03948	0.0017
Final Estimate	0.0396	0.0083	0.0213

TABLE 5.2: Euclidean error RMSE, as well as mean and standard deviation position error. Data captured during simulation of *HoverMission*, and is expressed in metres.

The measurement criteria for these experiments are defined to be:

- test completion without crashes
- oscillatory behaviour
- whether pose estimates correspond to observed quadcopter response.

These are determined by visual inspection of the quadcopter during missions and inspection of the filtered pose estimates after mission completion.

5.2 Simulation Results

5.2.1 HoverMission

An image of the drone hovering above the landing platform located on a model of the DNV ReVolt ship is seen in Figure 5.1a. The ground truth pose and pose estimates from a 40-second *HoverMission* are displayed in Figure 5.2, where the estimated pose and the ground truth pose are expressed in body frame coordinates.

In order to determine the accuracy- and precision of the pose estimation system, one may contrast the estimate versus the ground truth in Figure 5.2. A table of RMSE, mean, and standard deviation of Euclidean distance error can be seen in Table 5.2 and likewise for yaw error in Table 5.3. The Euclidean distance errors for the individual sensors are displayed in Figure 5.3

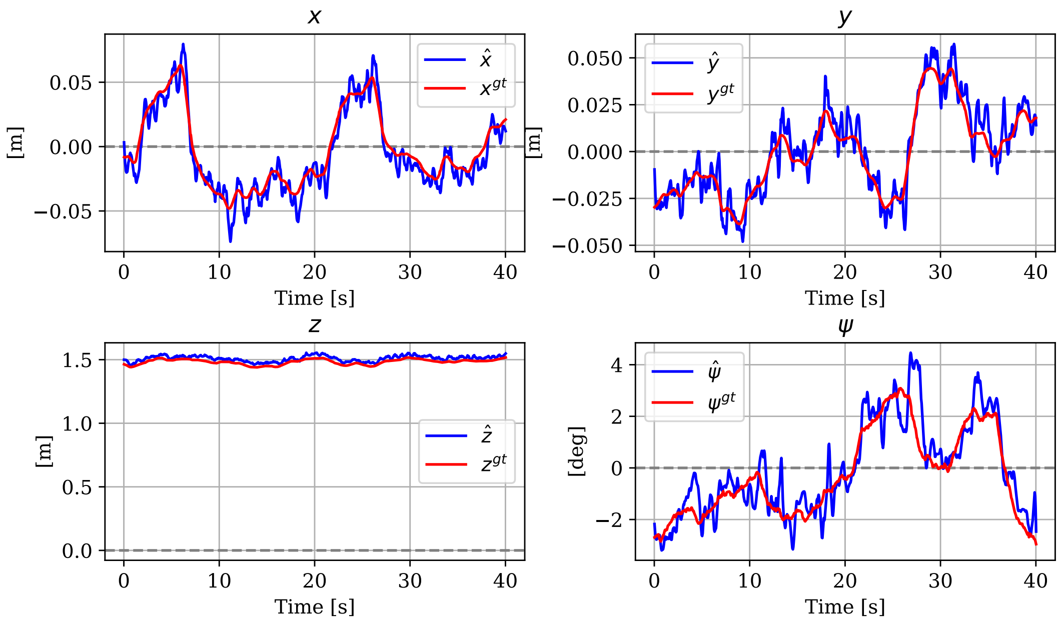
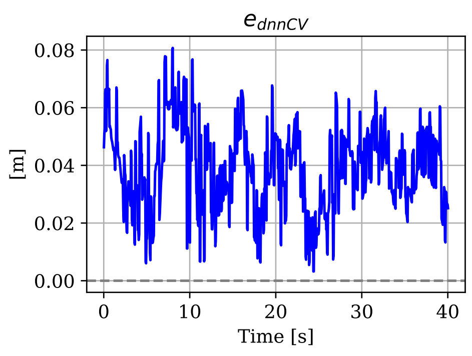


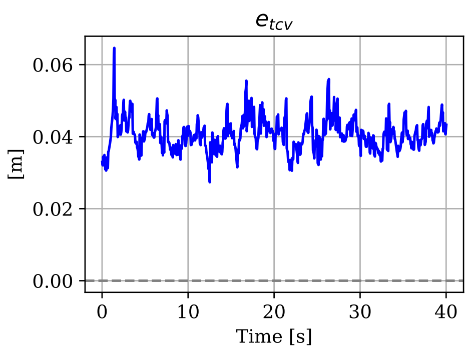
FIGURE 5.2: The filtered estimates versus ground truth from simulating *HoverMission*, expressed in body frame coordinates.

Estimate	**RMSE (deg)**	**Mean (deg)**	**Std. dev (deg)**
DNN CV	1.3049	-0.7123	1.0933
TCV	1.6748	0.2843	1.6505
IMU	0.7260	-0.7257	0.0227
Final Estimate	0.9414	-0.2074	0.9182

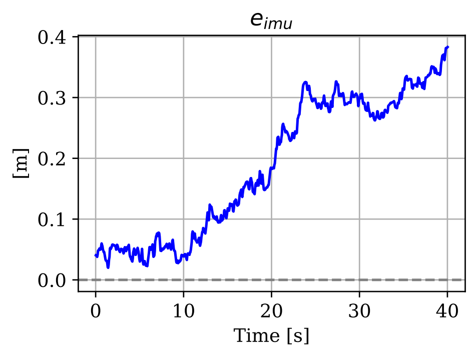
TABLE 5.3: Yaw estimate errors RMSE, Mean and Std. deviation for different sensors and final estimate, expressed in degrees. Data captured during simulation of *HoverMission*.



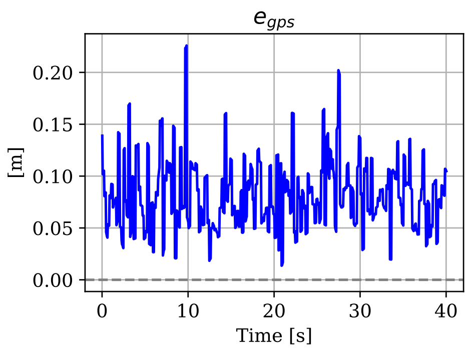
(A) DNN CV errors



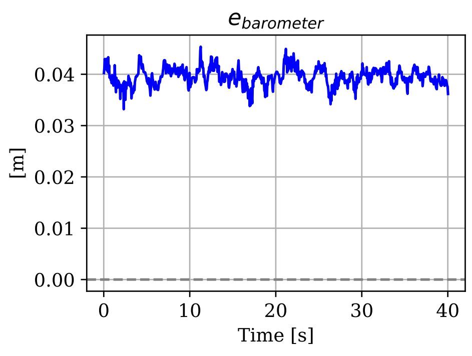
(B) TCV error



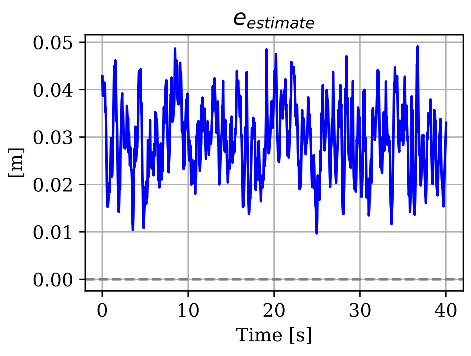
(C) IMU error



(D) GPS error



(E) Barometer error



(F) Filtered estimate error

FIGURE 5.3: Euclidean estimation errors while simulating *HoverMission*.

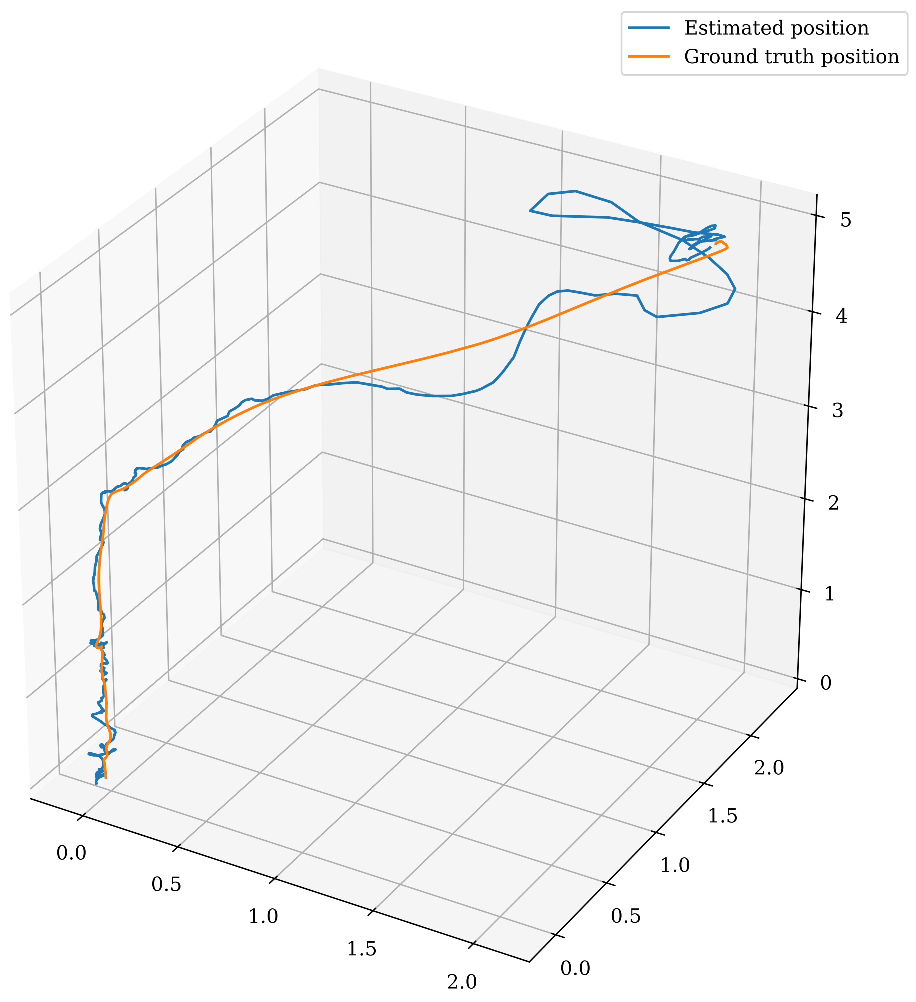


FIGURE 5.4: Quadcopter landing trajectory while simulating *LandingMission*.

