Master's thesis

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

Aleksander Roterud Asp

Autonomous UAV Navigation Based on Fiducial Markers

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
June 2020

NTNU
Norwegian University of Science and Technology

Aleksander Roterud Asp

Autonomous UAV Navigation Based on Fiducial Markers

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
June 2020

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

Abstract

Drones are quickly rising in popularity for carrying out missions in environments where humans have difficulties performing tasks manually. They excel in regards to both safety and efficiency. However, accuracy in the landing phase for autonomous drones is a significant challenge. This thesis presents the author's contribution in the continued development of a system where a drone is to land on and pick up a micro underwater glider autonomously. This operation is to be performed using computer vision and DUNE - Unified Navigation Environment.

In this thesis, a custom fiducial marker for use in ocean surface environments is presented. An automatic HSV filter is created and optimized to solely filter through aerospace-orange-colored objects from the camera view. A basic camera calibration method using OpenCV and CharUco boards to compensate for radial and tangential distortion is detailed. Next, a color detection scheme is described as an initial method to provide a navigational target. However, the main maneuvering is performed based on the detection and pose estimation of the ArUco marker within the custom marker. A constant bearing guidance system is presented and implemented, and a simplified wave approximation is defined and used to test viability in oceanic environments. A state machine that governs simulated and real flights is presented, as well as the results of these types of flight.

Simulation results with camera-based navigation and stationary markers produce highly accurate landings. However, the conditions are not realistic; there is no simulated wind, the terrain is a monochrome blue color, and no light reflection can occur.

Field tests prove the automatic HSV filter to be inadequate, as its performance is inconsistent. A static HSV filter is optimized based on data from a 3DR Solo drone, and is shown to be highly versatile. A build-up in camera feed delay limits evaluation of the implemented navigation methods with respect to drone maneuvering. However, the accuracy of the navigation targets acquired is discussed.

Sammendrag

Droner vokser raskt i popularitet for å gjennomføre oppdrag i miljø hvor det er problematisk for mennesker å utføre oppgaver manuelt. De utmerker seg med hensyn til både sikkerhet og effektivitet, men nøyaktigheten i landingsfasen for autonome droner er en betydelig utfordring. Denne avhandlingen presenterer forfatteren sitt bidrag i den kontinuerlige utviklingen av et system hvor en drone skal autonomt lande på og plukke opp en mikro undervannsglider. Dette skal gjennomføres med bruk av datasyn og DUNE - Unified Navigation Environment.

I denne avhandlingen presenteres en spesiallaget markør, tilpasset til bruk ved havoverflaten. Et automatisk HSV filter er opprettet og optimert til å kun filtrere gjennom *aerospace*-oransjefargede objekter fra kamerabildet. En grunnleggende kamerakalibreringsmetode ved hjelp av OpenCV og CharUco-brett er beskrevet og brukt for å kompensere for radiell og tangentiell forvrengning. Videre er en fargedetekteringsmetode beskrevet som en innledende prosedyre for å anskaffe navigeringsmål. Hoved manøvreringen er derimot basert på posisjon- og orientering-estimering av ArUco markøren inne i den spesiallagde markøren. Et *constant bearing guidance system* er definert og implementert, og en forenklet bølgeapproksimering er brukt til å teste om systemet er levedyktig i havmiljø. En tilstandsmaskin som styrer både simulerte og reelle flyvninger er presentert, samt resultater fra disse typer flyvninger.

Simuleringsresultater med kamerabasert navigasjon og stasjonære markører produserer nøyaktig landinger. Forholdene er derimot ikke realistiske; det er ingen simulert vind, terreng er havblått, og ingen lysrefleksjon kan inntreffe.

Feltprøver viser at det automatiske HSV-filteret er utilstrekkelig, da filtreringen er inkonsekvent. Et statisk HSV-filter er optimalisert basert på data fra en 3DR Solo-drone, og viser seg å være svært allsidig. En oppbygging av forsinkelse i bildeoverføring begrenser evalueringen av de implementerte navigasjonsmetodene med hensyn til dronemanøvrering. Imidlertid diskuteres nøyaktigheten til de oppnådde navigasjonsmålene.

Preface

This thesis concludes my time and work as a Master's student of Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The thesis represents the final development of my Pre-Master's project, which was co-authored by Marius Eskedal.

The hardware needed to conduct the experiments presented in this project was provided by NTNU, specifically by the UAV laboratory and postdoctoral researcher Artur Zolich. The Unified Navigation Environment (DUNE) is the main software needed to build and run the project and was developed by Laboratório de Sistemas e Tecnologia Subaquática (LSTS) in Porto, Portugal. The UAV laboratory provides a repository for NTNU related DUNE development.

The COVID-19 outbreak, and subsequently the lockdown of NTNU and the UAV laboratory, restricted the available time window and possibility of conducting field tests. As a result, limited field operations have been completed.

I want to thank my supervisors for their help throughout the semester. Meetings and discussions with main supervisor Tor Arne Johansen and co-supervisor Artur Zolich helped me define my target areas and project goals. Their feedback during the project also helped when I found myself at crossroads. Zolich specifically helped me get a quicker understanding of the different hardware components as well as the software stack and played a major role in overcoming technical challenges. I would also like to thank PhD student Martin Sollie for his help and advice. Lastly, I would like to acknowledge the contributions made by my Master's peer, Marius Eskedal. Eskedal and I had several productive meetings throughout the semester, where we shared ideas and discussed potential issues and solutions.

Table of Contents

List of Tables

List of Figures

Abbreviations

AOI	=	Area of interest
AUV	=	Autonomous underwater vehicle
AVC	=	Advanced video coding
BBB	=	BeagleBone Black
BGR	=	Blue-Green-Red
DOF	=	Degrees of freedom
DUNE	=	Unified navigation environment
EKF	=	Extended Kalman filter
GPS	=	Global positioning system
HSV	=	Hue-Saturaion-Value
IMC	=	Intermodule communication
LOS	=	Line-of-sight
LSTS	=	Underwater Systems and Technology Laboratory, (Laboratório de Sistemas e Tecnologia Subaquática)
MP	=	Mega pixel
MUG	=	Micro underwater glider
NED	=	North-East-Down
NTNU	=	Norwegian University of Science and Technology
RGB	=	Red-Green-Blue
ROUV	=	Remotely operated underwater vehicle
ROV	=	Remotely operated vehicle
RTP	=	Real-time protocol
UAV	=	Unmanned Aerial Vehicle
UDP	=	User datagram protocol
USARSim	=	Unified System for Automation and Robot Simulation
USV	=	Unmanned surface vehicle
VTOL	=	Vertical take-off and landing

Chapter 1

Introduction

1.1 Project description

This project is part of a broader international project called Oasys. The Oasys project aims to develop fully automated robotic systems in order to reduce the cost of ocean observation and monitoring. Oasys is a collaboration between OsloMet, NTNU, Norwegian Polar Institute and TriOS [17].

The ultimate goal of this project is to autonomously pick up a micro underwater glider (MUG) from the ocean surface. This goal is to be achieved using DUNE, camera vision methods, a custom marker and a multi-rotor drone with a magnetic pick-up system. There is no communication between the MUG and the drone, although the drone itself can be equipped with a radar. This thesis details the theory behind major areas of the system as well as the implementations, experiments and results acquired in the pursuit of the end goal.

1.2 Motivation

The motivation behind the Oasys project is described in full on their website [17] and is summarized as follows:

”One of the barriers towards a better understanding and sustainable development of marine-related economic activity is the high cost associated with ocean observing systems. Autonomous robotic systems are steadily revolutionizing the way we obtain data and interact with the ocean. However, most of existing autonomous systems still require the involvement of manned missions in the deployment/recovery phases which represents a high percentage of the total operational costs”

As this thesis is within the scope of the Oasys project, the motivation is similar, but more related to the specific challenges that appear from attempting to realize the project goals. Among these challenges are the dangers and difficulties of manned missions in many offshore environments. Conditions at, above and below the ocean surface are restricting factors for both manned and unmanned missions. However, the advances in UAV technology are making these conditions less restrictive and are reducing the associated mission risk.

Detecting objects, animals or people in the ocean is critical in many planned missions, as well as in unexpected recovery or rescue missions. In poor conditions, sending a single person or even a team of professionals carries a risk. Having the option of immediately deploying an unmanned vehicle equipped with sensors that negate some of the difficult conditions, for example, poor lighting or strong currents, increases safety and chance of success.

1.3 Problem Overview

The challenges that arise from the project description are many, but can be divided into separate sections, some of which will not be tackled in this project. The division is as follows:

- Drone and systems calibration
- Initial localization of the MUG
- Detection of the MUG
- Approaching and landing on the MUG
- Magnetic pick up

1.3.1 Calibration

Calibration of the drone and its onboard hardware must be executed before the process of locating and landing on the MUG can begin. Particularly the drone's camera must be calibrated.

1.3.2 Initial Localization

The initial localization of the MUG is an aspect of the project that has been disregarded. Instead, the location of the MUG is assumed to be known at take-off. The decision to exclude this step is partly due to the typically low battery-life of unmanned aerial vehicles (UAVs) in flight, rendering them unsuited for extended search missions.

1.3.3 Detection of the MUG

There is no communication link between the drone and the MUG. In order to detect and navigate towards the MUG, computer vision software is to be implemented. Detection of the MUG is simplified by attaching a marker to it. Specifically, a custom 200x100 mm international orange (aerospace) marker, with an 80x80 mm ArUco code in its center. Methods for detecting ArUco markers are present in the open-source computer vision and machine learning software library OpenCV. Before the drone is close enough to the marker for ArUco detection to be feasible, color detection will be the primary method to obtain navigation targets.

1.3.4 Approach and landing

Once the marker has been detected, the position and attitude of the onboard camera relative to the marker can be calculated. This is due to the known dimensions of the ArUco marker, and algorithms provided by the OpenCV framework. These now known variables for relative distance and attitude can be used in a control system to pilot the drone towards the MUG. The control system to be implemented must, therefore, be able to communicate with the camera and perform inertial navigation maneuvers.

The MUG only has a small part of its body on the surface, a ”wing”, which is the same size as the custom marker. Since the wing is significantly smaller than the S1000 drone intended for use in this project, there is a chance that the ArUco marker will not be visible to the camera in the final descent of the drone. Because of this issue, as well as the current-induced motion the MUG experiences, the motion of the MUG should be modelled. This model can be used to make predictions of the MUG’s position when the ArUco marker leaves the camera view, improving landing accuracy. However, these predictions become increasingly inaccurate over time. If the marker is not detected within a given time period, and the drone has not landed, the drone should ascend until the marker can be detected again.

1.3.5 Magnetic Pick Up

The magnetic pick-up system will employ an electropermanent magnet. Electropermanent magnets have their magnetic fields switched on and off using a pulse of electric current [27]. Therefore the drone’s onboard hardware must be able to switch the magnet on and off by triggering an electric pulse.

The magnet itself was not acquired during the duration of this thesis, and therefore there were no attempts to implement a magnetic pick-up system.

1.4 Outline

Chapter 2 - Literature Review: This chapter discusses some of the previous work conducted in fields related to this project.

Chapter 3 - Basic Theory: The chapter presents the relevant theory for the main ideas and concepts used later in the project. This includes reference frames and transformations, camera vision, segmentation, guidance systems and Kalman filters.

Chapter 4 - System Overview: Chapter 4 describes the central software and hardware components used in the project. This includes DUNE, ArduPilot, the drone's camera and radar, the ArUco markers and more.

Chapter 5 - Modelling: Chapter 5 details the dynamics of a general multi-rotor drone, as well as the modelling of impactful disturbances such as ocean current, waves and wind.

Chapter 6 - Implementation: The chapter describes the implementation details of the camera vision aspect, the guidance system, the state machine and more. The integration into DUNE is also described.

Chapter 7 - Experiments and Results: The chapter presents the experiments performed in regards to camera calibration, filtering, pose estimation as well as simulated and real flights.

Chapter 8 - Discussion and Conclusion The final chapter contains a discussion of the achieved results, a conclusion and suggestions of future work.

Chapter 2

Literature Review

The main challenge of the project is making the drone autonomously land on the floating target using computer vision. To help overcome the multiple obstacles this challenge poses, there exists relevant research to learn from and build on.

Vision-Based Landing of a Simulated Unmanned Aerial Vehicle with Fast Reinforcement Learning [34] looks into one of the critical steps before testing computer vision systems and control systems in the field, that is, simulation. In the paper, a vision-based landing approach for autonomous UAVs is proposed, using fast reinforcement learning. This approach is tested in an extended version of the USARSim (Unified System for Automation and Robot Simulation) environment with a simulated quadrocopter [34]. In the simulation, the quadrocopter has a camera fixed at the center of its belly, and the target landing site is an entirely black circle surrounded by circles in a range of gray. The approach makes use of the OpenCV framework to detect the target and the Least-Square Policy Iteration as the reinforcement learning method. In the event that ArUco markers are found to be sub-optimal, applying techniques and concepts from this paper is an alternative.

Vision-Based Autonomous UAV Navigation and Landing for Urban Search and Rescue [31] researches the use of computer vision in the landing of a simulated unmanned aerial vehicle using fast reinforcement learning. The research is done in the context of the need for safe autonomous navigation for UAVs in disaster struck environments, in order to properly locate potential survivors. The paper presents a system for UAV navigation and landing that does not require any prior information about its environment. The computer vision system is used to obtain potential landing sites, thereafter, an algorithm is used to determine the optimal landing site. When the final site has been obtained, the system computes a minimum-jerk trajectory with both nearby obstacles and the UAV dynamics taken into consideration and then performs the landing maneuver. The entire navigation and landing system is run online on a low-power embedded computer aboard the UAV. The paper [31] does not consider landing on a moving target, but the software and algorithms for continuously evaluating potential landing sites and avoiding obstacles have transferable value to moving-target situations.

Autonomous Landing of a UAV on a Moving Platform Using Model Predictive Control [23] presents an autonomous landing method for micro UAVs to land on moving targets in the presence of uncertainties and disturbances. For optimal localization of the moving platform, a Kalman filter is implemented, and model predictive control is developed as part of the system architecture. The computer vision system makes use of an AprilTag, similar to the before-mentioned ArUco codes, to estimate the relative position and pose of the platform with respect to the camera. The simulation results in this paper demonstrate an autonomous landing on a platform travelling up to $12\,\mathrm{m\,s^{-1}}$ with an error of less than $37\,\mathrm{cm}$ from the platform's center. The use of markers in computer vision systems to estimate camera pose, as well as landing on a moving target, are highly relevant tasks for this project.

In *Autonomous Maritime Landings for Low-cost VTOL Aerial Vehicles* [30] an architecture that negates sensor limitations and allows for precise pose estimation, even in the presence of wind disturbances, is proposed. The final landing method performs landing maneuvers in the body-fixed reference frame to nullify poor estimation accuracy caused by noisy measurements from GPS and magnetometers. The total system consists of three different stages. The initial stage calculates an intersection point based on current positions of both vehicles and the estimated velocity of the marine vessel. The UAV flies to this point autonomously using constant heading control. Note that the intersection point can be updated repeatedly to compensate for drift and varying vessel velocity. Stage two begins when the UAV is in the vicinity of the marine vessel and starts its search for an AprilTag. Once the AprilTag has been located, the control system switches from inertial control to relative control, using the body frame as a reference and initiates the final landing sequence. Several of these stages relate to the challenges in this project, especially landing on a marine vessel. However, a sizeable boat was used as the marine vessel in this paper [30], with a significantly larger landing area than that of the micro underwater glider.

Vision-Based Autonomous Landing of a Quadrotor on the Perturbed Deck of an Unmanned Surface Vehicle [33] also uses a fiducial marker, placed on the platform of the unmanned surface vehicle (USV) to accommodate the task of finding the USV's relative position and pose. An extended Kalman (EKF) filter is used to estimate the current position of the USV, to compensate for the potential temporary loss of marker detection. The EKF provides accurate enough estimations, that in combination with odometry, this method is found to be sufficient and applicable in poor weather conditions and in the absence of a global positioning system. A relatively small USV is used in this paper; however, it is still larger than the quadrocopter itself, demanding less accuracy than what is required in this project.

Multi-rotor pickup of object in the sea [26] is a previous Master's thesis written by Jens Joberg regarding similar goals and issues. In Joberg's thesis, a state machine governing the different stages of a flight is proposed, a triangulation method to acquire position estimates is developed and implemented in DUNE, and both simulations and field tests with a multi-rotor drone are performed using a constant bearing path controller. The result of this thesis provides evidence that a constant bearing guidance system produces relatively accurate flight maneuvers. The concept of both a state machine and a dedicated guidance system are highly applicable in this thesis.