5.2.2 LandingMission

The *LandingMission* is simulated ten times to test the system's robustness and reliability, and the quadcopter managed to land successfully on the landing platform all ten times. The landing trajectory for a *LandingMission* that was completed successfully is seen in Figure 5.4, which is the test case that is further discussed and analyzed in this thesis.

Ground truth state versus state estimates during the *LandingMission* can be seen in Figure 5.5.

5.2.3 PhotoMission

The ground truth flight trajectory of a successful *PhotoMission*, as well as the estimated flight trajectory, can be seen in Figure 5.6. The quadcopter captured a photography during the *PhotoMission*, which is displayed in Figure 5.1b. By studying the ground truth trajectory in Figure 5.6 one can see that the quadcopter managed to complete the mission, managing to move to the desired setpoints as well as returning to the landing platform with automatic landing.

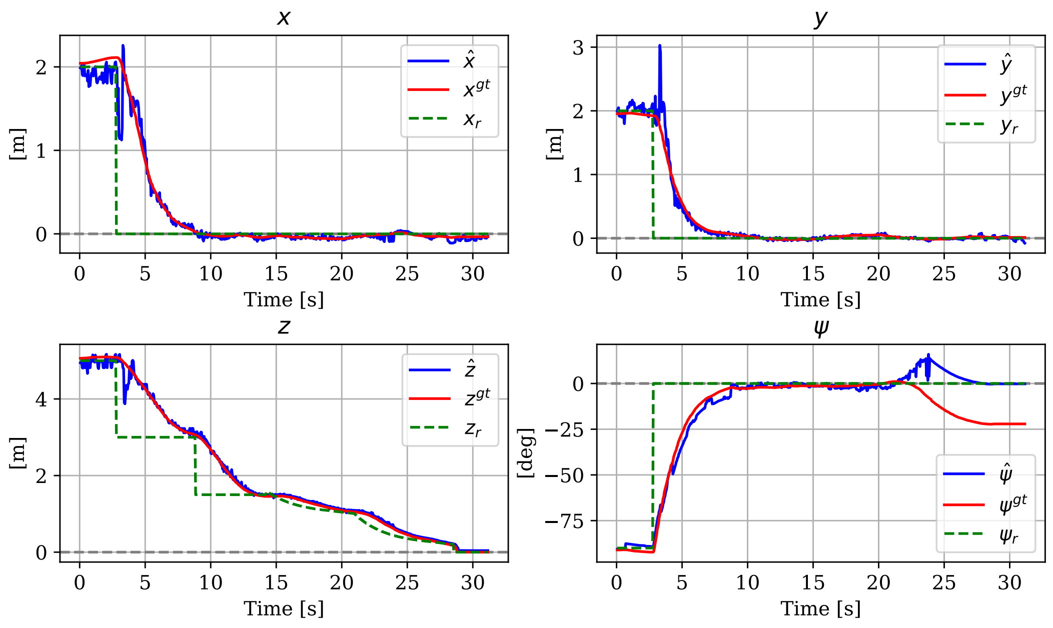


FIGURE 5.5: State estimates while simulating *LandingMission*.

A simulation of another mission is displayed in Figure 5.7, where the quadcopter performs a hexagonal flight maneuver in order to demonstrate that the Mission Control FSM can be used to create many different missions.

5.3 Experimental Results

The experiments are performed in both an indoor environment. An image of the quadcopter hovering in the indoor environment is displayed in Figure 5.9a. The experiments are also performed with the landing platform mounted on the DNV ReVolt vessel in a warehouse environment which can be seen in Figure 5.11.

5.3.1 Indoor experiments

The physical AR.Drone is not equipped with a GPS. Thus, the system requires the landing platform to be in the camera view for position updates, other than altitude measurements from the barometer.

The system used in experiments is altered compared to the system used in simulations, addressed in Section 4.7, in order to maximize performance. Even though the systems used for experimental testing and simulation testing are different, the results can arguably be compared and contrasted. The results can be compared because the alterations removed features such as yaw control, IMU integration and GPS, while not adding any other features. Therefore the systems can be compared on the parts that remain the same.

A *Hover and Land* mission is performed in an indoor environment. During the *Hover and Land* mission the quadcopter is applied a setpoint 1.5m above the center of the landing platform, at $x = 0$m, $y = 0$m and $z = 1.5$m. After 75 seconds of hovering the drone is to perform an automatic landing on the landing platform. The estimates from the experiment, ending with an automatic landing, can be seen in Figure 5.8. An image of the quadcopter hovering during the experiment can be seen

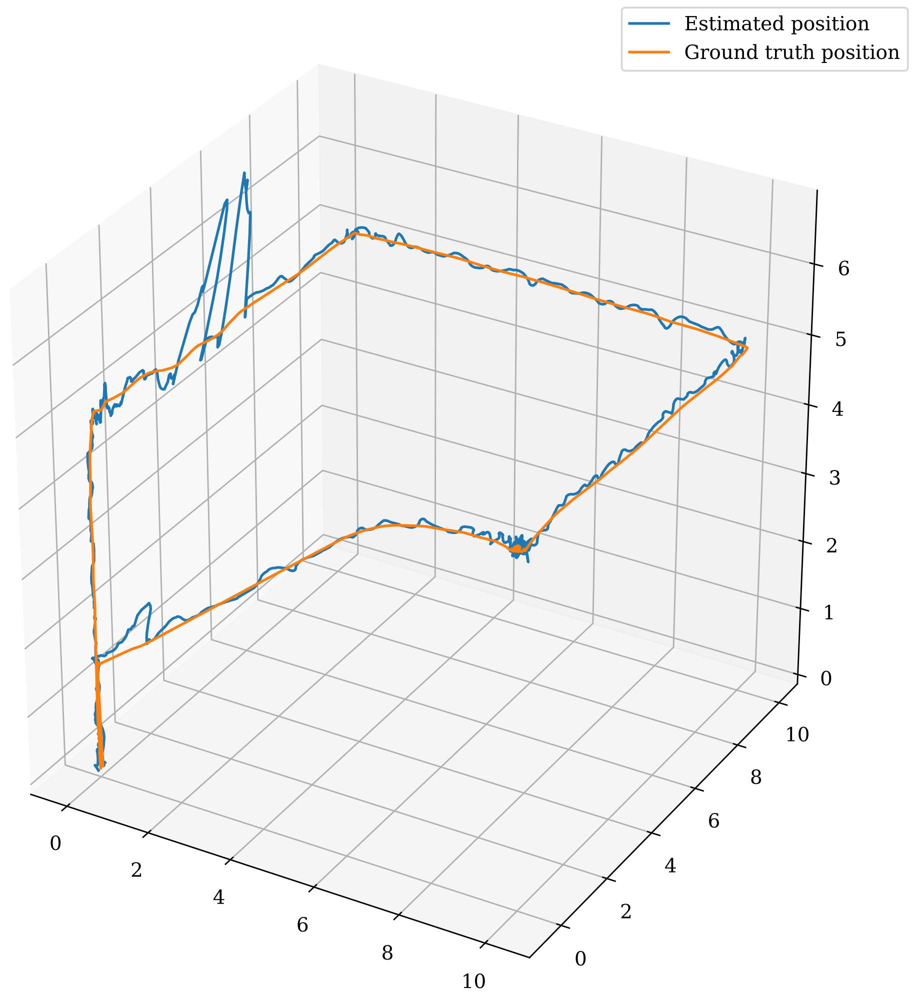


FIGURE 5.6: Simulation of *PhotoMission*. Successful completion of mission with landing on platform. Large spikes in estimates appear when landing platform is moving out and in of camera view.

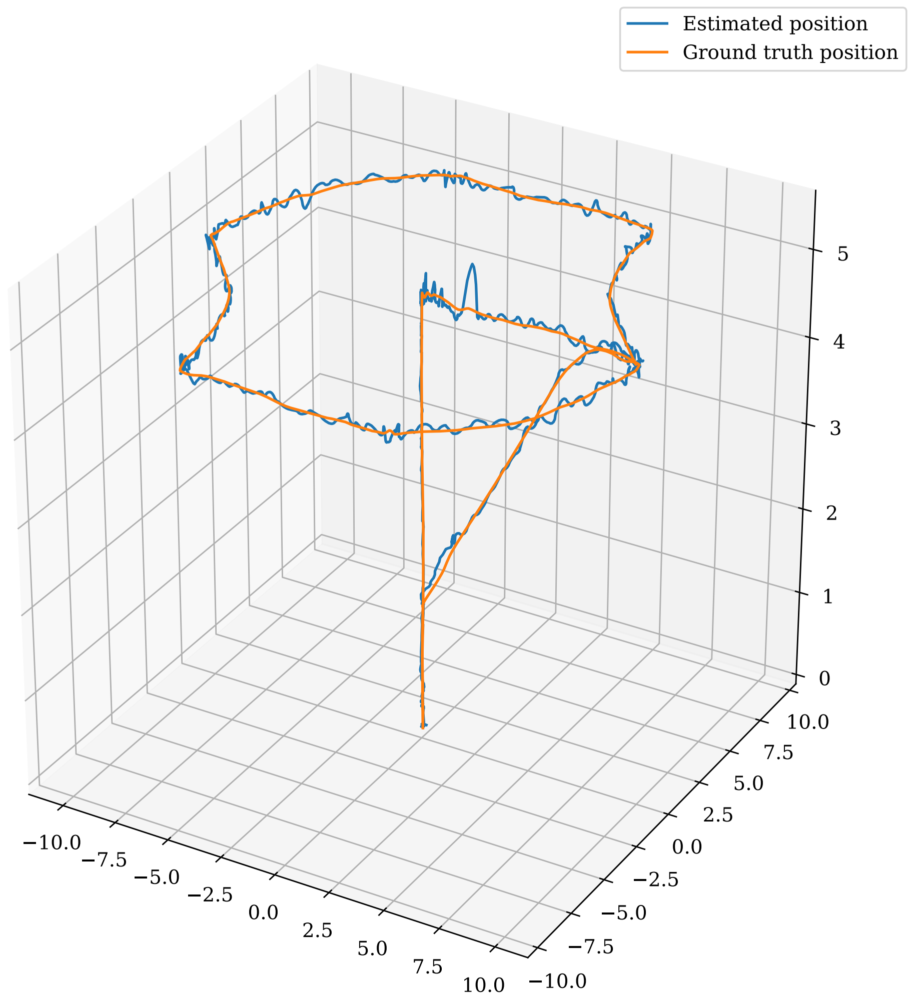


FIGURE 5.7: The quadcopter system is able to perform complex flight maneuvers autonomously in simulations, here demonstrated by completing a hexagonal flight pattern in 3d space.

in Figure 5.9a and an image of the quadcopter successfully landed on the landing platform can be seen in Figure 5.9b.

An *RealMission* experiment where the quadcopter takes off, hovers and lands is successfully completed, and the results of this mission can be seen in Figure 5.10. The quadcopter is initialized on the landing platform, taking off and trying to hover centered over the landing platform at an setpoint of 1.5m. The drone hovers for at minimum of 5 seconds before initiating automatic landing when it is sufficiently close to the setpoint.

5.3.2 Experiments On The DNV ReVolt

A *Hover and Land* mission is also performed on the DNV ReVolt autonomous marine vessel. The ReVolt vessel is mounted on a car trailer. Images of the landing platform mounted on the deck of ReVolt can be seen in Figure 5.11.

The experiment is performed in an open space in a warehouse that is partially shielded from the wind. The autonomous quadcopter system manages to perform hovering over the landing platform and automatic landing. The automatic landing is initiated by Mission Control when the position estimates are sufficiently close to the desired reference point over the landing platform. The quadcopter landed on the landing platform, deeming the mission a success.

The position estimate of the quadcopter during the mission is presented in Figure 5.12. These estimates correspond well with the observed flight trajectory of the quadcopter.

This experiment is also attempted in an outdoor environment by moving the ReVolt outside. Slight wind gusts caused the quadcopter to repeatedly swerve out of control and lose track of the landing platform. Consequently, the mission is unsuccessful. In sum, it is seen that the quadcopter does not manage either stable hovering or automatic landing in slightly windy conditions.

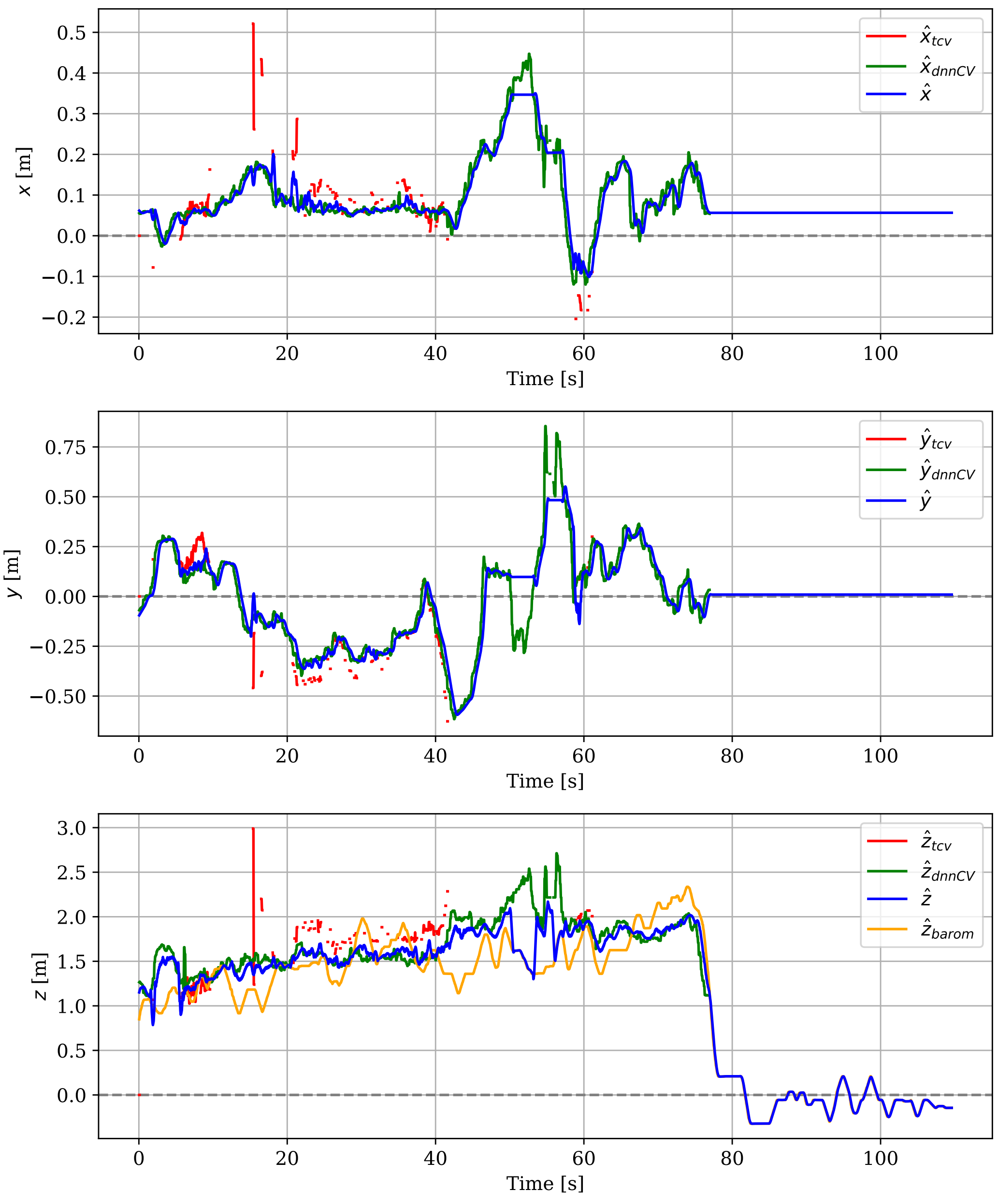
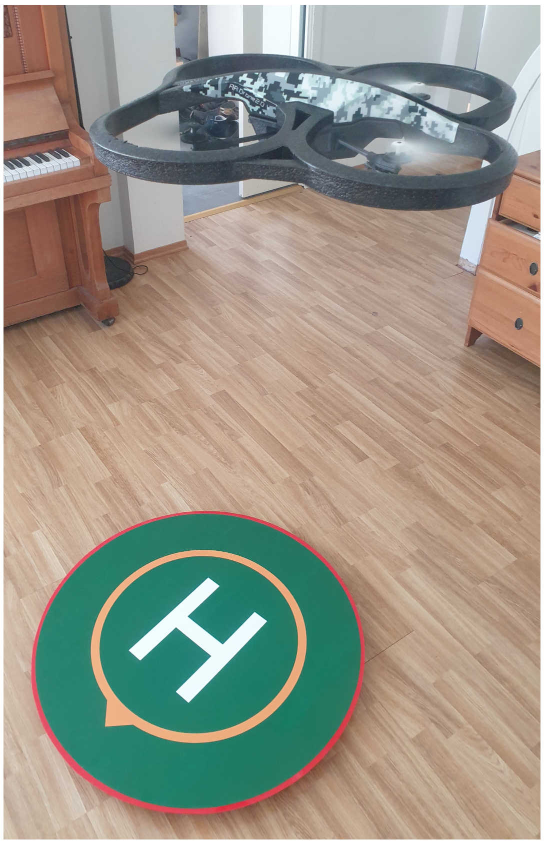