Automatic Landing of Multi-Rotor on Moving Platform [28] is also a previous Master's thesis on a similar topic, written by Vuk Krivokapic. In Krivokapic's thesis, relevant weather disturbances are modelled and included in his developed simulation environment for automatic multirotor landing, which is implemented in DUNE. Algorithms for high-level velocity control are developed and tuned using this simulation environment. Algorithms for error handling during a landing approach are also developed. Based on his results, Krivokapic suggests a minimum landing platform size, as well as minimum weather conditions required to carry out a successful landing. The modelling and simulation of weather conditions is an important aspect that can be implemented in this thesis as well.

Chapter 3

Theory

3.1 Reference frames and Transformations

3.1.1 Reference frames

A coordinate system intended to express an object's position is generally made up of two or three axes, depending on the number of dimensions the object can move in and which of these are considered relevant. The position of the object can then be uniquely expressed using the same number of coordinates as the number of axes.

Coordinate systems can have their origin fixed in different locations. A reference frame defines the location and orientation of a coordinate system. For small aerial vehicles, it is a convention to adhere to the North-East-Down (NED) reference frame as an inertial frame. An inertial frame is defined as a reference frame in which the object does not accelerate when there is zero net-force acting upon said object [24]. In the NED-frame, the x-axis points north, the y-axis east and the z-axis down towards earth to complete a right-hand coordinate system. The origin of the NED frame is a starting point of the earth's surface.

In the case of controllable moving bodies, e.g. UAVs, it is also common practice to define a body reference frame. In the body frame, the origin of the coordinate system is typically defined as either the object's center of mass or the object's geometric center. For aerial vehicles it is common that the x-axis points out of the nose/front of the vehicle, the y-axis points out of the right wing/side, and the z-axis out of the belly/bottom of the aircraft. Since most events of interest will be below a flying aircraft, it is sensible to define positive z-values below the vehicle.

A final common frame for aerial vehicles is the vehicle-frame. The vehicle frame is simply a NED coordinate system with the origin fixed in the geometric center or mass center of the vehicle. With several different reference frames, a transformation from a point in one frame to another is needed.

3.1.2 Rotations and Transformations

The relationship between coordinate frames can be expressed using a product of rotational matrices. A rotation matrix for each axis is needed as well as the angular rotation about each axis with respect to one of the frames. The rotation matrix for each axis is defined in equation 3.1.

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{3.1a}$$

$$\boldsymbol{R}_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.1b}$$

$$\boldsymbol{R}_z = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1c}$$

This yields a rotation matrix from coordinate system *a* to coordinate system *b* as follows:

$$\boldsymbol{R}_a^b = \boldsymbol{R}_x \boldsymbol{R}_y \boldsymbol{R}_z$$

With this rotation matrix one can perform coordinate transformation from reference frame *a* to reference frame *b* with the following transformation:

$$\boldsymbol{p}^b = \boldsymbol{R}_a^b \boldsymbol{p}^a$$

To describe both orientation and position of a coordinate frame with respect to a reference frame, the homogeneous transformation matrix T_a^b is used:

$$\boldsymbol{T}_a^b = \begin{bmatrix} \boldsymbol{R}_a^b & \boldsymbol{t}_{ab}^a \\ \boldsymbol{0}_{1x3} & 1 \end{bmatrix},$$

where $\boldsymbol{t}_{ab}^a$ is the position of the origin of frame *b* relative to frame *a*, expressed in *a* coordinates.

3.2 Camera Vision

A camera image is the projection of a three-dimensional environment onto a two-dimensional plane. This projection process, or transformation, varies depending on the intrinsic and extrinsic parameters of the camera. The intrinsic parameters, or calibration matrix, define a mapping between the camera coordinates and pixel coordinates in the image frame. The extrinsic parameters define the camera's pose with respect to the world frame.

3.2.1 Pinhole Model

A common camera model, which will be used in this project, is known as the pinhole camera model. Figure 3.1 is obtained from p.320 in *Robotics, Vision and Control* [22] and depicts the pinhole model with a 3D point projected onto the image plane.

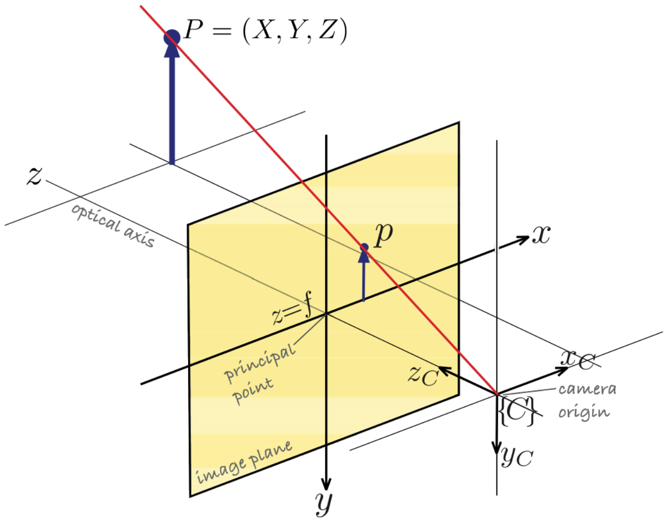


Figure 3.1: Pinhole model.

In Figure 3.1 the origin of the image plane, also known as the principal point, is a distance f in front of the camera frame $\{C\}$, where f is the focal length of the camera lens. To determine how a point $\boldsymbol{P}^w = (X, Y, Z)$ in the world frame is projected onto the image plane $\boldsymbol{p}^i = (x, y)$, properties of similar triangles are utilized. This gives:

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{3.2}$$

which shows that the unit of x and y will be identical to the unit of focal length. Since images commonly use pixels as unit of measure, the focal length is defined in pixels. To allow for any pixel aspect ratio, f_x defines the focal length in pixel width and f_y defines the focal length in pixel height. This yields the slight modification of 3.2:

$$x = f_x\frac{X}{Z}, \quad y = f_y\frac{Y}{Z}, \tag{3.3}$$

Unlike the current definition of the image frame, digital images typically have their origin in the top left corner. To account for this, an offset equal to the x- and y-component of the principal point, (c_x, c_y) is included. The vector $[u, v]^T$ is used to describe a point on the image plane with origin in the top left corner and is defined as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x + c_x \\ y + c_y \end{bmatrix} = \begin{bmatrix} f_x X/Z + c_x \\ f_y Y/Z + c_y \end{bmatrix} \tag{3.4}$$

However, equations 3.2, 3.3 and 3.4 are only valid under the assumption that the camera frame and the world frame have identical orientations and origins. This is not always the case, which creates the need for a joint rotation and translation matrix, $\{\boldsymbol{R}|\boldsymbol{t}\}$. Incorporating the matrix into the equation and transforming the model into homogeneous coordinates leads to its matrix form as

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{3.5}$$

$$s\tilde{\boldsymbol{p}}^i = \boldsymbol{C}\{\boldsymbol{R}|\boldsymbol{t}\}\tilde{\boldsymbol{P}}^w, \tag{3.6}$$

where s is a scaling factor and $\tilde{\boldsymbol{p}}^i$ is the homogeneous image coordinates resulting from the projection of the homogeneous world point $\tilde{\boldsymbol{P}}^w$. $\boldsymbol{C}$ is the calibration matrix, containing the intrinsic parameters and $\{\boldsymbol{R}|\boldsymbol{t}\}$ is the projection matrix containing the extrinsic parameters [13].

3.2.2 Lens Distortion

Lens distortion is an optical aberration concerning how magnification in an image varies at a fixed focal length. The varying magnification causes straight lines in the real world to map to curved lines in the image frame. This is due to the curvature of the camera lens in use. Pinhole cameras experience two main types of distortion; radial distortion and tangential distortion.

Radial distortion is when a point in the real world is not projected onto the corresponding ideal position in the image plane, but rather distorted radially along the line, $\boldsymbol{OP}$, from the principal point in the image plane to the ideal projected position. Tangential distortion is when the projected point is distorted tangentially relative to the imagined circle with the line, $\boldsymbol{OP}$, as its radius. Figure 3.2 from *Laboratory calibration of star sensor with installation error using a nonlinear distortion model* [29] illustrates these distortions.

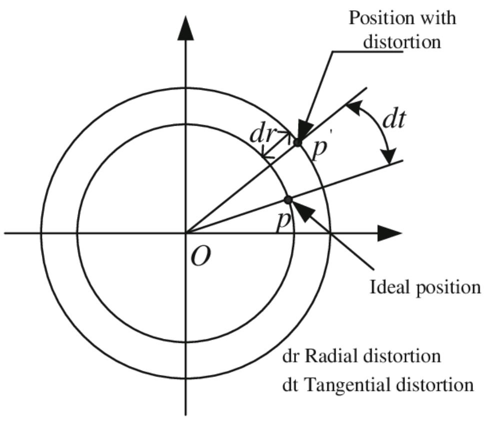
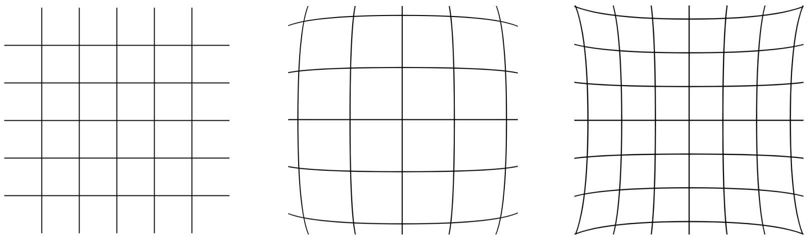


Figure 3.2: Lens Distortion in image plane.

The radial distortion is typically most severe, while tangential distortion is rarely prominent [13].

Barrel distortion, pincushion distortion and complex distortion are all different types of radial distortion. With barrel distortion, the image magnification decreases with the distance from the image optical axis. The optical axis is an imaginary line that passes through the center of curvature for systems composed of mirrors and lenses. This means that straight lines remain straight at the center of the image, but curve outwards away from the center. This creates the effect that the image has been mapped onto a sphere or barrel [3].

Pincushion distortion is the opposite of barrel distortion. Here, the magnification in the image increases with the distance from the optical axis. This causes lines away from the center to curve inwards. Complex distortion, or mustache distortion, is a mixture of both barrel and pincushion distortion. In complex distortion, the characteristics of a barrel distortion dominate in the center of the image, while pincushion characteristics dominate at the peripherals. Figure 3.3 displays the different types of radial distortion, excluding complex distortion.



(a) No distortion. **(b)** Barrel distortion. **(c)** Pincushion distortion.

Figure 3.3: Barrel distortion and pincushion distortion of undistorted grid.

3.3 Segmentation

The idea of image segmentation is to partition the input image in a manner that makes it more meaningful from an analytical standpoint. A common use of segmentation is to locate objects or contours by detecting edges and corners. In a segmented image, areas that have the same label or colour, share at least one similar characteristic.

There are many different methods available to achieve segmentation, one of the simplest being thresholding. The most basic threshold method takes a gray-scale image as an input and outputs a black and white binary image by transforming each pixel to either black or white, depending on the threshold value.

Another method that segments an input image into a binary black and white image is hue-saturation-value (HSV) filtering. HSV filters are typically used for color filtering as the HSV color space describes colors with more meaningful attributes for computing purposes than red-green-blue (RGB) values. The hue attribute represents the base color, often quantified as a numeric degree, where 0° represents red primary, 120° represents green primary, and 240° represents blue primary. The saturation parameter describes the intensity of the color, and the value parameter describes the illumination or brightness of the color. At zero value, the color is completely black regardless of hue and saturation. At 100% value and zero saturation, the color is white regardless of hue. At 100% value and saturation, the color is its most intense and bright.

3.4 Guidance System

A guidance system computes the desired position, velocity and attitude of a controllable vehicle, which are to be used in a given control system. These desired parameters will vary, depending on the guidance method in use. In the case of target tracking using velocity control, commonly used methods include line-of-sight guidance, pure pursuit guidance and constant bearing guidance.

Line-of-sight (LOS) guidance is a 3-point guidance scheme in which the interceptor, the controllable vehicle/object, must limit its motion along the reference-target line of sight vector. This guidance method is typically used in surface-to-air missiles [25].

The pure pursuit guidance method is similar to that of the LOS method, but is instead a 2-point guidance scheme, where no reference point is in use. With a pure pursuit approach, the interceptor aligns its linear velocity with the interceptor-target line of sight vector. This is a common strategy in nature as well, where many predators that chase prey adopt this method. However, in modern technology it is commonly employed in air-to-surface missiles [25]. The desired velocity can be calculated as follows:

$$\boldsymbol{v}_d^n = -k \frac{\tilde{\boldsymbol{p}}^n}{||\tilde{\boldsymbol{p}}^n||}, \tag{3.7}$$

where

$$\tilde{\boldsymbol{p}}^n := \boldsymbol{p}^n - \boldsymbol{p}_t^n \tag{3.8}$$

is the line-of-sight vector between the interceptor and the target and

$$k = U_{a,max} \frac{||\tilde{\boldsymbol{p}}^n||}{\sqrt{(\tilde{\boldsymbol{p}}^n)^T \tilde{\boldsymbol{p}}^n + \Delta_{\tilde{p}}^2}} \tag{3.9}$$

where $U_{a,max}$ defines the max approach speed toward the target and $\Delta_{\tilde{p}} > 0$ impacts the transient interceptor-target rendezvous behaviour [25]

Constant bearing guidance differs from the previously described methods as it is a predictive approach. Instead of following a target, this method predicts an intersection point that controls the interceptor towards this point. It is a 2-point guidance scheme and is often referred to as proportional navigation. It is considered ideal for scenarios that involve non-maneuvering targets [25]. The desired velocity can be calculated as follows

$$\boldsymbol{v}_d^n = \boldsymbol{v}_t^n + \boldsymbol{v}_a^n \tag{3.10}$$

where $\boldsymbol{v}_a^n = -k \frac{\tilde{\boldsymbol{p}}^n}{||\tilde{\boldsymbol{p}}^n||}$ and $\boldsymbol{v}_t^n$ is the target velocity.

3.5 Kalman filter

Modern control systems are typically equipped with a state estimator used in the processing of sensor and navigation data. This raw data is typically sent to a signal processing unit for quality control and wild point removal before being transmitted to a control system. The state estimator is capable of noise-filtering, making state predictions and reconstructing unmeasured states. One of the more famous algorithms for this purpose is the Kalman filter, first introduced in the 1960's [25].

The Kalman filter is an efficient recursive algorithm that uses a series of noisy measurements from a system's sensors in order to estimate the states of a dynamic system. The Kalman filter works for both linear and nonlinear systems. The noise-filtering capabilities of the Kalman filter allow it to remove both white noise and colored noise from the state estimates and even wild point removal can be implemented. If a temporary loss of measurements should occur, the filter equations behave as a predictor, ensuring the controlled vehicle does not immediately deviate far from its desired pose. At the moment when new measurements are available, the predictions are corrected and updated to give the minimum variance estimate.

A necessary assumption when designing a Kalman filter is that the model of the system is observable. This assumption must hold in order for the estimated states to converge to the actual states. Additionally, with an observable system, the state vector can be reconstructed recursively using the measurement vector and the control input vector.

A discrete-time Kalman filter is often the state estimator applied in electromechanical systems and is defined in terms of the system model as follows [25]:

$$\boldsymbol{x}(k+1) = \boldsymbol{\Phi}\boldsymbol{x}(k) + \boldsymbol{\Delta}\boldsymbol{u}(k) + \boldsymbol{\Gamma}\boldsymbol{w}(k), \tag{3.11a}$$

$$\boldsymbol{y}(k) = \boldsymbol{H}\boldsymbol{x}(k) + \boldsymbol{v}(k) \tag{3.11b}$$

where

$$\boldsymbol{\Phi} = \exp(\boldsymbol{A}h) \approx \boldsymbol{I} + \boldsymbol{A}h + \frac{1}{2}(\boldsymbol{A}h)^2 + ... + \frac{1}{N!}(\boldsymbol{A}h)^N \tag{3.12}$$

$$\boldsymbol{\Delta} = \boldsymbol{A}^{-1}(\boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{B}, \tag{3.13}$$

$$\boldsymbol{\Gamma} = \boldsymbol{A}^{-1}(\boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{E} \tag{3.14}$$

and h is the sampling time.

The algorithm for the linear discrete-time Kalman filter is depicted in Table 3.1, based on Table 11.1 from *Handbook of Marine Craft Hydrodynamics and Motion Control* [25].