FIGURE 5.8: Estimates during indoor *Hover and Land* mission


(A) The quadcopter hovering indoors over the landing platform

(B) Quadcopter successfully landed after *Hover and Land*

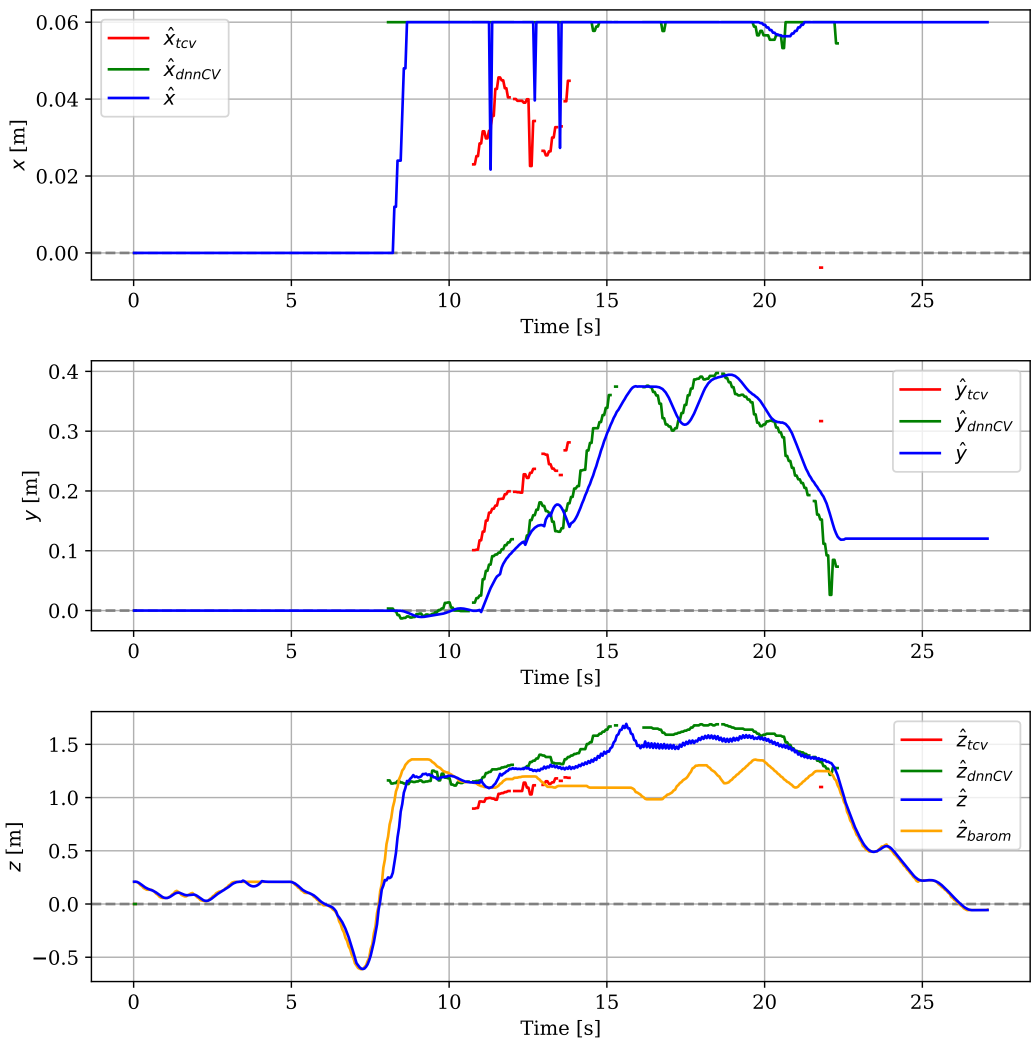


FIGURE 5.10: Estimates from indoor *RealMission*, where quadcopter performs autonomous takeoff, hovering and landing on the landing platform

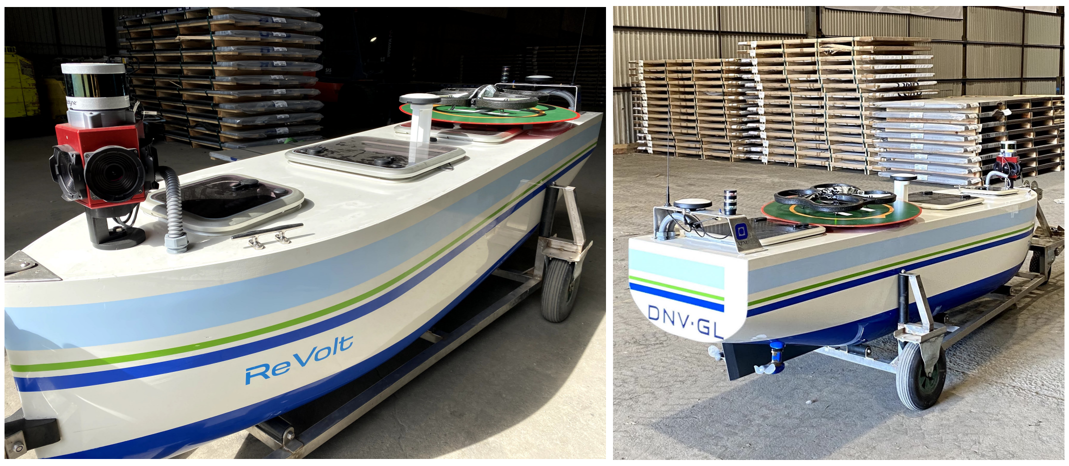


FIGURE 5.11: The DNV ReVolt with the landing platform mounted on the deck

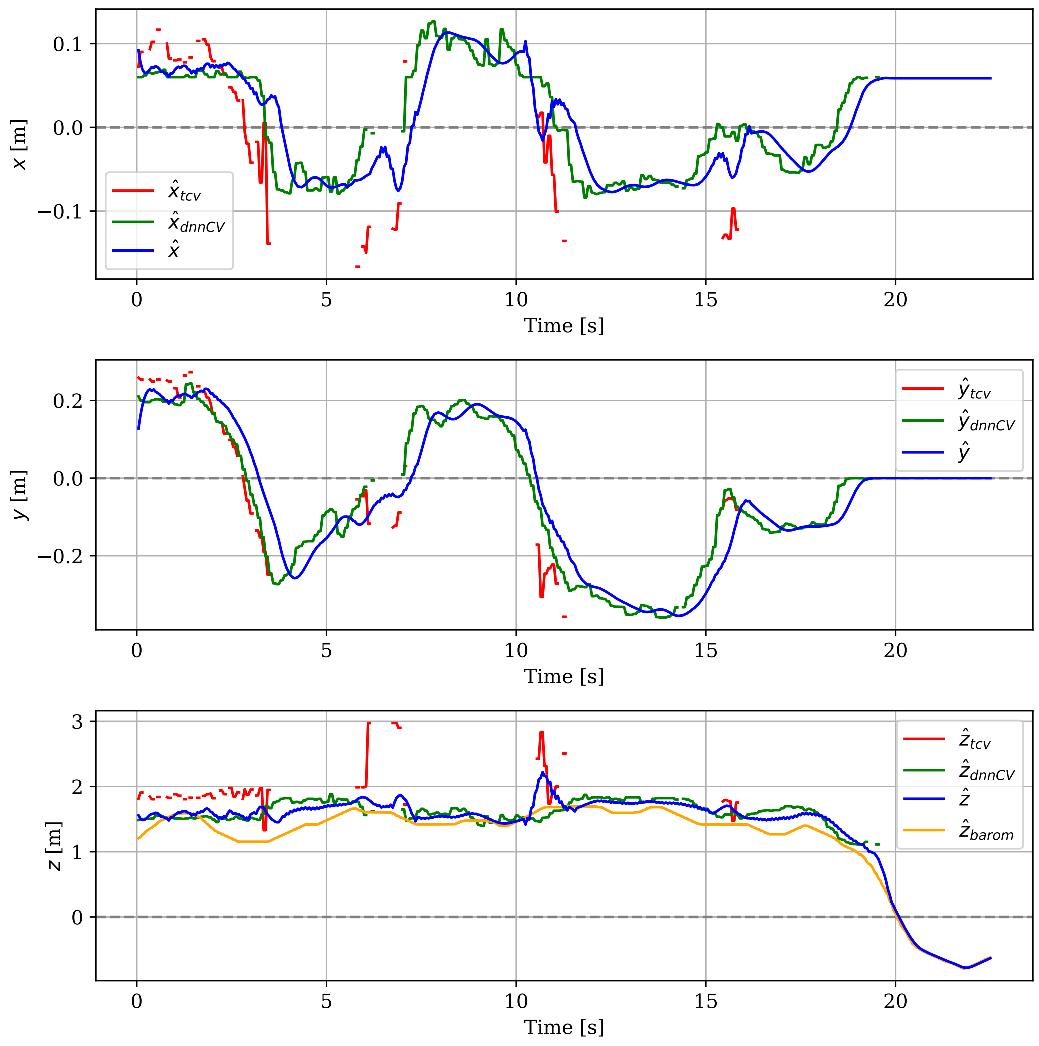


FIGURE 5.12: Position estimates during *Hover and Land* on the DNV ReVolt

Chapter 6

Discussion

Several results were presented in the previous chapter and are discussed in this chapter. The simulation results are first discussed, where the capacity of the system is tested in an environment with few limitations. Subsequently, the experimental results are discussed and contrasted with the simulation results.

6.1 Discussion of Simulation Results

Results from three different simulation missions were presented in Chapter 5, and are discussed based on the individual missions that were performed. The results from *HoverMission* provide the basis for a discussion of the pose estimation and stability. The *LandingMission* provide the basis for a discussion of the landing capability of the system. Results from *PhotoMission* provide the basis for a discussion of pose estimates without computer vision and the system's capacity for long-range missions.

6.1.1 HoverMission

By analyzing the filtered estimate in Figure 5.2 and contrasting that to the ground truth pose one can see that:

- The estimate is close to the ground truth in both the x- and y-axis, oscillating around the ground truth line, with a maximum estimation error of about 4cm and 2.5cm, respectively. The error in the x- and y-axis seem to be zero mean.
- The estimate is close to the ground truth in both the x- and y-axis, oscillating around the ground truth line, with a maximum estimation error of about 4cm and 2.5cm, respectively. The error in the x- and y-axis seem to be zero mean.
- The altitude estimate is slightly above the ground truth altitude, seeming to correspond to the 4cm bias of the barometer
- That the yaw estimate is close to the ground truth line

By examining the ground truth pose of the quadcopter in Figure 5.2 one can analyze the ability of the quadcopter system to hover at the desired setpoint:

- the quadcopter manages to hover in the air the whole duration of the test. The stability of the system has been further demonstrated by successfully performing a 30-minute *HoverMission* without any crashes or significant oscillations occurring.
- there are some oscillations in the x- and y-axis, with a maximum amplitude of just below 7.5 cm, seen in the x-axis of fig. 5.2 about 6 seconds into the mission.

- there are some oscillations in the z- axis, with an amplitude of about 5cm.
- the ground truth altitude is slightly below the reference point of 1.5m. This is due to an error in altitude estimation and not due to faulty control.
- there are some oscillations in yaw, with a maximum amplitude of about 3 degrees.

In sum, it is seen that the quadcopter system is able to hover close to the reference point, having some oscillations in all axes as well as a constant bias in altitude.

Arguably, these oscillations are small, being a few centimeters from the desired reference point. Comparing the amplitudes of these oscillations to the scale of the quadcopter and landing platform, as well as being 1.5m over the landing platform, yields that the errors are sufficiently small to be of minor significance to the success of the test.

One of the assumptions for the Kalman Filter is that the sensor noise is white, having zero mean and a constant power spectral density. The middle column of Table 5.2 and Table 5.3 represents the mean sensor value for all the sensors during the mission. The assumption of zero mean should be considered fulfilled for the DNN CV error, TCV error and GPS error, as the mean Euclidean errors are close to zero for these sensors.

The mean barometer error is 4cm (5.2), and the standard deviation of the measurements is minimal, being 0.002 cm. By inspecting the plot for the barometer error Figure 5.3e one can see that it is oscillating about a point at about 4cm, suggesting that the barometer, within the time frame of the experiment, suffers from a stable bias.

A non-zero mean results in a non-optimal Kalman Filter. A non-optimal KF means that the sensor updates are not statistically optimal to minimize estimation error covariance.

The IMU predictions have significant mean error and RMSE, as well as standard deviation. This data is based on unaided IMU predictions. The strength of IMU predictions is that they are high frequency and able to provide a smooth and accurate estimate in combination with other sensors. The problem with predictions is that prediction errors grow with time. The IMU predictions, with no updates for 40 seconds will have a non-zero mean and have a high RMSE. This is verified in Table 4.1. Nevertheless, the IMU predictions provide more certainty to the model when combined with sensor updates than sensor updates alone.

The mean GPS error is tiny, below 0.5cm. The GPS data is created as a normal distribution of the previously known ground truth drone data, with a std. distribution of 5cm, and therefore it is expected to be close to zero-mean.

In sum, the *HoverMission* demonstrates that the pose estimation system is sufficiently accurate for quadcopter missions in the simulator. All the sensors provide close to accurate measurements. The KF fuses the sensor measurements to produce one estimate that is more accurate and precise. The mean of the final estimate is very close to zero, at 0.83cm with a standard deviation of 2.13cm.

The control system is well-tuned and able to control the quadcopter to the desired position using the pose estimate as feedback. Despite a few low-amplitude oscillations, the quadcopter control is deemed sufficiently stable and accurate.

6.1.2 LandingMission

There are quite large pose estimate error spikes at the start of the mission. The estimated position jumps from close to ground truth, to a value about one meter above and below the ground truth, in a short time span and large positive and negative prediction errors, called "spikes" in the prediction error graph Figure 5.5. The $\hat{x}$ spikes down roughly 1 meter and $\hat{y}$ spikes up about 1 meter.

These spikes occur when the quadcopter performs sudden pitch and roll maneuvers. The quadcopter performs these maneuvers in order to gain speed to move towards the landing platform. Pitch and roll maneuvers cause the affect the camera angle, causing a perception of the landing platform moving in the camera image. Thus, the estimates may change due to pitch and roll movements because the CV algorithms estimate position based on the location of the platform in the camera frame. One of the assumptions for the DNN CV algorithm, presented in Section 4.3, is that the quadcopter pitch and roll are zero. Therefore, pitch and roll maneuvers makes CV estimate errors larger since the assumption of the pose estimates is violated.

In order to reduce the pitch and roll maneuvers one could apply velocity references as smoothly increasing references instead of the currently implemented step-functions. Smoothly increasing velocity references would cause the internal controller of the AR.Drone to increase the velocity gradually, resulting in small pitch and roll maneuvers. However, the pitch and roll maneuvers would be drawn out in time, such that the estimation errors caused by these maneuvers would be drawn out in time as well. Therefore, it is not certain that such a change would provide a better flight response.

The quadcopter was able to move towards the landing platform even though x and y estimates were off during the start of the landing mission. The estimation error spikes were brief, and the inertia of the quadcopter is sufficiently high so that the drone flight path was not affected before had reliable data to work with.

The automatic landing was completed in a controlled manner. The duration of the whole mission was just short of 30 seconds.

The simulated *LandingMission* demonstrates the robustness of the system to provide accurate pose estimates and navigate in a controlled manner in order for the quadcopter to execute an automatic landing without GPS in a simulation environment.

The simulation demonstrated a robustness in the system since the prediction error spikes did not seriously affect the drone flight characteristics.

6.1.3 PhotoMission

By analyzing the pose estimates during simulation of *PhotoMission* against the ground truth data, displayed in Figure 5.6, one can see that there are significant pose estimation spikes at two points during the mission. These deviations occur when the landing platform is moving in and out of view of the bottom facing camera. This happens when the quadcopter is flying away from the landing platform during the beginning of the mission, and when the quadcopter is coming back to the landing platform towards the end of the mission.

These spikes occur because there is a transition face where only a small fraction of the landing platform is seen in the camera frame when the landing platform is moving out and in the camera view. Therefore, the bounding box that is subsequently drawn around the landing platform by the DNN CV algorithm is small. The DNN CV algorithm uses the size of this bounding box to estimate the size of the entire landing platform, thus estimating the size of the landing platform to be very small in the camera frame. This results in the DNN CV algorithm estimating that the quadcopter is located high above the landing platform, producing pose estimates that are way off in altitude. This effect can be seen both in the case when the

LP is entering and exiting the camera frame and in both Figure 5.6 and Figure 5.7. Further refinement of the DNN CV pose estimation algorithm is required in order to fix such errors.

GPS measurements compensate for these errors, pulling estimates down and more close to the ground truth pose, demonstrating that a combination of sensors is an effective way of compensating for sensor failure. As a result, the quadcopter is able to continue moving in the correct direction until the landing platform is completely outside the camera view.

Comparing the ground truth position of the quadcopter with the estimated position in Figure 5.6 one can see that the estimate is oscillating around the ground truth line. This is due to the system is being updated using GPS measurements, which have a standard deviation of 5cm in three axis. The GPS measurements are therefore quite imprecise. A Kalman Filter tuning with less uncertainty in predictions would have reduced the oscillatory effect on the estimate. However, the GPS measurements are infrequent, at 2 Hz. The predictions have become quite uncertain after 0.5s, so the GPS measurements update the estimate to a large degree. Nevertheless, the current tuning provides a good quadcopter response as seen by the ground truth line.

This mission is built as a set of pre-defined flight commands, which the quadcopter performs. The same Mission Control framework can be used to construct different missions. Another long-range mission in which the quadcopter completes a hexagonal flight pattern demonstrates this. The flight trajectory for this mission can be seen in Figure 5.7.

6.2 Discussion of Experimental Results

Two *Hover and Land* missions are performed, one in an indoor environment and the other with the landing platform mounted on the DNV ReVolt. *RealMission* is performed in an indoor environment. Yaw control is disregarded for these experiments, the IMU measurements are discarded, and the physical drone is not equipped with a GPS.

These results provide the base for discussing the limitations with the physical quadcopter, and the challenges with experiments compared to simulations.

6.2.1 Hover and Land

In Figure 5.8 one can see that the quadcopter manages to hover over the landing platform for 75 seconds before it performs an automatic landing procedure. The quadcopter lands on the landing platform, with the final position seen in Figure 5.9b.

The quadcopter response during the experimental *Hover and Land* suffer from oscillations than the response observed in the simulated *HoverMission*. In Figure 5.8 one can see that the quadcopter experienced oscillations in x and y that reached a maximum amplitude of about 0.5m. The estimated position correlates well with visual observation of the quadcopter during the mission.

The amplitude of the oscillations are higher than acceptable, significantly affecting the stability and execution of the mission.

The cause for this deviation from the simulation in the practical experiment must be analysed and identified. Two possible causal factors may be sensor noise, or more likely, the control delay with the AR.Drone.

Control Delay and Drone Inertia

There is a significant delay in the control loop between the ground station computer and the AR.Drone. The communication delay is reported to be 130ms and the DNN CV has a 38-45ms processing delay. The sum of the two delays results in a control delay of about 170ms.

In addition to the control delay, there is a significant amount of delay before onboard controller manages to reach the velocities specified by the PID-controller. The drone has a significant amount of inertia and is slow to start- and stop moving. The motors spinning the propellers also have a control delay, where they require time to change the rate of spin. In sum, these delays are making the drone react slowly to velocity commands.

The drone inertia is deemed during testing to be a cause of instability. The quadcopter is prone to move past the landing platform, which causes oscillations.