Table 3.1: Discrete-time Kalman filter

Description	*Equation*
Design Matrices (usually constant)	$\boldsymbol{Q}(k) = \boldsymbol{Q}^T(k) > 0, \quad \boldsymbol{R}(k) = \boldsymbol{R}^T(k) > 0$
Initial Conditions	$\bar{\boldsymbol{x}}(0) = \boldsymbol{x}_0$ $\bar{\boldsymbol{P}}(0) = E[(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))^T] = \boldsymbol{P}_0$
Kalman gain Matrix	$\boldsymbol{K}(k) = \bar{\boldsymbol{P}}(k)\boldsymbol{H}^T(k)[\boldsymbol{H}(k)\bar{\boldsymbol{P}}(k)\boldsymbol{H}^T(k) + \boldsymbol{R}(k)]^{-1}$
State estimate update	$\hat{\boldsymbol{x}}(k) = \bar{\boldsymbol{x}}(k) + \boldsymbol{K}(k)[\boldsymbol{y}(k) - \boldsymbol{H}(k)\bar{\boldsymbol{x}}(k)]$
Error covariance update	$\hat{\boldsymbol{P}}(k) = [\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)]\bar{\boldsymbol{P}}(k)[\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)]^T$ $+\boldsymbol{K}(k)\boldsymbol{R}(k)\boldsymbol{K}^T(k), \quad \hat{\boldsymbol{P}}(k) = \hat{\boldsymbol{P}}^T(k) > 0$
State estimate propagation	$\bar{\boldsymbol{x}}(k+1) = \boldsymbol{\Phi}(k)\hat{\boldsymbol{x}}(k) + \boldsymbol{\Delta}(k)\boldsymbol{u}(k)$
Error covariance propagation	$\bar{\boldsymbol{P}}(k+1) = \boldsymbol{\Phi}(k)\hat{\boldsymbol{P}}(k)\boldsymbol{\Phi}^T(k) + \boldsymbol{\Gamma}(k)\boldsymbol{Q}(k)\boldsymbol{\Gamma}^T(k)$

Chapter 4

System Overview

4.1 DUNE - Unified Navigation Environment

DUNE is a runtime environment for unmanned systems' onboard software developed by the University of Porto [4]. DUNE is responsible for interacting with sensors, payloads and actuators, but also for communications, maneuvering, navigation, plan execution and vehicle supervision [5]. In essence, it acts as a task manager as well as a message bus manager. DUNE is independent of both architecture and operating system thanks to an abstraction layer, increasing its portability and versatility.

One of the main features of DUNE is the message passing concept. The idea is that different tasks, written for arbitrary parts of the system, run on separate threads, but can share data using a message bus.

4.1.1 The DUNE Task & IMC messages

In general DUNE tasks can be split into two categories, producer tasks and consumer tasks. The producer task is one that typically creates an intermodule communication (IMC) message variable, which typically contains raw or processed data. This data can then be dispatched to the message bus using the `dispatch(msg)` method, which sends the variable as an IMC message. In order for the consumer task to ”consume” the data in the associated IMC message, it needs to bind the IMC message with `bind<IMC::[msg_type]>(this)` and contain a consume method; `consume(const IMC::[msg_type]){}`. When a producer task dispatches a message, the consume method will be called for all active tasks that have a subscription to the given message type.

In order for different tasks to interact, they must know that the other exists. Therefore, tasks that communicate with each other through IMC messages must be declared in the same configuration file. Parameters for any task declared in the configuration file can be set in this file.

4.2 ArduPilot

ArduPilot is an open-source autopilot software suite developed for unmanned vehicles. The software suite consists of navigation software as well as ground station control software [8]. ArduPilot can be used for both simulations and field tests. A major benefit of ArduPilot is that it transforms input velocities in north, east and down directions to corresponding Euler angles and rotor velocities.

4.3 Neptus

Neptus is a command and control software developed by the Underwater Systems and Technology Laboratory (LSTS) and is a part of the LSTS toolchain [6]. Neptus is intended to command and control unmanned vehicles and supports autonomous underwater vehicles (AUVs), remotely operated underwater vehicles (ROUVs), UAVs and more [2]. Neptus supports all phases of a typical mission; planning, simulation and execution, as well as post-mission analysis.

4.4 FlightGear

FlightGear is an open-source flight simulator that provides a three-dimensional simulation of the vehicle and its surroundings. Environment, terrain and objects can be altered and added by the user to customize the simulation. The 3D simulation provides a more suitable environment for simulating camera vision aspects of the system compared to 2D simulations available in Neptus.

4.5 BeagleBone Black

BeagleBone Black (BBB) is a high-functionality, low-cost development platform. In this project, BBBs will be used to run software such as DUNE aboard the multi-rotor.

4.6 PixHawk

PixHawk is an independent open-hardware project for autopilots. It supports multiple software and flight stacks, including ArduPilot, which is used in this project.

4.7 Tello

Tello is a light-weight drone developed by Ryze Technology. The Tello drone is a quadcopter equipped with a 5 megapixel (MP) camera and has a maximum flight time of 13 minutes [19]. The small drone is useful for testing the computer vision aspect of the system, where the larger S1000 requires more preparation and planning to use, as a licensed operator is required.

4.8 3DR Solo

The 3DR Solo is a medium-sized drone weighing 1.5 kg . It has a stated flight time of 20 minutes and a range of 800 meters [7]. It supports the use of several mountable cameras, and in this case, a GoPro Hero 4 is utilized. The 3DR Solo can be used for testing the camera vision aspect as well as autonomous navigation and maneuvering, as it can utilize both a BBB and a Pixhawk to run the ArduPilot software as well as DUNE.

4.9 DJI S1000 multirotor

After successful simulations and small scale tests, the main drone to be used is a DJI S1000 multirotor. The S1000 is an octocopter weighing approximately 4 kilograms. It has a flight time of up to 15 minutes and can take off with a load weighing up to 7 kilograms [10], which is theoretically sufficient to lift the MUG out of the water.

4.10 Camera

The camera acquired for use with the S1000 in this project is the oCam-1CGN-U-T developed by Withrobot in 2017 [16]. It is a 1 MP colour shutter camera that supports external triggering. The color output is Bayer RGB.

4.11 Markers

The marker used in this project is a 200 x100 mm rectangle in the colors of international orange (aerospace) and white. An 80 x80 mm ArUco code from the original ArUco dictionary is placed in the center. The marker is printed on regular paper and then laminated. An illustration of the marker is presented in Figure Figure 4.1.

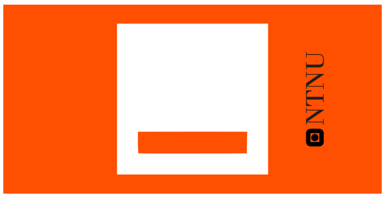


Figure 4.1: Custom marker.

4.12 Micro Underwater Glider

The concept for the MUG has been previously agreed upon during a meeting with representatives from NTNU and OsloMET. The concept is illustrated in Figure 4.2. The top part of the MUG is a wing, the same size as the custom marker. Ideally, the wing will stay just above sea level and allow for detection. The MUG weighs roughly 3.

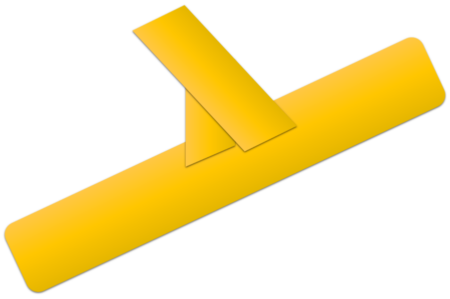


Figure 4.2: MUG concept.

4.13 System Overview

A general overview of the system setup and communication is displayed in Figure 4.3. In the shown configuration, DUNE is run either on a BBB or a PC. Using a BBB as a part of the drone's payload allows for onboard computing, which will cause less delay in the system, but will also have minimal computing power compared to an external computer. However, using external computing will result in more delay in the video stream and will potentially create a range restriction, depending on the communication method between the drone and the external computer. Exactly how the camera feed is sent to camera vision task depends on the drone and camera in use. A more specific figure is provided in Section 7.5 for the particular setup used in field tests.

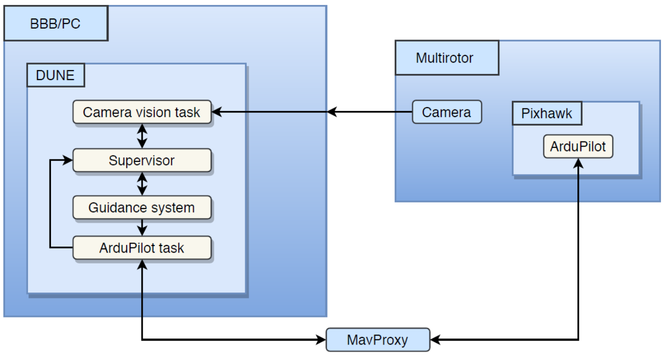


Figure 4.3: System overview.

Chapter 5

Modeling

5.1 Drone Dynamics

When modeling the dynamics of the system, the multi-rotor is assumed to be a rigid body, with center of mass in the geometric center and with six degrees-of-freedom (DOF). The translational equations of motion of the multi-rotor are then well established [20] and can be expressed in the body frame as:

$$\boldsymbol{F}^b = m(\dot{\boldsymbol{V}}^b + \boldsymbol{\omega}^b \times \boldsymbol{V}^b), \tag{5.1}$$

where $\boldsymbol{V}^b = [u, v, w]$ represents the drone's translational velocity in the body frame, $\boldsymbol{\omega}^b = [p, q, r]$ is the drone's rotational velocity expressed in body frame, m is the mass of the multi-rotor and the vector $\boldsymbol{F}^b$ represents all applied external forces.

The significant external forces involved in the system are the forces of gravity, thrust and drag which can be expressed as:

$$\sum \boldsymbol{F} = \boldsymbol{F}_g + \boldsymbol{F}_t + \boldsymbol{F}_d \tag{5.2}$$

The force of gravity is simply $mg\overrightarrow{k}$ in the inertial NED frame, and can be expressed in the body frame using the rotation matrix $\boldsymbol{R}_i^b = (\boldsymbol{R}_b^i)^T = (\boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi})^T$. The direction of the thrust force of the multi-rotor is entirely in the negative z-direction in the previously defined body frame. The drag force of a body moving at high speeds relative to the air around it can be modeled as

$$\boldsymbol{F}_d = \frac{1}{2}\rho \boldsymbol{V}_a^2 C_d A, \tag{5.3}$$

where $\boldsymbol{V}_a$ is the airspeed of the drone, ρ is the air density, C_d is the drag coefficient, and A is the surface area of the drone perpendicular to the airspeed direction.

Expressing the drag force in the body frame yields:

$$\sum \boldsymbol{F}^b = \boldsymbol{R}_i^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\sum_{i=1}^{N} F_{t\,i}^b \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\rho V_{ax}^2 C_d A_x \\ \frac{1}{2}\rho V_{ay}^2 C_d A_y \\ \frac{1}{2}\rho V_{az}^2 C_d A_z \end{bmatrix} \tag{5.4}$$

Multiplying in $\boldsymbol{R}_i^b$ gives

$$\sum \boldsymbol{F}^b = \begin{bmatrix} -sin(\theta)mg \\ sin(\phi)cos(\theta)mg \\ cos(\theta)cos(\phi)mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\sum_{i=1}^{N} F_{t\,i}^b \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\rho V_{ax}^2 C_d A_x \\ \frac{1}{2}\rho V_{ay}^2 C_d A_y \\ \frac{1}{2}\rho V_{az}^2 C_d A_z \end{bmatrix} \tag{5.5}$$

Note the negative sign for the thrust force as the z-axis is defined to be positive out of the belly/bottom of the drone. Plotting this into Equation 5.1 yields:

$$m(\dot{\boldsymbol{V}}^b + \boldsymbol{\omega}^b \times \boldsymbol{V}^b) = \begin{bmatrix} -sin(\theta)mg \\ sin(\phi)cos(\theta)mg \\ cos(\theta)cos(\phi)mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\sum_{i=1}^{N} F_{t\,i}^b \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\rho V_{ax}^2 C_d A_x \\ \frac{1}{2}\rho V_{ay}^2 C_d A_y \\ \frac{1}{2}\rho V_{az}^2 C_d A_z \end{bmatrix}, \tag{5.6}$$

which finally can be rewritten as:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} -sin(\theta)g \\ sin(\phi)cos(\theta)g \\ cos(\theta)cos(\phi)g \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{m}\sum_{i=1}^{N} F_{t\,i}^b \end{bmatrix} + \begin{bmatrix} \frac{1}{2m}\rho V_{ax}^2 C_d A_x \\ \frac{1}{2m}\rho V_{ay}^2 C_d A_y \\ \frac{1}{2m}\rho V_{az}^2 C_d A_z \end{bmatrix} - \begin{bmatrix} qw - rv \\ ru - pw \\ pv - qu \end{bmatrix} \tag{5.7}$$

For the rotational equations of motion expressed in the body frame, the following equation for the sum of moments is well established [20]:

$$\boldsymbol{J}\dot{\boldsymbol{\omega}}^b + \boldsymbol{\omega}^b \times \boldsymbol{J}\boldsymbol{\omega}^b = \boldsymbol{M}^b, \tag{5.8}$$

where $\boldsymbol{J}$ is the moment of inertia tensor and $\boldsymbol{M}^b$ is the sum of external moments. Due to the drone's geometric symmetry in both the $i^b k^b$-plane and the $j^b k^b$-plane, the inertia products $J_{xz} = J_{zx}$, $J_{xy} = J_{yx}$ and $J_{yz} = J_{zy}$ all equal zero. This gives the following expression for the sum of external moments [28]:

$$\boldsymbol{M}^b = \begin{bmatrix} J_{xx}\dot{p} \\ J_{yy}\dot{q} \\ J_{zz}\dot{r} \end{bmatrix} + \begin{bmatrix} (J_{zz} - J_{yy})qr \\ (J_{xx} - J_{zz})pr \\ (J_{yy} - J_{xx})pq \end{bmatrix} \tag{5.9}$$

5.2 Ocean current

Ocean current is a prominent disturbance concerning the horizontal position of the MUG. As potential field testing in water will occur in Trondheimsfjorden, both general theory on ocean current characteristics as well as explicit data from Trondheimsfjorden are taken into account. Since the MUG will drift at the ocean surface, deep-water currents are disregarded, and only surface current characteristics are considered.

A surface current travels at nominal speeds ranging from $0.05\,\mathrm{m\,s^{-1}}$ to $0.5\,\mathrm{m\,s^{-1}}$ [21]. The relevant testing area in Trondheimsfjorden is well guarded from larger ocean currents, e.g. the Norwegian Current, by land masses and a winding fjord path from the open ocean. This is partly the reason for the relatively slow surface currents present. Data from *yr* [9] suggests the average surface current velocity is roughly $0.05 - 0.15\,\mathrm{m\,s^{-1}}$. The data also indicates a slowly varying current speed and direction. This allows for the current to be modeled as a constant velocity plus noise for short simulations. The surface current velocity is modeled as follows:

$$\begin{bmatrix} V_{c_n} \\ V_{c_e} \\ V_{c_d} \end{bmatrix} = \begin{bmatrix} C_n \\ C_e \\ 0 \end{bmatrix} + \begin{bmatrix} S_g(t) \\ S_g(t) \\ 0 \end{bmatrix}, \tag{5.10}$$

where $\boldsymbol{V_c}$ is the surface current velocity expressed in NED frame and $S_g(t)$ is a Gaussian white noise process with variance $\sigma^2 = 0.0001$.

5.3 Waves

Modelling of waves is essential when creating a controller for a ocean surface vessel. However, since waves will only affect the non-controlled MUG in this project, a simple model for the wave-induced velocity of the MUG is proposed. In this model, the approximation of Stoke's drift velocity for deep-water waves is implemented, with a modification, and the z-position of the MUG is assumed to follow the z-position of the sea surface at the same location. Ocean waves are also approximated to noisy sine waves. The wave-induced velocity is expressed in the NED frame and is modeled as:

$$\boldsymbol{V}^i_{wi} = \begin{bmatrix} u_s cos(\alpha) \\ u_s cos(\beta) \\ a\omega cos(\omega t) \end{bmatrix}, \tag{5.11}$$

where $u_s = C\omega k a^2 e^{2kz}$ is the modified horizontal component of the Stoke's drift velocity for deep-water waves. $\omega = \frac{2\pi}{T}$ is the angular wave frequency, T is the wave period, $k = \frac{2\pi}{\lambda}$, is the wavenumber, a is the wave amplitude, λ is the wavelength and z is the vertical axis, where waves oscillate around the mean level $z = 0$. The addition of variable C is the modification to the original Stoke's drift approximation. C is a scaling factor, from 0 to 1, since the effect of Stoke's drift will vary with depth and sea state. The variables α and

β are the angles between the direction of wave propagation and north, and the direction of wave propagation and east, respectively. The last element of the wave-induced velocity vector is the derivative of the z-position of an object solely moving vertically as a result of floating on waves on the free ocean surface.