Movement slicing is important for dealing with the control delay and drone inertia. Movement slicing was used with moving slots and halting slots of 2- and 2 seconds and reduced the problem of the drone overshooting the center of the landing platform.

DNN CV Pose Estimator

The DNN CV algorithm is able to provide frequent position estimates over the course of the mission. The DNN CV is the most significant sensor in this experiment. The physical quadcopter is not equipped with a GPS, the IMU is discarded, and the TCV algorithm does not produce many estimates. Thus, the pose estimate is mostly based on DNN CV measurements.

The pose estimate, being based mostly from DNN CV measurements, is sufficiently accurate to enable autonomous hovering and landing. These results signify that the DNN CV measurements are sufficiently accurate to be used for quadcopter control.

During the *Hover and Land* mission, about 50 seconds into the test, some DNN CV measurements are discarded. This is seen by the flat estimation line for x and y in Figure 5.8. The measurements were discarded in the pre-filter due to estimating the quadcopter to be over higher than the maximum allowed limit, set to be the altitude of the ceiling at 2.20m. When the measurements are discarded the Kalman Filter does not update the state estimate using the measurement because the measurements are deemed inaccurate. Thus, the discarding filter is demonstrated to be capable of discarding inaccurate measurements. The DNN CV algorithm recovers a few seconds later, and the KF updates the estimate accordingly. Therefore, it is seen that the DNN CV fails occasionally, however on the course of the whole mission, it performed well enough for the mission to succeed.

When the quadcopter is close to the landing platform, below 1m in altitude, the landing platform covers its entire vertical camera view and much of the horizontal. Therefore, the algorithm fails to accurately predict the position in the x-axis or the quadcopter altitude if the quadcopter is close to the landing platform. Thus, the DNN CV should only be used in the ranges of operation where it is effective. DNN CV measurements are discarded when the quadcopter is below 1m.

TCV Pose Estimator

The TCV algorithm is unable to produce many estimates. The estimates from the TCV algorithm are displayed in Figure 5.8 where the scattered red lines and dots are the points in time where the TCV algorithm managed to produce estimates.

The TCV algorithm, proposed by Sundvoll [1], was created for the simulation environment and subsequently adapted for use in the real world, and may not be well-developed for real-world conditions.

The physical landing platform is similar to the simulated landing platform in simulations, although there are some differences. The TCV algorithm performs a color segmentation at the start of the inference on the image. Suppose the specified colors for the color segmentation do not match the color that the landing platform appears to have in the camera view. In that case, the algorithm will not provide any estimates. Therefore it is essential to have an appropriate tuning suitable for the visual conditions where the TCV algorithm is used.

The TCV algorithm is based on detecting specific features and points of the landing platform, namely the ellipse, corners on the white H and the corners of the Arrow. Admittedly, there is quite a lot of motion blur in the camera feed from the AR.Drone when the drone is moving. Motion blur may make the detection of such features more challenging as the features are blurred and lines and segments are less distinct. Using a quadcopter with a higher resolution camera and a camera gimbal could solve this problem.

Motion blur may reduce the usefulness of a TCV algorithm based on detecting details in the camera image. Nevertheless, the TCV algorithm proved to be more precise than the DNN CV algorithm in simulations, demonstrating that it can be precise.

The TCV algorithm produced misdetections at 17 seconds into the test, estimating the quadcopter to be 3m over the landing platform. Further analysis of the TCV algorithm is required to find the cause of these misdetections.

Further development of the TCV algorithm is required to be able to replicate the promising results in simulations.

Barometric Altitude Measurements

By contrasting the barometric altitude estimates in Figure 5.8 with the altitude estimates by the DNN CV algorithm, it seems that the estimates differ slightly. The difference between the altitude estimates from the two algorithms are generally less than 0.5m. The pressure sensor generally measures the altitude to be less than measured by the DNN CV algorithm. Visual observations confirmed that the quadcopter was hovering at 1.5m to 1.8m, thus in the range that is mostly measured by the DNN CV algorithm.

The barometric altitude sensor is not precise, varying over the course of the mission. The estimates do not always match the visually observed altitude of the quadcopter. The pressure at sea level is typically about 100,000 pascal, and decreasing with about 11.3 pascal for every meter in altitude. Every meter in altitude is therefore calculated by a 0.01% change in pressure. Such a small change may be difficult for a low-cost sensor such as the pressure sensor on the AR.Drone 2.0 to measure precisely. The Kalman Filter pressure sensor noise parameter is high compared to that of the DNN CV algorithm. Thus, the filtered estimate is closer to the measurement from the DNN CV algorithm than the measurement barometric pressure sensor.

The pressure measurements are filtered using an MMA filter before they update the Kalman Filter. The averaging filter size is 200, and the median filter size is 200, resulting in a total filter size of 401. That means that the last 401 pressure measurements are stored in the filter. The middle 200 measurements in the filter are averaged, which results in one pressure output. Such a large filter is necessary in order to increase precision. The pressure sensor sends measurements at a rate of 200 Hz. The filter, therefore, contains pressure data from the last 2 seconds. Such filtering introduces a time delay in the measurements. Swift altitude changes are therefore subject to a delay before being measured by the filtered barometer. Nevertheless, this filter is necessary for the noisy pressure sensor, resulting in an improved sensor performance.

When the quadcopter is sufficiently low, the CV measurements are not accurate and are therefore discarded. The cutoff threshold for the DNN CV algorithm is 1m, and the cutoff threshold is 0.4m for the TCV algorithm. The TCV algorithm is not able to produce any significant amount of estimates during this experiment, and therefore, below 1m the only altitude measurements available are barometric measurements. The barometer, therefore, is essential for the quadcopter takeoff and landing.

At the end of the mission in Figure 5.8 the altitude measurements correspond with visual observations of the quadcopter flight, estimating the quadcopter altitude to decrease during landing, ending around zero. The estimates are slightly dipping below zero when the quadcopter reaches the ground, and the propellers are still spinning.

A barometer is not fit for the purpose of measuring small altitude changes. The measured air pressure is dependent on wind, temperature, the pressure created by the quadcopter propellers and quadcopter velocity. Small altitude changes result in tiny changes in air pressure, and it might therefore be within the measurement uncertainty of the barometer. A sonar is a sensor more fit for such an application as low-altitude autonomous quadcopter flight. A barometer may be a good sensor in combination with others for estimating altitude during high-altitude flight, as sonars and computer vision sensors may be less reliable at high altitudes.

6.2.2 RealMission

The x estimate in Figure 5.10 is flat and mostly stationary at $\hat{x} = 0.06m$.

The quadcopter is not stationary in x-axis, as the estimate suggests. The estimate is not changing since at the quadcopter's altitude the camera was unable to view the entire landing platform. Thus DNN CV algorithm was inaccurate. The DNN CV algorithm estimates quadcopter position based on the location of a bounding box surrounding the landing platform in the camera frame. Since the landing platform is covering the entire camera frame in the x-axis, the quadcopter is unable to estimate the position accurately.

The altitude estimate of the DNN CV is also faulty, estimating a higher than accurate altitude. From visual observations during the mission, it was observed that the drone was hovering around 1m above the landing platform, which corresponds well to the barometer measurements seen in Figure 5.10. The altitude estimates of the DNN CV algorithm are not accurate unless the quadcopter is sufficiently high above the landing platform.

The barometer accurately measured the altitude to be about 1m. However, the Kalman Filter trusts the DNN CV estimates more than the barometer measurements because they are tuned to be less noisy. Thus, the quadcopter did not move higher up because the filtered estimate followed the faulty DNN CV measurements. This displays a fault in the system and requires further investigation.

The system may require tuning which makes the system less biased to the DNN CV sensor. The altitude limit for when to use DNN CV measurements could be increased since it does not accurately measure quadcopter position when close to the landing platform.

During the start of the mission, one can see that the barometric measurements and the filtered altitude estimate dip below zero. Such negative altitude measurements have been observed repeatedly when the propellers are spinning while the quadcopter is on the ground. The spinning propellers increase the atmospheric pressure, reflected in the negative altitude measurements. During takeoff, the propellers start to spin for a few seconds before reaching a sufficient speed for flying.

During both takeoff and landing, the quadcopter is too close to the landing platform for the DNN CV algorithm to be accurate. Therefore the altitude estimates are dependant solely on the barometer data. This can be seen by the fact that the yellow line representing barometric data perfectly overlaps the blue altitude estimate line in Figure 5.10 during the start and end of the mission.

The system demonstrates that it can perform complete autonomous missions, including takeoff and landing.

6.2.3 Experiments on the DNV ReVolt marine vessel

The *Hover and Land* experiment is performed on the DNV ReVolt vessel in an open warehouse environment. The quadcopter manages both hovering and landing on the platform mounted on the ReVolt vessel.

The pose estimates displayed in Figure 5.12 are similar to the results from the indoor *Hover and Land* experiment. The TCV algorithm is not able to produce estimates consistently. Pose estimates are based mostly on DNN CV measurements and barometer measurements.

The DNN CV algorithm occasionally classifies the whole ReVolt ship as the *H* because the ship is white and straight, similar to part of the *H*. The *H* is not used for pose estimation, so the incorrect classifications do not cause any problems. However, in order to fix the DNN CV classifying the ReVolt as *H* the RealSet dataset can be increased by adding labeled images of the landing platform mounted on the ReVolt. The DNN CV algorithm can thus be trained to detect the difference between the *H* and the ReVolt.