Figure 5.1 depicts graphs of the ocean surface z-position at a fixed point in space as well as the wave-induced z-velocity. Figure 5.2 shows the wave-induced velocity the MUG experiences in the north direction, assuming the direction of wave propagation is northwards, i.e., $\alpha = 0$. The parameters used to generate these figures are $a = 0.2 + S_g(n)$ where $S_g(n)$ is a Gaussian white noise process with a standard deviation of 0.03. The wave period $T = 3\,\mathrm{s}$ and wavelength $\lambda = 8\,\mathrm{m}$. These values are based on the findings in *A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii* [32]. The scaling factor C is set to 0.1.

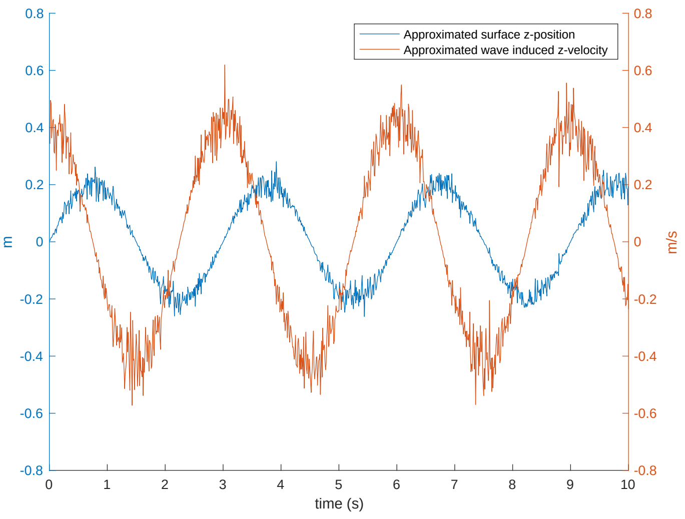


Figure 5.1: Ocean surface z-position (blue) and wave induced z-velocity (orange).

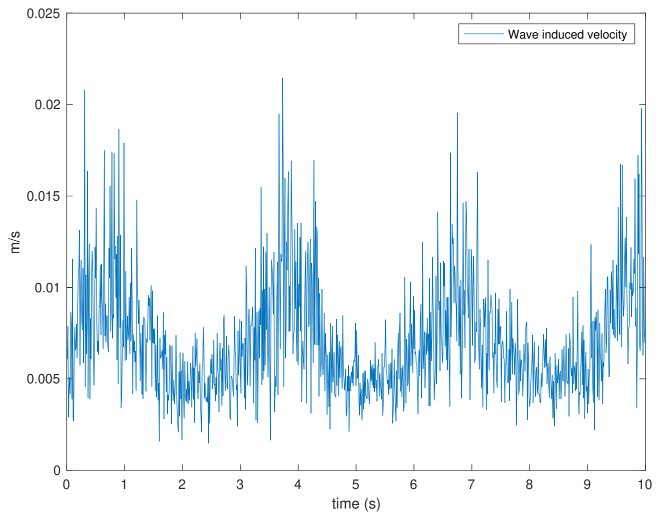


Figure 5.2: Northward wave-induced velocity experienced by the MUG

5.4 Wind

Wind is perhaps the most significant disturbance affecting the UAV, but it will have close to no impact on the MUG other than the wind-induced waves and currents. This is due to the fact that nearly all the volume and mass of the MUG will be below surface level. Only the previously described wing of the MUG will be at surface level, or slightly above. Hence, the horizontal component of the wind will have limited surface area to hit, causing a negligible wind force on the MUG.

The wind velocity can be modeled as the sum of a constant wind velocity and a varying one, representing gusts. This model is more realistic for wind speeds of $10\,\mathrm{m\,s^{-1}}$ or more, as real-life low-speed winds tend to change direction more frequently [20]. The constant wind component, $\boldsymbol{V}_{w_c}$, is expressed in the NED frame while the gust component, $\boldsymbol{V}_{w_g}$ is expressed in the drone's body frame.

$$\boldsymbol{V}_w = \boldsymbol{V}_{w_c} + \boldsymbol{V}_{w_g} \tag{5.12}$$

In order to model the wind gusts, the transfer functions from the Dryden wind turbulence model in *Small Unmanned Aircraft* [20] are used, with a modification. The original model assumes the UAV body x-axis as the main direction of motion. With this baseline, the transfer function for gust along the x-axis is of order one, while the other transfer functions are second order. However, the UAV body x-axis is not necessarily the main direction of motion, nor the direction in which the copter travels at the highest speeds. Therefore, the original transfer function for gust in the body x-axis in *Small Unmanned Aircraft* has been modified, and is displayed in Equation 5.13.

$$H_u(s) = \sigma_u \sqrt{\frac{3V_a}{L_u}} \frac{\left(s + \frac{V_a}{\sqrt{3}L_u}\right)}{\left(s + \frac{V_a}{L_u}\right)^2} \tag{5.13a}$$

$$H_v(s) = \sigma_v \sqrt{\frac{3V_a}{L_v}} \frac{\left(s + \frac{V_a}{\sqrt{3}L_v}\right)}{\left(s + \frac{V_a}{L_v}\right)^2} \tag{5.13b}$$

$$H_w(s) = \sigma_w \sqrt{\frac{3V_a}{L_w}} \frac{\left(s + \frac{V_a}{\sqrt{3}L_w}\right)}{\left(s + \frac{V_a}{L_w}\right)^2} \tag{5.13c}$$

In Equation 5.13, V_a is the nominal airspeed of the drone, σ_i are the intensities of the turbulence in each body axis, and L_i are the spatial wavelengths. The parameter values depend on the altitude of the drone as well as the desired level of turbulence. Table 5.1, based on Table 4.1 from *Small Unmanned Aircraft*, shows parameter values for low to medium altitudes with light to moderate turbulence.

Table 5.1: Dryden gust parameters

Gust description	*Altitude (m)*	*L_u=L_v(m)*	*L_w (m)*	*σ_u=σ_v (m/s)*	*σ_w (m/s)*
Low altitude, light turbulence	50	200	50	1.06	0.7
Low altitude, moderate turbulence	50	200	50	2.12	1.4
Medium altitude, light turbulence	600	533	533	1.5	1.5
Medium altitude, moderate turbulence	600	533	533	3.0	3.0

The Dryden transfer functions in Equation 5.13 each take a white noise signal as input, in order to generate wind gust velocities in the body axes. These gust components are then added to the constant wind components, which first have been rotated from NED frame to body frame. The sum of these components amounts to the total wind velocity, as described in Equation 5.12. Figure 5.3 illustrates the process. The figure is based on Figure 4.9 in *Small Unmanned Aircraft* [20].

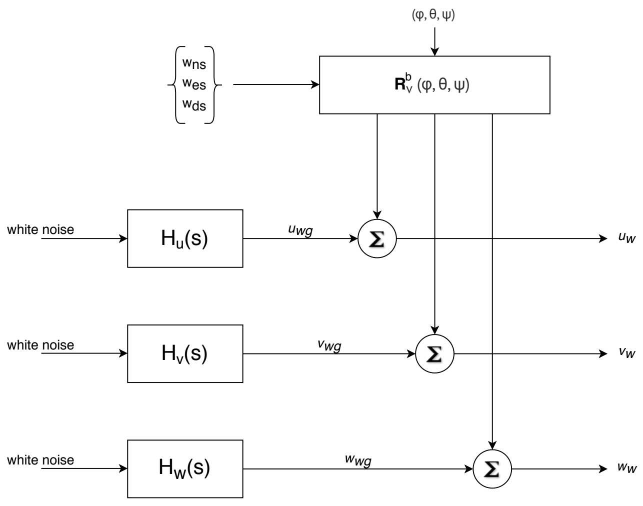


Figure 5.3: Total wind velocity in the body frame as the sum of constant wind and gusts. The gust is generated by feeding white noise signals into the Dryden transfer functions.

Figure 5.4 shows the simulated total wind velocity using the Dryden model for turbulence. A constant wind of $10\,\mathrm{m\,s^{-1}}$ is added to each wind gust element. The turbulence is generated by passing Gaussian white noise signals with unit variance through the Dryden transfer functions.

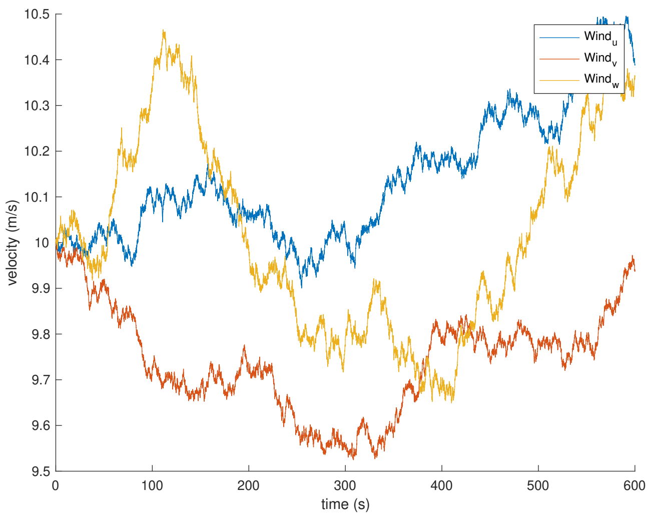


Figure 5.4: Response of the modified Dryden turbulence model

Chapter 6

Implementation

6.1 OpenCV

As mentioned in Chapter 1, the open-source computer vision and machine learning software library, OpenCV, provides functionality for ArUco marker detection and pose estimation. The software is installed and built on a Linux machine as described by the OpenCV documentation [15], with the `OPENCV_EXTRA_MODULES_PATH` included in order to use the ArUco module. OpenCV defines the ArUco marker frame as follows: the z-axis is perpendicular to the marker plane, centered in the middle of the marker and pointing outwards. The y-axis points upwards, and the x-axis to the right, completing the right-handed coordinate system [14]. Figure 6.1 illustrates this.

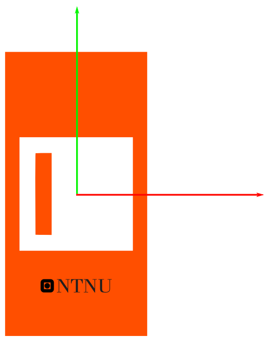


Figure 6.1: Custom marker frame. X-axis in red, y-axis in green and z-axis pointing out.

6.2 Camera Calibration

In order for the camera images to be useful in the detection and pose estimation of the ArUco markers, any distortion present in the images must be minimized. OpenCV recommends using a CharUco board when calibrating cameras with their built-in functions [12]. A CharUco board is the result of merging a regular chessboard pattern and ArUco markers. The board is generated by code utilizing existing functionality in OpenCV, where dimensions, square sidelengths and ArUco dictionary is set.

```
using namespace cv::aruco;

Ptr<CharucoBoard> m_charuco_board =
   CharucoBoard::create(5, 7, 0.024, 0.012, m_aruco_dictionary);
```

The same dictionary as the custom NTNU marker utilizes, ArUco ORIGINAL, is chosen for the CharUco board. The markers have a sidelength of $1.2\,\mathrm{cm}$ and the chess squares $2.4\,\mathrm{cm}$. Figure 6.2 shows the generated board.

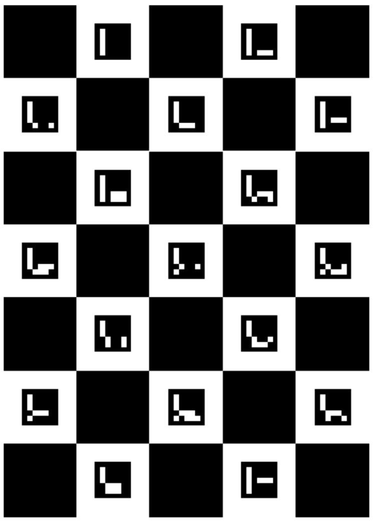


Figure 6.2: CharUco board.

To find accurate values for the camera parameters and distortion coefficients, at least ten images of the CharUco board, at varying angles and distances, are required. In theory, only two images are needed for sufficient accuracy, but due to noise present in the images, a higher quantity is necessary to achieve precise parameter values. Since the size of the chess squares and marker squares are known, the parameters can be determined using basic geometrical equations. OpenCV describe their algorithm in detail on their website [13].

A task is created in DUNE to handle camera calibration. The task takes in the video feed from a given camera as input and captures each frame using the OpenCV video capture object `cv::VideoCapture`. The task then runs `cv::aruco::detectMarkers()` for each captured frame. If a frame has a sufficient number of detected markers, specified by a threshold parameter, the frame is stored and will later be used to calibrate the camera. The code will then sleep for 1.5 seconds, giving the user time to move either the camera or the CharUco board to avoid reusing the same frame. When a predefined quantity of acceptable frames have been found and stored, the `calibrateCameraCharuco()` function is run with these frames as input. The function calculates the intrinsic parameters and distortion coefficients of the camera, which are then saved. The calibration function also returns the re-projection error, providing a metric to assess the outcome of the process. The closer to zero the re-projection error is, the better the result. When a calibration returns an acceptably low value for the re-projection error, the newly stored intrinsic parameters and distortion coefficients will be used to undistort future images. These parameters and coefficients only need to be found once for each camera and can be utilized as constants in later applications. Undistorting an image is supported by the OpenCV function `undistort()`.

6.3 Filtering

As mentioned in Section 4.11, the custom marker has the distinct international aerospace orange color, which is highly visible on the ocean surface. A threshold segmentation method is applied to each frame to extract the marker from the environment:

```
using namespace cv;

(...)

inRange(Mat frame_HSV, Scalar(H_low, S_low, V_low),
   Scalar(H_high, S_high, V_high), Mat frame_filtered);
```

Where `frame_filtered` is the filtered image output given the scalar ranges for hue, saturation and value. With ideal threshold ranges, the output image will show the aerospace orange color as white and everything else as black.

An experiment to determine optimal filter values has been conducted prior to this project by postdoctoral researcher Artur Zolich. The experiment entails setting up the oCam camera in an environment with natural light, with the custom NTNU marker in the camera view. An image was taken automatically every 5 minutes for 7 hours. For each image, a section of the custom marker only containing the aerospace colour is cropped and processed. Disregarding outliers, the resulting filter values are shown in Table 6.1 and the camera settings in Table 6.2.

Table 6.1: HSV filter values

	H	*S*	*V*
Low	1	78	32
High	30	205	255

Table 6.2: oCam Settings

Exposure (ms)	*Gain*	*WB Blue*	*WB Red*
1	33	166	122

The ideal filter values depend on both camera settings and light conditions. As the camera settings can be set manually to fixed values, the primary variable is lighting. A constant filter range will not be effective in largely varying light conditions, which is to be expected in local experiments given Trondheim's weather. Therefore, an automatic filter calibration function is created.

The `auto_filter(cv::Mat frame_bgr)` function takes the current blue-green-red (BGR) frame from the given video feed as input. It then retrieves the red, green and blue colour values from four percent of the pixels, evenly spaced out, and compares them to user-set limits. The red-green-blue (RGB) values for aerospace orange is 255-79-0 [1], however, slightly different values are to be expected due to several variables such as type of printer, toner ink, lamination sheet, screen colour calibration and light conditions. If the pixel's RGB-values are within the predefined limits, the pixel's hue, saturation and value values are stored in their corresponding vectors. The `auto_filter(...)` function continuously averages these vectors and sets minimum and maximum values for the filter parameters by adding and subtracting an offset to each attribute average:

```
using namespace std;

(...)

H_avg = accumulate(H_vec.begin(), H_vec.end(), 0) / H_vec.size();
H_low = (H_avg - H_offset > 0) ? H_avg - H_offset : 0;
H_high = (H_avg + H_offset < 255) ? H_avg + H_offset : 255;
```

6.4 Color detection

In order for pose estimation to be feasible, the distance between the camera and the marker must be relatively small. For a search method to be effective, the drone must fly at an altitude significantly higher than this distance. Therefore, a method to detect the marker at higher altitudes is implemented.

Color detection is the basis of the implemented method, which allows utilization of the already realized HSV filter. Once an image frame is filtered, it needs to be undistorted in order for the marker to achieve the correct shape, dimensions, and position. This is realized through the OpenCV function `cv::undistort()`. Next, the filtered and undistorted image moments are computed, again using OpenCV functionality with `cv::moments`. The combined centroid of every pixel that passes through the HSV filter is computed utilizing the image moments. The calculated centroid is then expressed in image coordinates and used to derive a low estimate of the horizontal distance between the camera and the marker. The derivation consists simply of dividing the distance from the object to the image center along the x-axis, expressed in number of pixels, by the image width. The same procedure is performed for the y-axis and image height. This way, the maximum estimated distance is $0.5\,\mathrm{m}$ along each axis, which avoids a large-valued input to the controller and in turn avoids high velocities and quick accelerations.