The success of the ReVolt experiment provides a foundation for future marine quadcopter missions. The quadcopter system demonstrated that it could land on a marine vessel. However, marine use-cases present some yet unmet challenges for this system. The current system is not designed for a moving landing platform and will have trouble hovering and landing over the platform if the ship was moving at any significant speed. The CV algorithms assume that the landing platform is oriented horizontally. This assumption will not hold if there are a significant amount of waves making the ship heave, roll and pitch. If the assumptions do not hold, then the CV pose estimations will not be accurate.

The current quadcopter system is not able to handle any significant amount of wind. Robustness to wind is required for marine operations, which often are exposed to severe amounts of wind and weather. Therefore, further development of the system is required before marine operations are possible.

6.3 Comparison Between Results From Simulations and Experiments

Simulation results demonstrated that the system is capable of fully autonomous long-range quadcopter missions. The quadcopter control system is well-tuned for an efficient response. The pose estimation algorithm is estimates pose accurately.

In the simulated environment, some assumptions hold true:

- There is no wind or any external disturbances.
- There is negligible communication delay between the quadcopter and the computer.
- Visual conditions are stable, with no glare or change in brightness, and there is negligible motion blur from the simulated camera.

These assumptions do not hold when performing experiments with the real quadcopter. There is, however, negligible wind in indoor experiments. Experiments performed outdoors may result in the quadcopter being subject to large amounts of wind. A communication delay negatively affects the controllability of the system [70]. Unstable- and challenging visual conditions, as well as significant motion blur, make the problem of detecting the landing platform using computer vision more challenging. Increased sensor noise is a limiting factor as well. These limitations negatively impact the performance of the quadcopter system. The simulations are therefore expected to provide superior results compared with the experiments.

The simulation missions tested the capacity of what the system could perform in an environment with few limitations. The results in simulations met the objective of the thesis, demonstrating to be a system capable of autonomous quadcopter missions.

Experiments tested the robustness of the system and the capacity of the system given the limitations of hardware. The experimental results with the AR.Drone 2.0 demonstrated that the system is robust enough to perform autonomous missions without human intervention. The system met the objective of the thesis, demonstrating to be capable of autonomous missions.

The simulations could be extended to include limitations such as wind, challenging visual conditions, sensor noise and communication delays. Further analysis of the different limitations could be used to determine the impact of the individual limitations. Such analysis is outside the scope of this thesis.

Arguably, the system applied to a modern state-of-the-art quadcopter with up-to-date sensors and communications may be subject to few of the limitations observed by experiments with the AR.Drone 2.0. The simulation environment, which also has less limitations, may therefore resemble experiments using a more up-to-date quadcopter that is equipped with more precise sensors and on-board processing. Simulation results may therefore be replicated in the real-world by applying the system to an up-to-date quadcopter.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The objective of this thesis was to work towards an autonomous quadcopter system, able to perform missions without human supervention. As a result, an autonomous quadcopter system was presented, and several simulations and experiments were performed to evaluate the system's performance.

The proposed quadcopter system was based on pose estimation by a combination of many sensors, including computer vision, IMU, barometric pressure data and GPS data. The pose estimates were input to the control system which consisted of a Mission Planner and a PID-controller.

The quadcopter system demonstrated in simulations to be able to achieve autonomous quadcopter missions, consequently meeting the aim of the thesis. The missions that were demonstrated were:

- stable hovering
- successful automatic landing
- successful autonomous long-range missions using GPS, capturing images and passing through specified setpoints.

The simulation environment was an environment with no external disturbances, little sensor noise and a low communication delay. Simulation results therefore demonstrated the capacity of the system to perform missions without significant limitations.

The physical experiments proved to be more challenging than simulations, with increased sensor noise, significant communication delay and external disturbances, which were limiting the capacity of the system. Nevertheless, the autonomous quadcopter system managed autonomous missions, fulfilling the original objective of creating a system able to perform basic autonomous missions without human intervention.

- successful hovering in an indoor environment
- successful automatic landing mission
- successful autonomous takeoff- hover and landing mission
- successful missions with the landing platform mounted on the DNV ReVolt.

If the quadcopter loses track of the landing platform, it has no GPS to guide it back to the landing platform. Equipping the quadcopter with a GPS is outside the scope of this thesis. Hence, the quadcopter was able to perform autonomous missions but lacks the robustness required for a reliable autonomous system. The lacking robustness is argued to be due to limitations with the quadcopter used for experiments, and an up-to-date quadcopter could possibly replicate the results seen in simulations.

7.2 Future work

Though promising results demonstrated in simulations and experiments, there is admittedly a need for further development of this system before it is sufficiently robust to meet the requirements of present-day use-cases for autonomous quadcopters.

In order to develop this system further, one may investigate the performance bottlenecks of this system. The current system is:

- vulnerable to even small amounts of wind, causing instability.
- lacking sensors such as GPS and sonar and has unreliable sensor measurements from other sensors
- lacking in robust control due to significant control delays
- dependent on ground station computer
- vulnerable due to lacking error handling and adaptive planning in the mission planning stem.
- vulnerable to computer vision misdetections

Accordingly, some further work is proposed in this section to improve the system to deal with these deficiencies and vulnerabilities in the system.

Experiments using a more up-to-date quadcopter with GPS sensor

An issue that degraded the experimental results was the communication delay between the AR.Drone and the ground-station computer. Significant control delay can hamper precise control and may even cause instability. Some modern-day quadcopters have significantly lower communication delays than the AR.Drone. A lower communication delay improve stability. As better quadcopters become available on the market, with more on-board processing power, the entire control system could be moved on-board, significantly reducing control delay.

Modern-day quadcopters have more precise on-board controllers than the AR.Drone 2.0, and may be equipped with additional sensors like a sonar altimeter and GPS, higher quality IMUs that are more reliable and a camera with higher resolution for a more robust computer vision system.

A GPS is necessary for the system to be sufficiently reliable for real-world use-cases because the computer vision system is required to have the landing platform in the bottom facing camera frame to provide pose estimates. GPS measurements are thus required for precise pose estimates when outside of the visual range of the landing platform. Simulations displayed the capacity of the quadcopter system with GPS.

It would therefore be interesting to test the quadcopter system on a modern-day quadcopter in order to see if the results from simulations could be replicated in the real-world.

Further improve the Mission Planner algorithm

The current implementation of the mission planner is a sequential machine that transitions to the next state. Basic error handling is implemented. Possible improvements to the current mission planning system include performing actions based on the uncertainty of pose estimates as well as implementing battery control, such as proposed by Bernardini et al. [12]. In addition, an adaptive mission planner could be implemented to decide which actions to perform, whether a task is impossible to complete or whether tasks should be performed in a different order than initially proposed.

Further improvement of the error handling system of the mission planner algorithm is required for the system to be fault-tolerant, reducing the requirement for supervision. The currently implemented error handling system is programmed to perform a safe descent if any errors were detected. While such an approach may be sufficient for the experiments performed in this thesis, such an implementation will not be acceptable if the drone is operating over water or in other areas where no safe descent paths are available.

Development for Marine Operations

For further development of marine operations with the DNV ReVolt the system requires further development. Marine Operations present some unmet challenges for the system. The system requires to be able to tolerate wind, be able to tolerate landing on a moving ship, and be able to estimate pose in relation to a ship that is rolling and pitching in the waves.

In addition, the quadcopter might need to receive the location of the ReVolt vessel in order to detect the ship if it is not located in the computer vision camera frame.

7.3 Final Note

The author is inspired by the fact that this project will be continued in a future master thesis provided by the Institute of Electrical Engineering at NTNU and by the supervisor for this thesis, Anastasios Lekkas.

The author hopes this thesis lays the foundation for further development of autonomous quadcopter systems that are precise, robust and reliable.

Bibliography

[1] T. Sundvoll. "A Camera-based Perception System for Autonomous Quadcopter Landing on a Marine Vessel. Master Thesis, Department of Engineering Cybernetics. Norwegian University of Science and Technology (NTNU)". In: (2020).

[2] P. B. Hove. "Perception system for pose estimation of autonomous quadcopter. Project Thesis, Department of Engineering Cybernetics. Norwegian University of Science and Technology (NTNU)". In: (2020).

[3] M. Bjelonic. *YOLO ROS: Real-Time Object Detection for ROS*. https://github.com/leggedrobotics/darknet_ros. 2016–2018.

[4] J. Nelson et al. *Roboflow*. https://roboflow.com/. Accessed: 2020-10-20.

[5] C. Kumar et al. "YOLOv3 and YOLOv4: Multiple Object Detection for Surveillance Applications". In: *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*. 2020, pp. 1316–1321. DOI: 10.1109/ICSSIT48917.2020.9214094.

[6] S. Naskar et al. "Application of radio frequency controlled intelligent military robot in defense". In: *2011 International Conference on Communication Systems and Network Technologies*. IEEE. 2011, pp. 396–401.

[7] F. Wang et al. "Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond". In: *IEEE/CAA Journal of Automatica Sinica* 3.2 (2016), pp. 113–120.

[8] R. Amini et al. "Advancing the Scientific Frontier with Increasingly Autonomous Systems". In: (2020). arXiv: 2009.07363 [astro-ph.IM].

[9] D. Birnbacher. "Fully Autonomous Driving: Where Technology and Ethics Meet". In: *IEEE Intelligent Systems* 32.5 (2017), pp. 3–4. DOI: 10.1109/MIS.2017.3711644.

[10] M.R. Haque et al. "Autonomous quadcopter for product home delivery". In: *2014 International Conference on Electrical Engineering and Information Communication Technology*. 2014, pp. 1–5. DOI: 10.1109/ICEEICT.2014.6919154.

[11] S. Chaudhary et al. "Design of all-terrain rover quadcopter for military engineering services". In: *Nanoelectronics, Circuits and Communication Systems*. Springer, 2019, pp. 507–513.