6.5 ArUco Detection and Pose Estimation

After the input image has been filtered successfully, the ArUco marker within the custom marker can be detected using the `cv::aruco::detectMarkers()` function. This enables the pose of the MUG in relation to the camera to be determined using `cv::aruco::estimatePoseSingleMarkers()`. The former function outputs the detected corners, which the latter uses as input. The pose estimation function takes in the marker size as well as previously determined camera parameters to counteract any distortion present and return accurate pose estimates. This pose estimation is returned as a vector of translational difference, `tvecs`, and a vector of rotational difference, `rvecs`. The translation will be in the same units as that of the marker size parameter.

According to OpenCV [11] the returned vectors are ones that transform the center of the marker in the marker coordinate system, to the camera coordinate system. However, OpenCV fails to mention which frame the translational and rotational vectors are expressed in; thus, an experiment must be conducted to determine this.

The filtering, detection, and pose estimation processes are all combined into a single DUNE task; `Camera/Detection/Tasp.cpp`. When the camera is close enough to the custom marker for it to be detected, the returned `tvecs` and `rvecs` are averaged over three iterations. This averaging is performed to limit the effect of miscalculations in pose estimation due to image noise, sudden movement and other disturbing elements. The averaged values are dispatched using the IMC message `IMC::RelativeState` which is consumed by the state machine described in Section 6.8. The color detection process also uses the `IMC::RelativeState` message to dispatch the estimated distances to the state machine. The color detection process is optional, as it is not needed for low altitude flights. Hence the DUNE task has a boolean argument representing whether or not to utilize this process. If color detection is used, the detection task switches to ArUco detection when the copter descends to a user-set altitude which defaults to $1.2\,\mathrm{m}$. At this altitude, it is assumed that the ArUco ID can be detected, and thus pose estimation can commence.

6.6 Wave simulation

The ocean current model and wave model described in Section 5.2 and 5.3 respectively, are implemented in a single DUNE task. Ocean current is actualized with two task arguments, one for the current's north-travelling component and one for the east component. These arguments are expressed in meters per second and remain constant during run time. The noise-inflicted wave amplitude in the approximation of Stoke's drift velocity for deep-water waves is realized using built-in functionality in C++. The amplitude is set using the custom function, as shown below:

```
using namespace std;

void set_noisy_amp(double std_dev)
{
  double mu = 0;
  default_random_engine generator(random_device{}());
  normal_distribution<double> dist(mu, m_args.std_deviation);
  m_amp = m_args.wave_amplitude + dist(generator);
}
```

The use of `default_random_engine` and `normal_distribution` ensure a Gaussian white noise process, as described in the model in Section 5.3

6.7 Guidance system

The drone dynamics described in Section 5.1 are handled in configuration files provided by the NTNU DUNE repository, and in the ArduPilot software. Therefore, they do not need to be explicitly considered when creating a guidance system.

A constant bearing guidance system identical to that described in Section 3.4 has previously been implemented as a task in DUNE by civil engineer Jens Joberg. A constant bearing guidance system is a simple yet effective method and is deemed suitable for this project. It has the advantage of being predictive over the other methods described in 3.4, which could prove essential when landing on a small moving target.

The constant bearing guidance task is modified by adding a boolean variable to the IMC message `IMC::ConstantBearingTarget`. The added variable represents whether or not the target position is relative to the drone. An advantage of using relative position is that the NED position of the drone does not need to be known. Furthermore, the output of the detection task is relative position and orientation.

The highest demand for accuracy is when the drone is maneuvering, and especially landing, based on the output of the marker detection task. This output is dependant on the pose of the camera frame, which is always a simple, constant transformation from the drone's body frame. Because of this, velocity control is implemented in the body frame of the drone, as opposed to the NED frame. The transformation from camera frame to NED frame would vary depending on the orientation of the drone. Since the heading angle would need to be updated every time the task executes, this solution would have a slightly slower response.

6.8 State Machine

A state machine is implemented as a DUNE task in order to supervise the stages of each flight systematically and safely. This task serves as the central governing task, sending and receiving messages to and from other smaller tasks as well as communicating with the DUNE ArduPilot task.

The state machine is illustrated in Figure 6.3 as a flow chart. Each state and their transition conditions are detailed below.

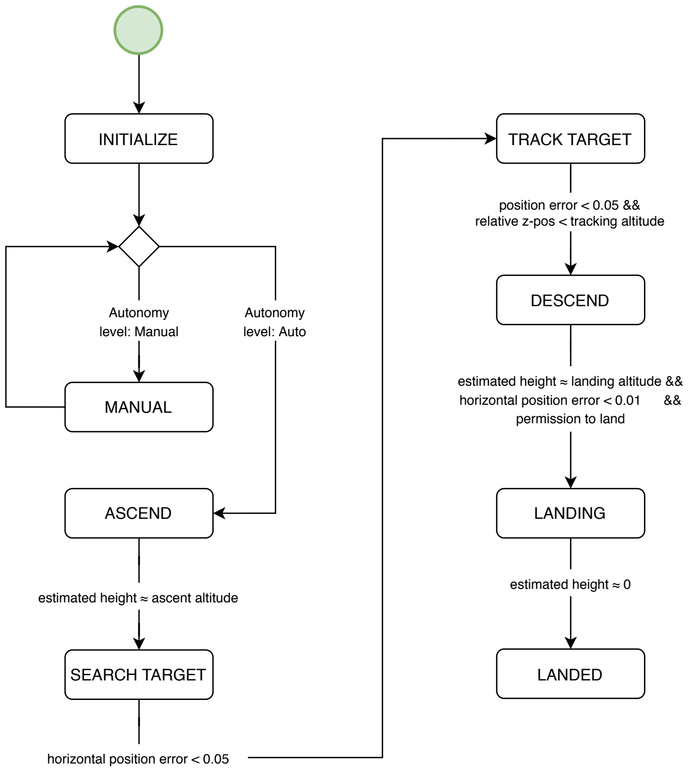

INITIALIZE
Autonomy level: Manual
Autonomy level: Auto
MANUAL
ASCEND
estimated height ≈ ascent altitude
SEARCH TARGET
horizontal position error < 0.05
TRACK TARGET
position error < 0.05 && relative z-pos < tracking altitude
DESCEND
estimated height ≈ landing altitude && horizontal position error < 0.01 && permission to land
LANDING
estimated height ≈ 0
LANDED

Figure 6.3: State machine with transition conditions.

When controlling a multi-rotor drone, simulated or real, ArduPilot requires the copter to be in *Guided* mode in order to perform a takeoff. As there is currently no implementation in the DUNE ArduPilot task to set the copter mode, a manual mode set to *Guided* mode and a manual takeoff need to be performed prior to the initialization of the state machine.

MANUAL
During initialization, the current state is set to MANUAL. In this state, no events will occur as it only contains a return line. As shown in the state machine figure, the transition from MANUAL to ASCEND takes place when the autonomy level of the drone changes to *Auto*. The state machine continuously listens for the IMC message `IMC::AutoPilotMode` and inspects its autonomy level upon receipt. The IMC message is sent from the DUNE ArduPilot task which frequently receives a Mavlink message from ArduPilot running on the copter. The act of manually setting the copter to *Guided* mode is what triggers a change in the autonomy level.

While in the MANUAL state, the state machine will not control or affect the drone in any way, allowing for a pilot to manually control the drone.

ASCEND
After transitioning to the ASCEND state, the IMC message `IMC::ControlLoops` is used to enable velocity control in DUNE's ArduPilot task. Furthermore, a target z-value is set and dispatched utilizing the IMC message `IMC::ConstantBearingTarget`, which is received by the constant bearing guidance task described in the previous section.

Upon reaching an altitude within a given range of the target altitude, the state transitions to SEARCH TARGET.

SEARCH TARGET
In the SEARCH TARGET state, the copter remains at its current altitude. It is assumed that the target marker is within the camera view in this state, facilitating the detection task to detect the marker and send `IMC::RelativeState` messages, which are received by the state machine. Based on these messages, a horizontal target position is set. Since the distances acquired in the detection task are relative to the drone, the `IMC::ConstantBearingTarget` variable `relative` is set to true.

In the event that the marker is outside of the camera view, the state machine will not receive any updates on the target position, causing the copter to hover at a fixed position. To prevent this, a function, `check_for_camera_update()`, is created and implemented. The function's purpose is to increase the altitude of the drone if a target update has not been received within a given time period. This is especially useful when descending to low altitudes, as the marker is then more likely to leave the camera view. The function is present in all of the states below unless explicitly stated otherwise.

When the horizontal error is reduced to $0.05\,\mathrm{m}$, transition conditions are met, and the state is set to TRACK TARGET.

TRACK TARGET
This state operates similarly to SEARCH TARGET, except that a vertical target is also set.

The vertical target is set to the task argument `m_args.tracking_altitude`, which defaults to a value of 1.0 meters, but can be altered in the configuration file.

As mentioned in Section 6.5, if active, the color detection process is deactivated at an altitude of 1.2 meters. This allows the pose estimation process to return the relative distance, as opposed to the color detection, resulting in more accurate values used in the transition conditions of this state.

When the drone has a total position error of less than $0.05\,\mathrm{m}$, the transition to the DESCENDING state takes place.

DESCENDING
In the DESCENDING state, all translational targets are set. The copter's x- and y-targets are set based on the relative position between the camera and the marker, and the z-target is set to `m_args.landing_prep_altitude`. This argument defaults to 0.3 meters, but is user-configurable.

Because the z-error is much larger than the x- and y-error, the z-error is prioritized by the guidance system. This produces a vertical descend rate which could cause the horizontal error to increase, and potentially cause the marker to disappear from the camera view. To avoid a cycle where the drone descends, then loses sight of the marker, and finally ascends with a similar horizontal error, extra conditions must be fulfilled during the descent. While descending, the horizontal error must be equal to or less than $0.04\,\mathrm{m}$. Otherwise, the z-target is simply set to the current z-position, allowing the horizontal error to be the focus.

When the drone is within two centimeters of the target altitude, the horizontal error is within one centimeter, and the `m_args.permission_to_land` variable is true, the state transitions to LANDING.

LANDING
Upon complete transition to the LANDING state, the target z-position is set to the value of the task argument `m_args.landing_altitude`. The main reason for this is that basing the target z-position on the ArUco detection output is not feasible at certain, low altitudes. At these altitudes, approximately 0.15 meters, the entire ArUco marker does not fit in the camera view and can therefore not be detected. This leads to no new z-target and would force the copter to ascend due to no received detection data. Instead, the target is constant, (although the relative target varies) and does not come from the detection task, which allows the `check_for_camera_update()` function to be omitted in this state.

Another advantage of setting the landing altitude based on the task argument is that the approximate landing speed can be observed without the high risk involved with landing on water or land. The x- and y-targets are still set based on detection data and will not update after the last reading

When verifying if the copter has indeed landed, a simple altitude inspection is performed and compared to the landing altitude argument. If the difference is within a given value, currently 0.03 meters, the copter is considered to have landed. This could potentially cause the drone to hover over the marker when instead a touch-down landing is desired. Still, when the copter is within the described bounds, the state machine transitions to its final state; LANDED.

LANDED
The LANDED state simply stores and displays the translational error in each horizontal axis. This error is the last recorded error by the pose estimation function. Hence, it will not coincide perfectly with measured results as the drone will have some horizontal translation in the landing phase, especially if wind is present.

There are currently no available transitions from the LANDED state, and manual takeover is necessary to pilot the drone.

6.9 FlightGear

FlightGear is utilized in order to perform simulations which implement camera detection methods. An object model that corresponds to the custom NTNU marker is provided by Artur Zolich. The marker object and drone are placed next to each other on an ocean-blue surface, meant to approximate color conditions at sea. The origin of the camera view is set $0.1\ \mathrm{m}$ below the bottom of the drone facing downwards.

The FlightGear user interface, *Phi*, implements a two-way data binding between FlightGear and the host machine's internal HTTP web server. One of the features of Phi is that the FlightGear camera view is available at the user's local address: `localhost:8080/screenshot`. This allows for simple frame capturing in DUNE using the local address as the path to the video stream.

6.10 Logging

DUNE has some built-in functionality for logging, including logging the output from the terminal where the DUNE configuration file is run. However, a separate logger is created and implemented as a periodic DUNE task in order to acquire a more comprehensive log, containing all relevant data. The log is timestamped and includes the values of every relevant member variable of all IMC messages received during run-time. As separate IMC messages are received at different times and with different frequencies, their variables are updated at their corresponding frequency, but only stored to file at a constant rate. The logging frequency's default value is 1 Hz, but this can be altered by setting the value of the `Execution Frequency` argument in the DUNE configuration file.

6.11 Ryze Tello

The Tello drone is useful in small scale testing with regards to the camera vision aspect of the project. The drone has it's own WiFi which users can connect to. When connected, simple command messages over UDP can be used to control the drone. Sending ”command”, followed by ”streamon” will enable the camera feed from the drone. After being enabled, the drone streams the camera feed over UDP at a specific port, which the connected user can listen to. This video feed is compressed with H.264 encoding, also called Advanced Video Coding (AVC). Although the drone has minimal hardware on board, using this compression method allows for transmitting frames at a rate of 30 frames per second at a resolution of 720p, which is sufficient for image analysis and testing purposes [18].

Initially, there was an issue with delay caused by OpenCV when trying to interpret the H.264 encoded video stream over UDP directly. This lead to delays of approximately 5 seconds, which is insufficient for the implemented control and guidance systems. The control systems are critically dependent on a non-delayed information stream for making correct control decisions, especially in field testing. To combat this delay, a solution was found by manually decoding the H.264 encoded stream, converting the decoded data to the OpenCV `cv::Mat` type and then using OpenCV for regular image analysis.
This method removed the delay from the stream and close to real-time performance was achieved, at the cost of occasional digital distortion which could prove to be problematic.

After several attempts to implement the manual H.264 decoder in DUNE, none were successful. As the Tello drone does not use ArduPilot, the control systems made for it would not be easily transformable to use ArduPilot through Mavlink. Because of this, no more attempts to include the decoder are made, and instead, FlightGear simulations become the ultimate step before field testing with a larger drone.

Chapter 7

Experiments and Results

7.1 Camera Calibration

To calibrate the Tello camera, a modified version of the calibration task described in Section 6.2 and the 5x7 CharUco board are used. The modification is simply changing the input to images instead of using a video stream and looking at individual frames. With this method the returned re-projection error is $e_r = 0.43$ and the camera matrix and distortion coefficients are found to be:

$$P = \begin{bmatrix} 912 & 0 & 480 \\ 0 & 917 & 362 \\ 0 & 0 & 1 \end{bmatrix} \quad K = \begin{bmatrix} -0.016 & -0.126 & 0.001 & 0.002 & 0.433 \end{bmatrix} \tag{7.1}$$

The built in 720p FaceTime HD web camera of a 2014 MacBook Pro is also calibrated. Calibration is performed using the 5x7 CharUco board and the calibration task described in Section 6.2. In this instance the total re-projection error is $e_r = 0.29$ and the camera matrix and distortion coefficients are found to be:

$$P = \begin{bmatrix} 1004 & 0 & 644 \\ 0 & 1004 & 367 \\ 0 & 0 & 1 \end{bmatrix} \quad K = \begin{bmatrix} 0.113 & -0.252 & 0.002 & 0.002 & 0.121 \end{bmatrix} \tag{7.2}$$

Finally, the 3DR Solo camera is calibrated. The resulting camera matrix and distortion coefficients are displayed in Equation 7.3. An image taken with the GoPro camera, as well as the undistorted version of the same image, are depicted in Figure 7.1. By simple visual inspection, the undistortion process appears successful, deeming the camera parameters accurate. However, no further examination is performed to quantify the accuracy of the distortion parameters.

$$P = \begin{bmatrix} 762 & 0 & 651 \\ 0 & 767 & 387 \\ 0 & 0 & 1 \end{bmatrix} \quad K = \begin{bmatrix} -0.330 & 0.201 & -0.008 & 0.003 & -0.096 \end{bmatrix} \tag{7.3}$$

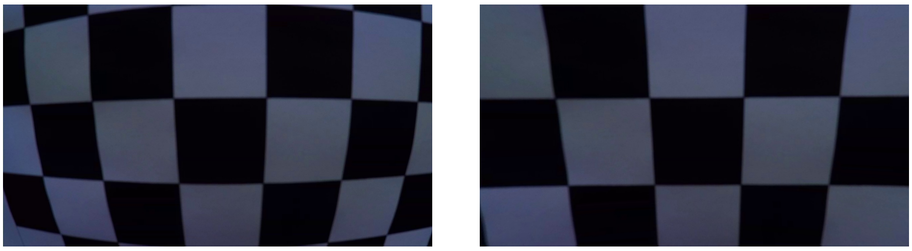

(a) Original image. **(b)** Undistorted image.

Figure 7.1: Original and undistorted image from the GoPro Hero 4 camera.

The matrices and vectors in Equation 7.1, 7.2 and 7.3 are stored locally for later use in the pose estimation task.

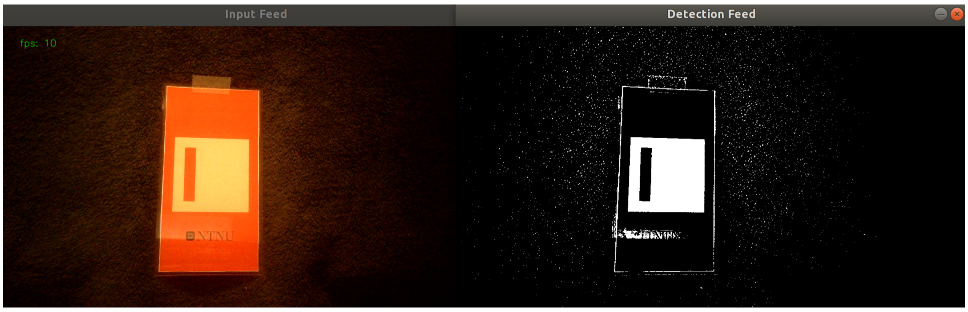
7.2 Filtering and Detection

The static HSV filter values stated in Table 6.1 are tested against the auto-filter, using the Macbook's built-in camera. As both filters are tuned based on images from the oCam camera, ideal performance is not expected, but the results indicate the versatility of the filters. The built-in FaceTime camera is using its default settings, which are non-configurable without third-party software. The HSV offsets mentioned in Section 6.3 are found through trial and error, and are presented in Table 7.1.

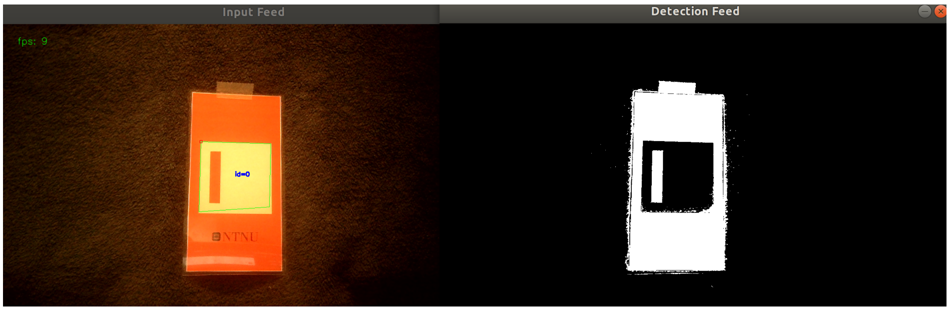
Table 7.1: Experimentally deducted offset in hue, saturation and value.

Hue	*Saturation*	*Value*
30	90	120

The result of both filter options in an indoor environment with yellow bulbs is illustrated in Figure 7.2. The marker is placed on a blue-green background to provide high contrast between itself and the environment around it. This is meant to approximate the high contrast conditions which will be present at sea.



(a) Using static filter values.



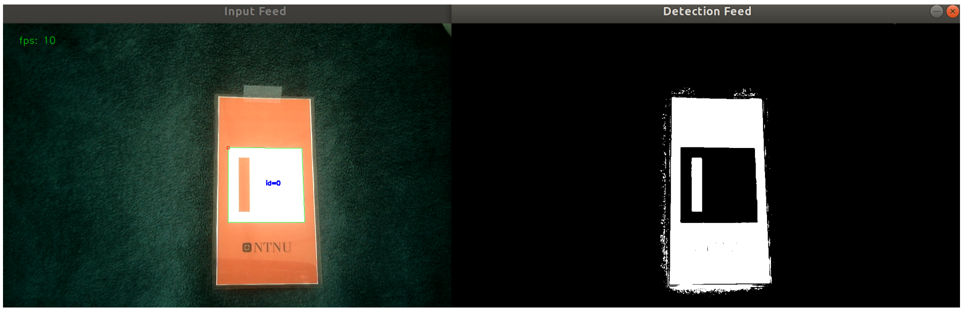
(b) Using automatic filter values.

Figure 7.2: Filter input and output in yellow-light illuminated environment.

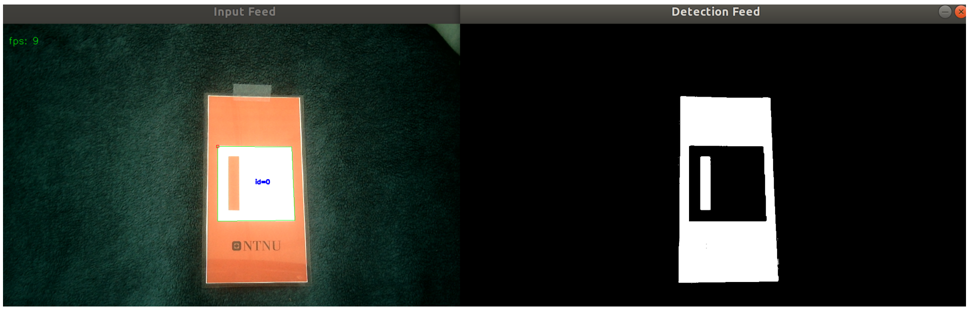
As seen in Figure 7.2a, the static filter values do not filter the image correctly, and no detection can occur. The blacks and whites of the marker are in fact inverted compared to the desired filtering.

The automatic filter values perform better in yellow light, and the marker is successfully identified. However, the green square outline in Figure 7.2b reveals that the estimated location of the marker is not highly accurate. Still, all intended areas of operation are in natural light environments, and therefore the above results are insignificant.

Figure 7.3 depicts the results of the two filter methods when the marker is placed in a natural light environment. Significant improvement is made with both methods, as can be seen by the highly accurate estimated location, illustrated by the green outline in each input image. There is, however, a noticeable amount of noise present in the output using the static filter values. This amount of noise with an optimized background bodes poorly for using this filtering method in conjunction with navigation based on color detection. The automatic filter method on the other hand, creates no visible image noise.



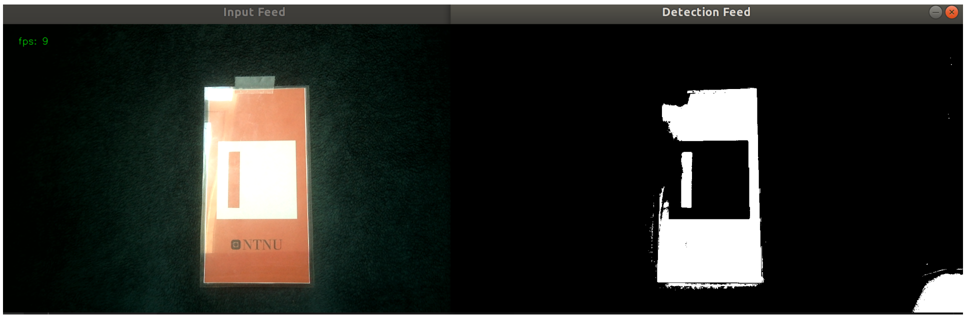
(a) Using static filter values.



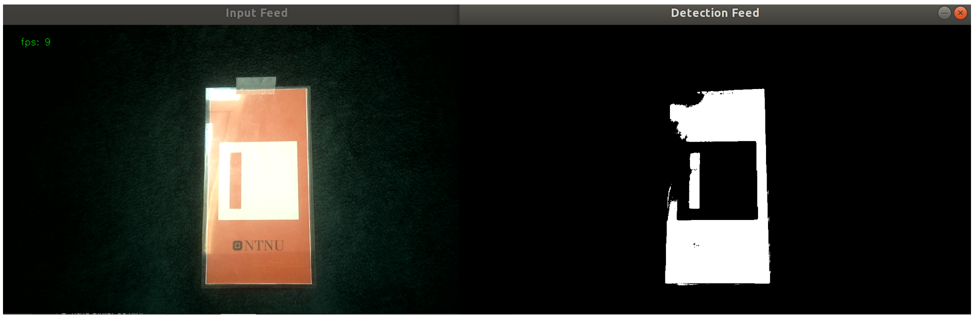
(b) Using automatic filter values.

Figure 7.3: Filter input and output in natural light environment.

A concerning discovery is made during testing regarding light reflection. The marker's lamination is necessary to protect it from dissolving in salty ocean waters, however, it has a significant drawback. The lamination causes the light-reflection from the marker to be substantially more prominent. As light reflects off the marker and hits the camera lens, it is interpreted as color, typically white or yellow, depending on the light source. Even a small amount of reflection can cause detection problems, illustrated in Figure 7.4.



(a) Using static filter values.



(b) Using automatic filter values.

Figure 7.4: Filter input and output in natural light environment, with light glare.

The auto-filter is also tested using recorded video from the GoPro camera in a flight test with the custom marker. A screenshot of the camera feed and filter output is captured and depicted in Figure 7.5. The figure shows successful filtering with minimal undesired areas passing through the filter. A light blue line is drawn from the image center to the centroid of the area that passes through the filter thresholds. The line acts as a visual reference for the target location sent to the constant bearing guidance task.

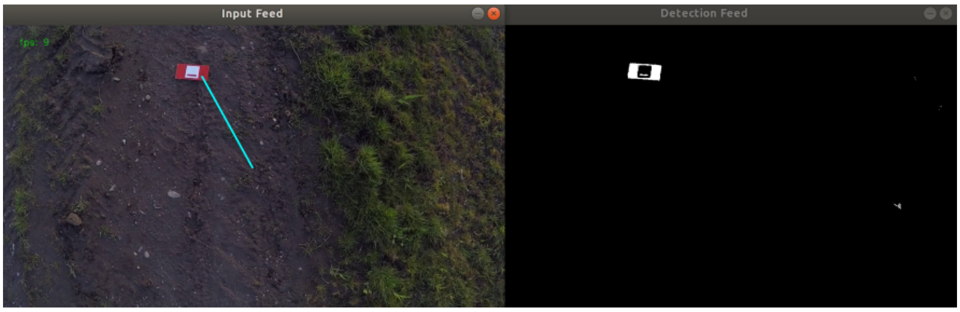


Figure 7.5: Screenshot of recorded flight, depicting the camera feed and the filter output.

7.3 Pose Estimation

7.3.1 Pose estimation with Charuco board

In order to determine which frame the pose estimation is expressed in, a simple experiment is conducted. The pose estimation task is run while a single black and white ArUco marker is initially held in the bottom right corner of the camera view. This yields a positive value for the x- and y-translation. Rotating the marker along its own z-axis does not impact these values. Placing the marker in different areas of the camera view reveals the frame in which pose estimation variables are expressed. The frame has its origin in the center of the image, with positive x-axis to the right and positive y-axis downwards. Hence, the returned translation values from the detection task are independent of marker orientation.

To isolate variables in the pose estimation challenge, the CharUco board is used first in the following experiment. The pose estimation task described in Section 6.5 is run at three unique distances between the CharUco board and the Tello camera. The measured distances are 17 cm, 32 cm and 53 cm in the z-plane, all from the center of the CharUco board. At 17 cm, 16 of the 17 markers are detected. Table 7.2 shows the calculated distance in the z-plane for each detected marker as well as the total average.

Table 7.2: Calculated camera height - CharUco board - 17 cm.

Marker	*Distance (cm)*	*Marker*	*Distance (cm)*
1	16.83	9	17.06
2	16.72	10	17.34
3	16.86	11	17.38
4	17.24	12	17.24
5	16.99	14	17.24
6	16.97	15	17.38
7	17.18	16	17.49
8	17.33	17	17.47
Average			**17.17**

Similar tables for the height of 32cm and 53cm are presented in table 7.3 and table 7.4. All 17 markers are detected at a distance of 32 cm, while 13 markers are detected at 53 cm.

Table 7.3: Calculated camera height - CharUco board - $32\,\mathrm{cm}$.

Marker	*Distance (cm)*	*Marker*	*Distance (cm)*
1	32.81	10	34.06
2	32.43	11	33.99
3	32.55	12	33.53
4	33.21	13	33.98
5	33.09	14	34.45
6	33.51	15	34.46
7	33.49	16	34.51
8	33.36	17	32.64
9	33.60		
Average			**33.52**

Table 7.4: Calculated camera height - CharUco board - $53\,\mathrm{cm}$.

Marker	*Distance (cm)*	*Marker*	*Distance (cm)*
1	58.06	10	60,80
2	58,97	11	61,15
4	59,37	12	59,40
5	59,18	14	62,42
6	57,75	15	33.98
7	57,71	16	62,67
9	59,62	17	60,65
Average			**59.83**

The final table shows an average error of $6.83\,\mathrm{cm}$, equivalent to 12.89%. These results are discussed in Section 8.1. Having established a baseline for the height calculation accuracy, pose estimation of the custom marker with HSV-filtering can commence. An illustration of the experiment setup is depicted in Figure 7.6.

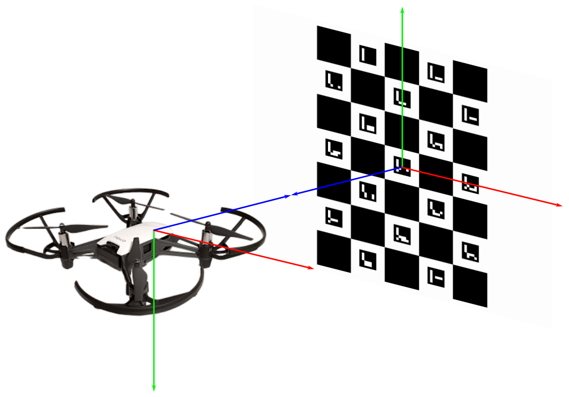


Figure 7.6: Experiment setup with camera frame and marker frame. Positive x-axis marked in red, positive y-axis in green and positive z-axis in blue.

7.3.2 Pose estimation with custom marker

Placing the custom marker $0.5\,\mathrm{m}$ away, directly across from the built-in FaceTime camera, HSV filter values are calibrated manually for the given camera, distance and light conditions at hand. Figure 7.7 shows the experiment setup and table 7.5 shows the final filter values.

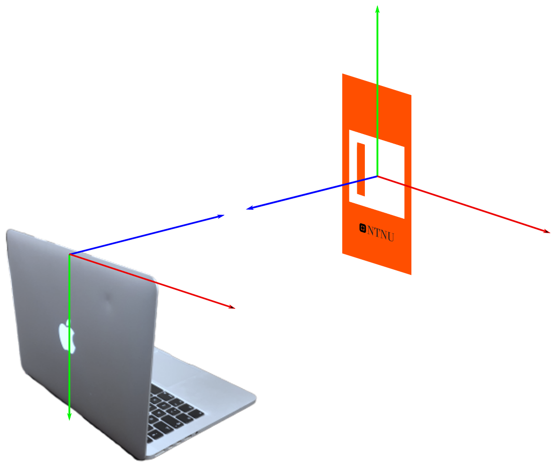


Figure 7.7: Experiment setup with camera frame and marker frame. Positive x-axis marked in red, positive y-axis in green and positive z-axis in blue.

Table 7.5: HSV filter values

	H	*S*	*V*
Low	1	180	60
High	80	255	255

Given the experiment setup shown in Figure 7.7, the expected translation vector and rotation matrix are simply:

$$\begin{bmatrix} 0 & 0 & 0.5 \end{bmatrix}^T \tag{7.4a}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{7.4b}$$

This can be determined by inspection as the difference in orientation is a simple rotation along the x-axis by 180 degrees.

With accurately tuned filter values, the detection and pose estimation task is run. The resulting translation vector, as well as the rotation matrix acquired after using the OpenCV function `Rodriguez()` to transform the rotation vector returned by `estimatePoseSingleMarkers()` are:

$$\begin{bmatrix} 0.0016 & 0.0292 & 0.4978 \end{bmatrix}^T \tag{7.5a}$$

$$\begin{bmatrix} 1 & 0.012 & -0.029 \\ 0.014 & -0.998 & 0.068 \\ -0.028 & -0.068 & -0.996 \end{bmatrix} \tag{7.5b}$$

Small deviations are to be expected as the distances and angles in the experiment setup were measured with a measuring tape and protractor. However, the error in the calculated y-translation could negatively impact navigation accuracy if not corrected or compensated for.