[12] S. Bernardini et al. "Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 24.1 (May 2014). URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13670.

[13] M. F. Sani et al. "Automatic landing of a low-cost quadrotor using monocular vision and Kalman filter in GPS-denied environments". In: *Turkish Journal of Electrical Engineering and Computer Sciences* 27 (2019), pp. 1821–1838.

[14] T. G. Carreira. "Quadcopter Automatic Landing on a Docking Station". In: 2013.

[15] N. Karlsson et al. “The vSLAM Algorithm for Robust Localization and Mapping”. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 24–29. DOI: 10.1109/ROBOT.2005.1570091.

[16] R. Brockers et al. “Fully self-contained vision-aided navigation and landing of a micro air vehicle independent from external sensor inputs”. In: *Unmanned Systems Technology XIV*. Ed. by Robert E. Karlsen et al. Vol. 8387. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. June 2012, 83870Q. DOI: 10.1117/12.919278.

[17] J. Engel et al. “Scale-aware navigation of a low-cost quadrocopter with a monocular camera”. In: *Robotics and Autonomous Systems* 62.11 (2014), pp. 1646–1656.

[18] A. Bochkovskiy et al. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: (2020). arXiv: 2004.10934 [cs.CV].

[19] T. Zhang et al. “Efficient Golf Ball Detection and Tracking Based on Convolutional Neural Networks and Kalman Filter”. In: (2020). arXiv: 2012.09393 [cs.CV].

[20] H. Xu et al. “Performance Comparison of Small Object Detection Algorithms of UAV based Aerial Images”. In: *2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*. 2020, pp. 16–19. DOI: 10.1109/DCABES50732.2020.00014.

[21] W. Budiharto et al. “Fast Object Detection for Quadcopter Drone Using Deep Learning”. In: *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*. 2018, pp. 192–195. DOI: 10.1109/CCOMS.2018.8463284.

[22] N. Imanberdiyev et al. “Autonomous navigation of UAV by using real-time model-based reinforcement learning”. In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2016, pp. 1–6. DOI: 10.1109/ICARCV.2016.7838739.

[23] N. Smolyanskiy et al. “Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 4241–4247. DOI: 10.1109/IROS.2017.8206285.

[24] R. Brockers et al. “Towards autonomous navigation of miniature UAV”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 631–637.

[25] S. Roumeliotis et al. “Circumventing Dynamic modeling”: Evaluation of the Error-State Kalman Filter applied to mobile Robot Localization”. In: 2012.

[26] M. B. Alatise et al. “Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter”. In: *Sensors* 17.10 (2017). ISSN: 1424-8220. DOI: 10.3390/s17102164. URL: https://www.mdpi.com/1424-8220/17/10/2164.

[27] K. Gustavsson. “UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals”. In: 2015.

[28] C. A. Rabbath. “A finite-state machine for collaborative airlift with a formation of unmanned air vehicles”. In: *Journal of Intelligent & Robotic Systems* 70.1 (2013), pp. 233–253.

[29] F. Ghallabi et al. “Control architecture for service drone in informationally structured environment”. In: *2015 IEEE/SICE International Symposium on System Integration (SII)*. 2015, pp. 611–618. DOI: 10.1109/SII.2015.7405049.

[30] Y. Luo et al. “Intelligent control and navigation of an indoor quad-copter”. In: *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*. 2014, pp. 1700–1705. DOI: 10.1109/ICARCV.2014.7064572.

[31] T. H Nguyen et al. “Post-Mission Autonomous Return and Precision Landing of UAV”. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2018, pp. 1747–1752. DOI: 10.1109/ICARCV.2018.8581117.

[32] J. Kim et al. “Accurate Modeling and Robust Hovering Control for a Quad-rotor VTOL Aircraft”. In: *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, U.S.A. June 8–10, 2009*. Dordrecht: Springer Netherlands, 2010, pp. 9–26. ISBN: 978-90-481-8764-5. DOI: 10.1007/978-90-481-8764-5_2. URL: https://doi.org/10.1007/978-90-481-8764-5_2.

[33] T. I. Fossen. “Handbook of Marine Craft Hydrodynamics and Motion Control”. In: (2011).

[34] E. Brekke. “Fundamentals Of Sensor Fusion. Target tracking, navigation and SLAM”. In: (2020).

[35] G. Welch et al. “An introduction to the Kalman filter”. In: (1995).

[36] J. Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[37] C. G. Harris et al. “A combined corner and edge detector.” In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[38] H. Bay et al. “Surf: Speeded up robust features”. In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[39] N. Karlsson et al. “The vSLAM algorithm for robust localization and mapping”. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, pp. 24–29.

[40] S. Balaban. “Deep learning and face recognition: the state of the art”. In: 9457 (2015), 94570B.

[41] T. Do et al. “Real-Time Self-Driving Car Navigation Using Deep Neural Network”. In: (2018), pp. 7–12. DOI: 10.1109/GTSD.2018.8595590.

[42] R. Silva et al. “Machine Vision Systems for Industrial Quality Control Inspections: 15th IFIP WG 5.1 International Conference, PLM 2018, Turin, Italy, July 2-4, 2018, Proceedings”. In: July 2018, pp. 631–641. ISBN: 978-3-030-01613-5. DOI: 10.1007/978-3-030-01614-2_58.

[43] J. Redmon et al. “YOLOv3: An Incremental Improvement”. In: (2018). arXiv: 1804.02767 [cs.CV].

[44] S. Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: (2016). arXiv: 1506.01497 [cs.CV].

[45] M. Tan et al. “EfficientDet: Scalable and Efficient Object Detection”. In: (2020). arXiv: 1911.09070 [cs.CV].

[46] K. He et al. “Mask R-CNN”. In: (2018). arXiv: 1703.06870 [cs.CV].

[47] J. Kocic et al. “An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms”. In: *Sensors* (May 2019). DOI: 10.3390/s19092064.

[48] B. Simmons et al. “Training a Remote-Control Car to Autonomously Lane-Follow using End-to-End Neural Networks”. In: *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. 2019, pp. 1–6. DOI: `10.1109/CISS.2019.8692851`.

[49] J. Redmon et al. *Darknet, A neural network framework*. `https://github.com/pjreddie/darknet`. 2018.

[50] T. Lin et al. “Microsoft COCO: Common Objects in Context”. In: (2015). arXiv: `1405.0312 [cs.CV]`.

[51] G. Jocher et al. “ultralytics/yolov5”. In: (Jan. 2021). DOI: `10.5281/ZENODO.4418161`. URL: `https://zenodo.org/record/4418161`.

[52] Z. Jiang et al. “Real-time object detection method based on improved YOLOv4-tiny”. In: (2020). arXiv: `2011.04244 [cs.CV]`.

[53] C. Wang et al. “Scaled-YOLOv4: Scaling Cross Stage Partial Network”. In: (2021). arXiv: `2011.08036 [cs.CV]`.

[54] D. Fung et al. “Recurrent CNNs for Bounding Box stability in Object Detection”. In: (2018).

[55] J. Walsh et al. “Deep Learning vs. Traditional Computer Vision”. In: Apr. 2019. ISBN: 978-981-13-6209-5. DOI: `10.1007/978-3-030-17795-9_10`.

[56] S. C. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal Of Big Data* 6 (2019).

[57] C. Li et al. “TrackNet: Simultaneous Object Detection and Tracking and Its Application in Traffic Video Analysis”. In: (2019). arXiv: `1902.01466 [cs.CV]`.

[58] D. Gunning. “Explainable artificial intelligence (xai)”. In: *Defense Advanced Research Projects Agency (DARPA), nd Web* 2.2 (2017).

[59] M. Kok et al. “Using Inertial Sensors for Position and Orientation Estimation”. In: *Foundations and Trends in Signal Processing*. Vol. 11. 1-2. 2017, pp. 1–153. DOI: `10.1561/2000000094`.

[60] C. E. Cummings et al. “Finite State Machine (FSM) Design Synthesis using SystemVerilog - Part I”. In: 2019.

[61] S. Piskorski et al. “Ar. drone developer guide”. In: *Parrot, sdk* 1 (2012).

[62] Parrot SA. *Parrot AR.Drone 2.0 Documentation*. `https://ardrone-autonomy.readthedocs.io/en/latest/`. 2015.

[63] M. Monajjemi. *ardrone_autonomy*. `https://ardrone-autonomy.readthedocs.io/en/latest/`. Accessed: 2020-12-05.

[64] M. Quigley et al. “Programming Robots with ROS. A practical introduction to the robot operating system”. In: (2015).

[65] *CVAT*. `https://cvat.org`. Accessed: 2020-10-20.

[66] Google Inc. *Google Colab: Providing free GPUs for online training of*. `https://colab.research.google.com/`.

[67] J. Nelson. *Roboflow Model Library: Free open-source Google Colab notebooks for training of DNN models*. `https://models.roboflow.com/`.

[68] A. Bochkovskiy. *YOLOv4 - Neural Network for Object Detection*. `https://github.com/AlexeyAB/darknet`. 2020.

[69] T. L. Grigorie et al. “Aircrafts’ altitude measurement using pressure information: barometric altitude and density altitude”. In: *wseas transactions ON circuits AND systems. ISSN* (2010), pp. 1109–2734.

[70] K. Gu et al. “Survey on Recent Results in the Stability and Control of Time-Delay Systems”. In: *J. Dyn. Syst. Meas. Contr.* 125 (June 2003), pp. 158–165. DOI: 10.1115/1.1569950.

NTNU

Kunnskap for en bedre verden