To test the performance of the automatic filter, the experiment described above is repeated in similar light conditions, with the auto-filter. With no movement or change in the camera view, the filter variables stabilize at values shown in Table 7.6

Table 7.6: Automated HSV filter values

	H	S	V
Low	0	67	110
High	36	247	255

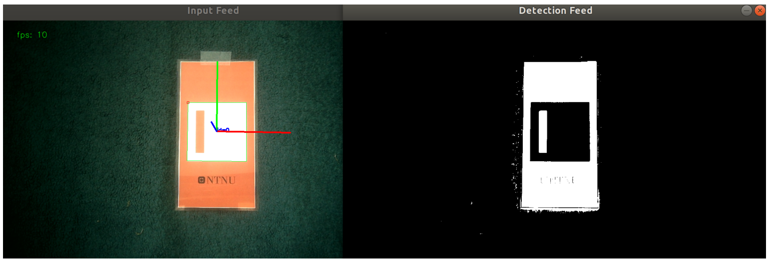
Running the detection and pose estimation task with these filter values gives the following results:

$$\begin{bmatrix} 0.0021 & 0.0177 & 0.4986 \end{bmatrix}^T \tag{7.6a}$$

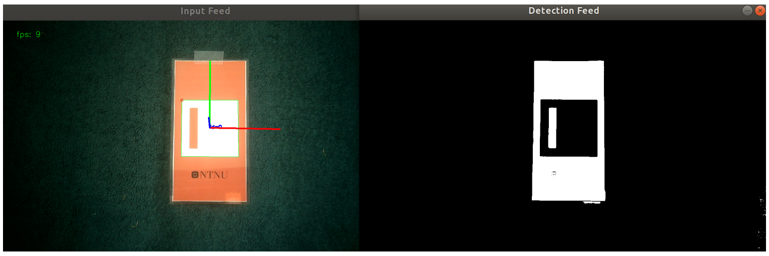
$$\begin{bmatrix} 1 & -0.010 & 0.037 \\ -0.011 & -0.999 & 0.029 \\ 0.036 & -0.029 & -0.998 \end{bmatrix} \tag{7.6b}$$

The results in Equation 7.6 confirm that pose estimation returns equally accurate values whether the automatic HSV filter or the static filter is used. However, there is still an error present in the y-translation regardless of which filter is utilized.

The axes of the marker can be illustrated using the `cv::drawAxis()` function for visual feedback during run-time. Figure 7.8 displays the result of implementing this function.



(a) Using static filter values.



(b) Using automatic filter values.

Figure 7.8: Filter input and output in natural light environment, with drawn axes

7.4 Simulation

7.4.1 Camera-free Simulation with Waves

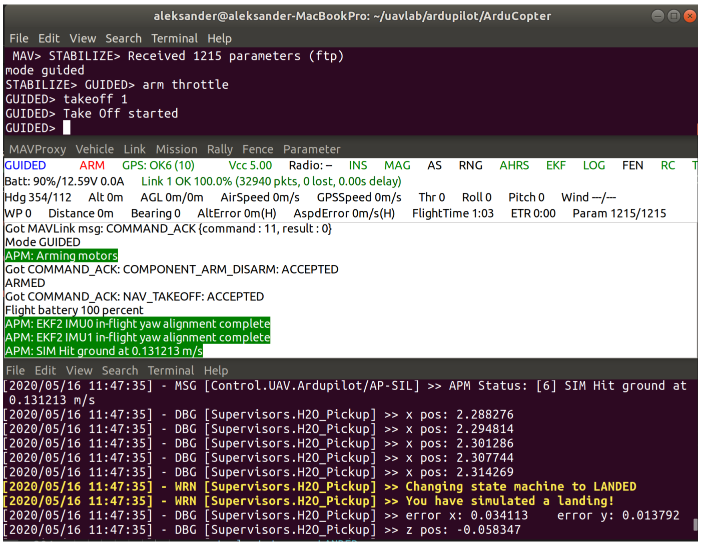
Initially, the camera vision aspect of the system is disregarded, and the guidance system is tested in a simulated environment. A simple `struct` is used to create a simulated MUG. The MUG is parameterized with position and velocity variables in three axes, expressed in the NED frame. The MUG's position and velocity will be affected by the simulated ocean current and waves.

A simulation is run with the state machine described in Section 6.8 and the wave task described in Section 6.6. The wave propagation direction is set to due north, the northward current velocity is $0.06\,\mathrm{m\,s^{-1}}$, the eastward current velocity is $0.03\,\mathrm{m\,s^{-1}}$, and the variables in Stoke's deepwater drift are identical to those described in Section 5.3.

The simulation is initiated by running `sim_vehicle.py --console` in the */ardupilot/ArduCopter* directory. The `--console` option opens a console with detailed information about the drone as well as MavLink messages. After the console displays `APM: EKF2 IMU(0, 1) is using GPS`, a manual takeoff is performed by executing the following commands:

- `mode guided`
- `arm throttle`
- `takeoff x`

where `x` is the desired takeoff altitude. Now the DUNE configuration file can be run by executing `./dune -c "config_file_name" -p AP-SIL` in the build folder of the project. This is when the state machine takes over, and the system runs autonomously. Figure 7.9 shows the terminals and the result of the flight.


aleksander@aleksander-MacBookPro: ~/uavlab/ardupilot/ArduCopter
File Edit View Search Terminal Help
MAVProxy Vehicle Link Mission Rally Fence Parameter
File Edit View Search Terminal Help

```
 MAV> STABILIZE> Received 1215 parameters (ftp)
mode guided
STABILIZE> GUIDED> arm throttle
GUIDED> takeoff 1
GUIDED> Take Off started
GUIDED>
```

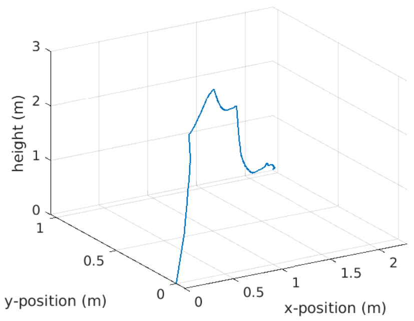
```
GUIDED    ARM    GPS: OK6 (10)    Vcc 5.00    Radio: --    INS    MAG    AS    RNG    AHRS    EKF    LOG    FEN    RC    T
Batt: 90%/12.59V 0.0A    Link 1 OK 100.0% (32940 pkts, 0 lost, 0.00s delay)
Hdg 354/112    Alt 0m    AGL 0m/0m    AirSpeed 0m/s    GPSSpeed 0m/s    Thr 0    Roll 0    Pitch 0    Wind ---/---
WP 0    Distance 0m    Bearing 0    AltError 0m(H)    AspdError 0m/s(H)    FlightTime 1:03    ETR 0:00    Param 1215/1215
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Mode GUIDED
APM: Arming motors
Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
ARMED
Got COMMAND_ACK: NAV_TAKEOFF: ACCEPTED
Flight battery 100 percent
APM: EKF2 IMU0 in-flight yaw alignment complete
APM: EKF2 IMU1 in-flight yaw alignment complete
APM: SIM Hit ground at 0.131213 m/s
```

```
[2020/05/16 11:47:35] - MSG [Control.UAV.Ardupilot/AP-SIL] >> APM Status: [6] SIM Hit ground at
 0.131213 m/s
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> x pos: 2.288276
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> x pos: 2.294814
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> x pos: 2.301286
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> x pos: 2.307744
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> x pos: 2.314269
[2020/05/16 11:47:35] - WRN [Supervisors.H2O_Pickup] >> Changing state machine to LANDED
[2020/05/16 11:47:35] - WRN [Supervisors.H2O_Pickup] >> You have simulated a landing!
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> error x: 0.034113    error y: 0.013792
[2020/05/16 11:47:35] - DBG [Supervisors.H2O_Pickup] >> z pos: -0.058347
```

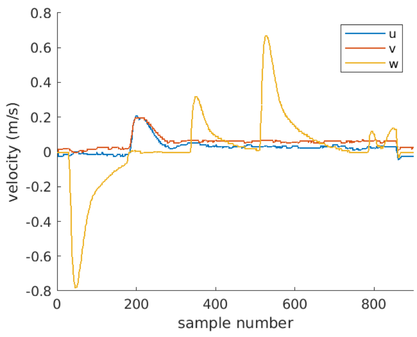
Figure 7.9: Console view of simulation.

As seen in the above figure, a successful landing occurs with a descent velocity of $0.131\,\mathrm{m\,s^{-1}}$ The translation error in the x- and y-direction are $0.034\,\mathrm{m}$ and $0.014\,\mathrm{m}$ respectively.

The flight data from this simulation is logged and processed to create plots of the drone's position and velocity during the mission. These plots are depicted in Figure 7.10a and 7.10b.



(a) Drone position during simulated flight.

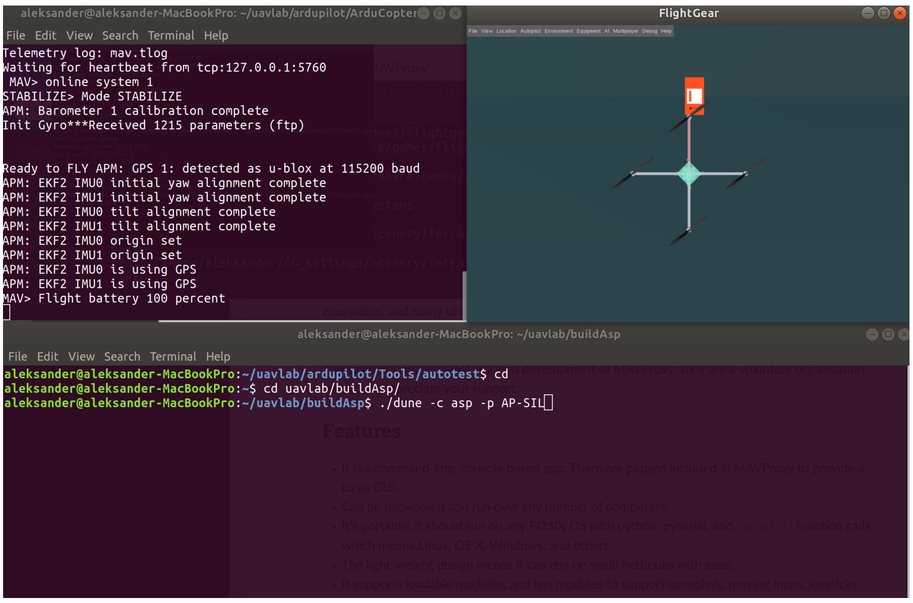


(b) Drone velocity during simulated flight.

Figure 7.10: Graphed position and velocity of simulated drone.

7.4.2 Simulation with FlightGear

To test the performance of the state machine with input from the `Camera/Detection` task, a simulation is run using FlightGear and the virtual custom marker. The initiation is similar to that described in the previous section, however, there are some discrepancies. Firstly, the wave task is not included in the configuration file. Instead, the detection and pose estimation task is included. Furthermore, FlightGear must be launched first, by running `./fg_quad_view.sh` in */ardupilot/Tools/autotest/*. Next, ArduPilot is started by running `sim_vehicle.py -L ZOLICH`, where the `-L` option sets the location to `ZOLICH`. With the simulated quadcopter at this location, the custom marker is located right next to it. Figure 7.11 illustrates this. The final step before running the DUNE configuration file is to perform a manual takeoff as described earlier.


aleksander@aleksander-MacBookPro: ~/uavlab/ardupilot/ArduCopter
File Edit View Search Terminal Help
FlightGear
aleksander@aleksander-MacBookPro: ~/uavlab/buildAsp
File Edit View Search Terminal Help

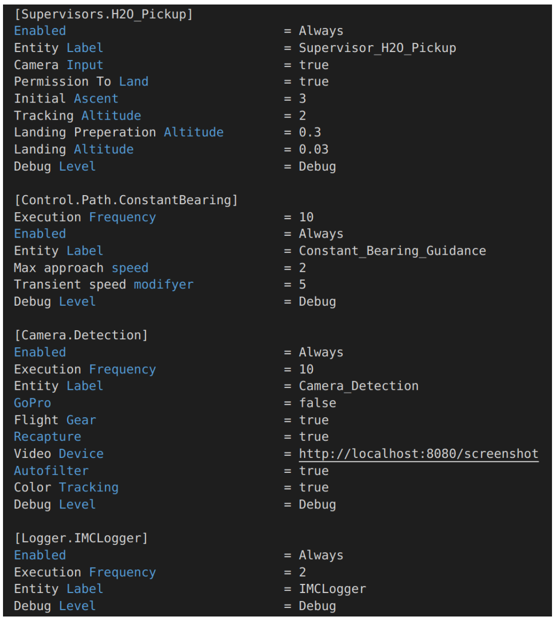
```
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
 MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Barometer 1 calibration complete
Init Gyro***Received 1215 parameters (ftp)

Ready to FLY APM: GPS 1: detected as u-blox at 115200 baud
APM: EKF2 IMU0 initial yaw alignment complete
APM: EKF2 IMU1 initial yaw alignment complete
APM: EKF2 IMU0 tilt alignment complete
APM: EKF2 IMU1 tilt alignment complete
APM: EKF2 IMU0 origin set
APM: EKF2 IMU1 origin set
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
MAV> Flight battery 100 percent
```

```
aleksander@aleksander-MacBookPro:~/uavlab/ardupilot/Tools/autotest$ cd
aleksander@aleksander-MacBookPro:~$ cd uavlab/buildAsp/
aleksander@aleksander-MacBookPro:~/uavlab/buildAsp$ ./dune -c asp -p AP-SIL
```

Figure 7.11: Initial conditions in FlightGear simulation. The view has been moved from below the drone.

The non-trivial section of the configuration file which produces this simulation is shown below in Figure 7.12



```
[Supervisors.H2O_Pickup]
Enabled                                = Always
Entity Label                           = Supervisor_H2O_Pickup
Camera Input                           = true
Permission To Land                     = true
Initial Ascent                         = 3
Tracking Altitude                      = 2
Landing Preperation Altitude           = 0.3
Landing Altitude                       = 0.03
Debug Level                            = Debug

[Control.Path.ConstantBearing]
Execution Frequency                    = 10
Enabled                                = Always
Entity Label                           = Constant_Bearing_Guidance
Max approach speed                     = 2
Transient speed modifyer               = 5
Debug Level                            = Debug

[Camera.Detection]
Enabled                                = Always
Execution Frequency                    = 10
Entity Label                           = Camera_Detection
GoPro                                  = false
Flight Gear                            = true
Recapture                              = true
Video Device                           = http://localhost:8080/screenshot
Autofilter                             = true
Color Tracking                         = true
Debug Level                            = Debug

[Logger.IMCLogger]
Enabled                                = Always
Execution Frequency                    = 2
Entity Label                           = IMCLogger
Debug Level                            = Debug
```

Figure 7.12: Excerpt from configuration file.

Although FlightGear does not use an actual camera, the pose estimation task still requires camera parameters. An attempt is made to calibrate the FlightGear view by placing a virtual CharUco board in the simulation environment. However, acquiring the necessary images, at different angles and distances, is no easy task and would require its own configuration file and tasks in DUNE. Using images from the same angle but varying distance, the calibration task is run. The returned distortion coefficients include a radial distortion value in the three-digit range. This value had a detrimental effect on the calculated z-value in the pose estimation function, causing it to be off by up to $100\,\mathrm{m}$. In addition to this, the horizontal error is not based on the image center. This causes the drone to line up next to the marker as opposed to directly above it. Similar results are obtained after several attempts, leading to the abandonment of this method for obtaining the camera parameters.

Since most of the camera parameters are known, they can simply be filled in, and the remaining unknown variables can be found by trial and error. As no physical camera is in use, the distortion coefficients are all assumed to be zero. Furthermore, the principal point is assumed to be in the image center exactly. The height and width of the image are known, and hence its center can be computed easily. The remaining variable is the focal length, expressed in both pixel width, f_x, and pixel height, f_y. Because the result from the previous calibration of the FlightGear view returned focal lengths of approximately 10000 pixel widths/heights, this number is used as the initial value for f_x and f_y. With these parameters, the issue of incorrect z-translation error is improved, but not resolved. The calculated z-translation is now approximately three times the correct value. Additionally, the horizontal error is now in reference to the image center, causing the desired behaviour where the copter hovers directly above the marker. However, the error in z-translation is still far too large for it to be usable in field testing.

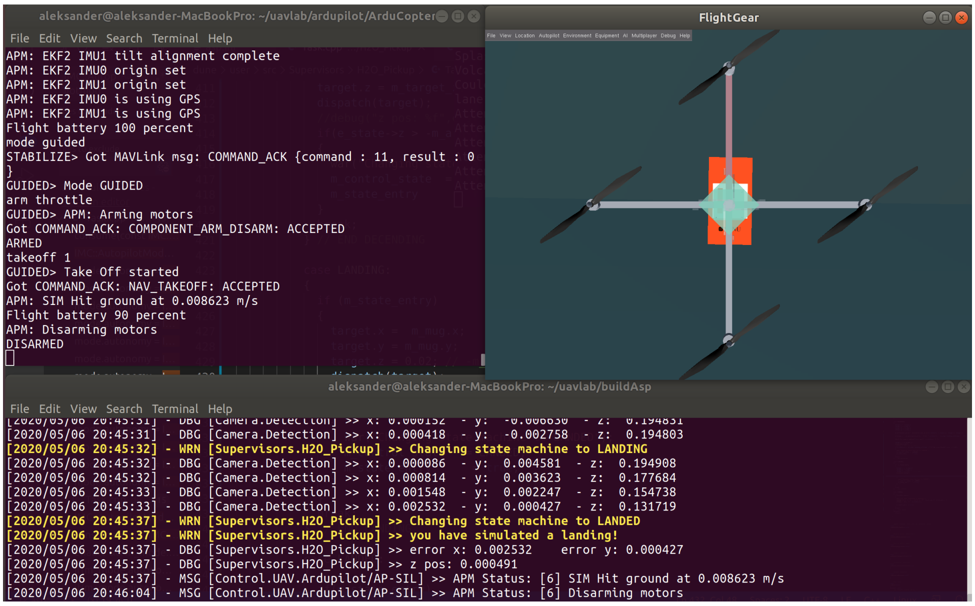
After several rounds of tuning, the following camera parameters were found to yield the best performance:

$$P = \begin{bmatrix} 1000 & 0 & 325 \\ 0 & 1000 & 275 \\ 0 & 0 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.7}$$

With these parameters, the z-error is negligible and complete simulations can proceed.

Using the values presented in Equation 7.7, a manual takeoff is executed and the drone is set to *Guided* mode. After the initial ascent, navigation based on color tracking commences. As the computed translational error based on detected color is a low estimate, the copter moves relatively slowly towards its target. When the horizontal error is low enough, the state machine switches state to TRACK TARGET and descends to the given tracking altitude. During this descent, conditions are met to switch from color tracking to pose estimation. Occasionally, this transition will cause sudden movement in the copter as the target is changed abruptly. Still, these positional changes are on a small scale, and the transition happens quickly. From here, the simulation follows the state machine with expected drone maneuvering.

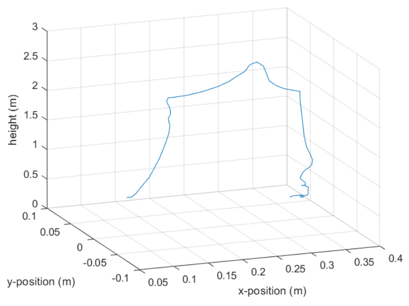
The end state of the simulation is illustrated in Figure 7.13, where the ultimate error in x- and y-translation are shown to be $0.25\,\mathrm{cm}$ and $0.04\,\mathrm{cm}$, respectively. The drone's position and velocity is also logged throughout the simulation and is depicted in Figure 7.14.


aleksander@aleksander-MacBookPro: ~/uavlab/ardupilot/ArduCopter
File Edit View Search Terminal Help
FlightGear
aleksander@aleksander-MacBookPro: ~/uavlab/buildAsp
File Edit View Search Terminal Help

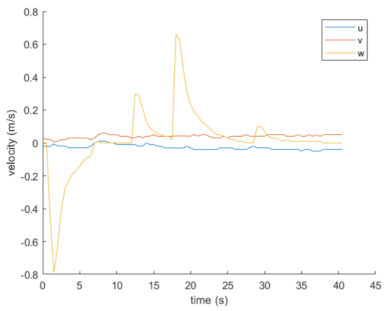
```
APM: EKF2 IMU1 tilt alignment complete
APM: EKF2 IMU0 origin set
APM: EKF2 IMU1 origin set
APM: EKF2 IMU0 is using GPS
APM: EKF2 IMU1 is using GPS
Flight battery 100 percent
mode guided
STABILIZE> Got MAVLink msg: COMMAND_ACK {command : 11, result : 0
}
GUIDED> Mode GUIDED
arm throttle
GUIDED> APM: Arming motors
Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
ARMED
takeoff 1
GUIDED> Take Off started
Got COMMAND_ACK: NAV_TAKEOFF: ACCEPTED
APM: SIM Hit ground at 0.008623 m/s
Flight battery 90 percent
APM: Disarming motors
DISARMED
```

```
[2020/05/06 20:45:31] - DBG [Camera.Detection] >> x: 0.000152  - y:  -0.006630  - z:  0.194831
[2020/05/06 20:45:31] - DBG [Camera.Detection] >> x: 0.000418  - y:  -0.002758  - z:  0.194803
[2020/05/06 20:45:32] - WRN [Supervisors.H2O_Pickup] >> Changing state machine to LANDING
[2020/05/06 20:45:32] - DBG [Camera.Detection] >> x: 0.000086  - y:  0.004581  - z:  0.194908
[2020/05/06 20:45:32] - DBG [Camera.Detection] >> x: 0.000814  - y:  0.003623  - z:  0.177684
[2020/05/06 20:45:33] - DBG [Camera.Detection] >> x: 0.001548  - y:  0.002247  - z:  0.154738
[2020/05/06 20:45:33] - DBG [Camera.Detection] >> x: 0.002532  - y:  0.000427  - z:  0.131719
[2020/05/06 20:45:37] - WRN [Supervisors.H2O_Pickup] >> Changing state machine to LANDED
[2020/05/06 20:45:37] - WRN [Supervisors.H2O_Pickup] >> you have simulated a landing!
[2020/05/06 20:45:37] - DBG [Supervisors.H2O_Pickup] >> error x: 0.002532    error y: 0.000427
[2020/05/06 20:45:37] - DBG [Supervisors.H2O_Pickup] >> z pos: 0.000491
[2020/05/06 20:45:37] - MSG [Control.UAV.Ardupilot/AP-SIL] >> APM Status: [6] SIM Hit ground at 0.008623 m/s
[2020/05/06 20:46:04] - MSG [Control.UAV.Ardupilot/AP-SIL] >> APM Status: [6] Disarming motors
```

Figure 7.13: End of FlightGear simulation. The view has been moved from below the drone.



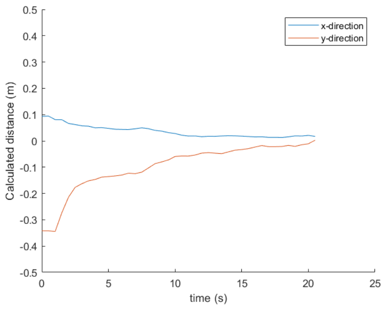
(a) Drone position during simulated flight.



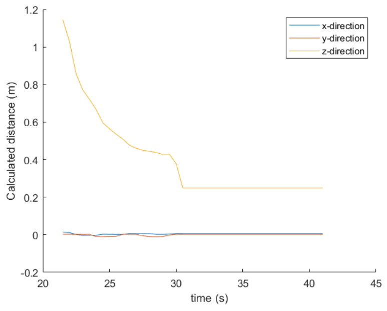
(b) Drone velocity during simulated flight.

Figure 7.14: Graphed position and velocity of simulated drone.

Figure 7.15 depicts the calculated distance to the virtual ArUco marker based on both the color detection and pose estimation method. Figure 7.16 shows the comparison of DUNE's built-in height estimate and the height estimate acquired from pose estimation of the ArUco marker. As seen in Figure 7.16, there is a discrepancy between the two estimates. This error decreases over time until the drone is hovering so low that the ArUco marker is lost from the camera view, and new pose estimates are no longer available.



(a) Using color detection.



(b) Using pose estimation.

Figure 7.15: Calculated distance to the virtual ArUco marker based on color detection and pose estimation during simulated flight.

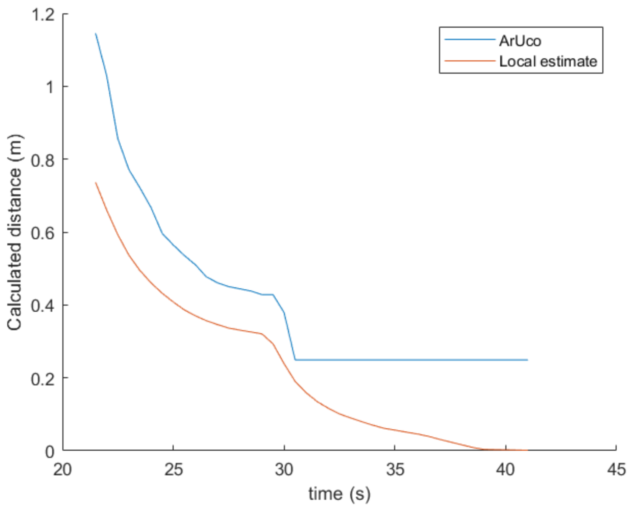


Figure 7.16: Comparison of DUNE's height estimate and the height estimate based on ArUco detection during the simulation.

7.5 Field Tests

Field tests are performed on land using the 3DR Solo drone, its associated remote controller and a GoPro Hero 4 camera. An overview of the main software and hardware components, as well as their communication flow, is depicted in Figure 7.17. External computing is utilized in these experiments, and the added delay produced by this is less than 0.25 seconds, which is sufficiently low for the intended operations. The remote controller is also a WiFi access point, which both the drone and the external computer are connected to. Sending `nc 10.1.1.1 5502` from the terminal to the remote controller over user datagram protocol (UDP) activates the video stream. The feed from the camera is sent to the controller via real-time protocol (RTP). This data feed is, in turn, sent to the computer's localhost.

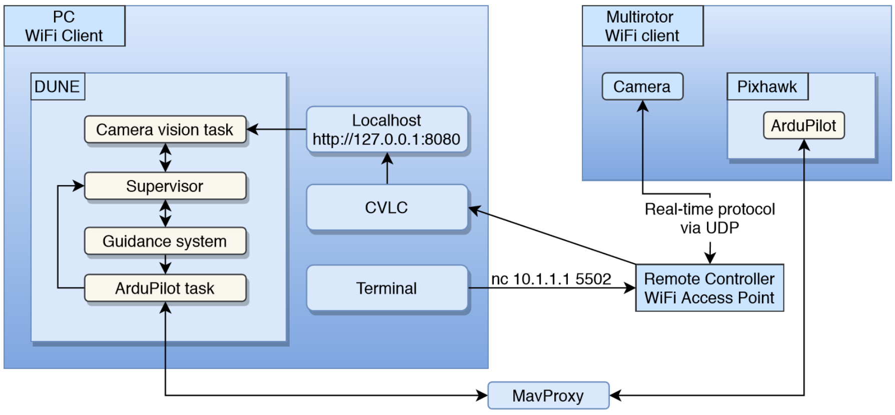
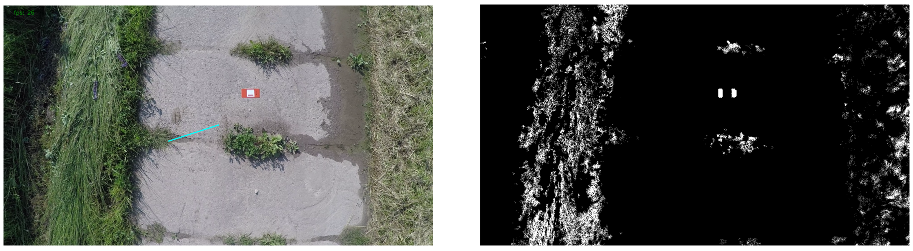


Figure 7.17: Software and hardware overview of field test setup

Initial testing revealed that the automatic HSV filter did not yield sufficient filtering. Several colors other than the aerospace orange pass through the filter, causing the color detection process to return incorrect navigation targets. The custom marker itself is also not filtered optimally, rendering ArUco detection infeasible. Figure 7.18 shows both the camera feed and the filter output of a given moment during an initial flight test. As seen in Figure 7.18a, the light blue line is pointing away from the marker, signifying that the target location sent to the drone's guidance system is in the wrong direction.



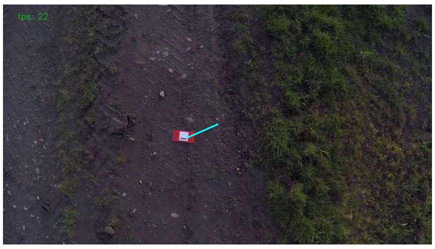
(a) Camera feed with drawn line to navigation target. **(b)** Filter output.

Figure 7.18: Filtered and unfiltered image frame from GoPro Hero 4 during flight with 3DR Solo.

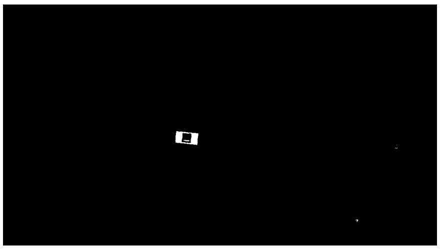
Using recorded video from the GoPro Hero 4 during flights with different light conditions, a static HSV filter is created and optimized specifically for these conditions. The filter range was found by trial and error, and the final filter values are presented in Table 7.7. The results of utilizing this filter are shown in Figure 7.19 and Figure 7.20. The results are promising, as the lighting conditions are significantly dissimilar in these flights, yet the filter process is successful in both.

Table 7.7: HSV filter values for GoPro Hero 4

	H	S	V
Low	100	80	100
High	170	255	255

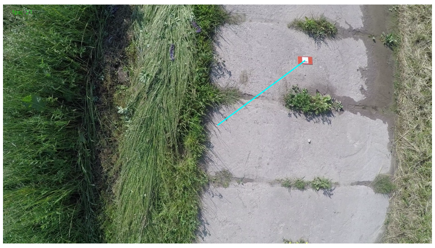


(a) Camera feed with drawn line to navigation target.

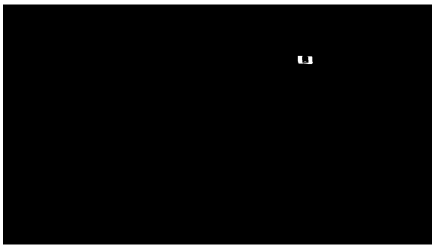


(b) Filter output.

Figure 7.19: Filtered and unfiltered image frame from GoPro Hero 4 during flight with 3DR Solo. The static HSV filter from 7.7 is used.



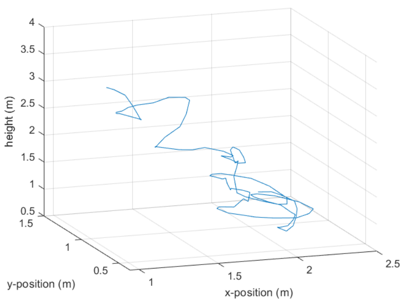
(a) Camera feed with drawn line to navigation target.



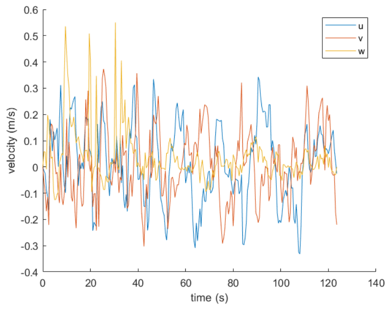
(b) Filter output.

Figure 7.20: Filtered and unfiltered image frame from GoPro Hero 4 during flight with 3DR Solo in bright daylight. The static HSV filter from 7.7 is used.

With the filter values in Table 7.7 further testing can commence. Field tests are initiated by a manual takeoff and maneuvering of the drone such that the custom marker is in the camera's field of view. Next, the camera feed is activated, and the drone's mode is switched to *Guided*, triggering the software to take over control. The static HSV filter works as intended, filtering out everything except the aerospace orange color. This allows for an accurate navigation target. However, a major issue is discovered during the final day of field tests. In addition to the (more or less) constant delay originating from sending the video feed, a delay build-up occurs. This built-up delay prevents the control system from acting in real-time and causes the motion of the drone to oscillate. The oscillation amplitude increasing as time passes, until it reaches a plateau. This oscillation prevents landing conditions from being fulfilled, and thus the lowest height the drone reaches is $17.34\,\mathrm{cm}$ according to DUNE's state estimate, and $51.13\,\mathrm{cm}$ according to ArUco pose estimation. The oscillation can be seen in both Figure 7.21 and Figure 7.22b.

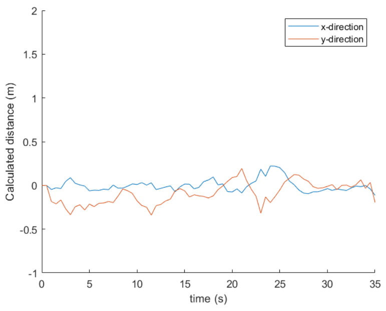


(a) Drone position during flight.

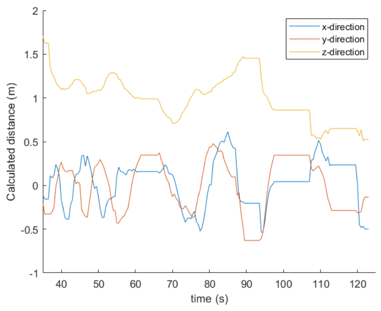


(b) Drone velocity during flight.

Figure 7.21: Graphed position and velocity of the 3DR Solo during flight.



(a) Using color detection.



(b) Using pose estimation.

Figure 7.22: Calculated distance to the virtual ArUco marker based on color detection and pose estimation.

Although it seems as if Figure 7.22 illustrates that the color detection method is more accurate than the pose estimation method, this is not the case. As described in Section 6.4, the maximum calculated error in each horizontal direction is $0.5\,\mathrm{m}$ due to the nature of the implemented method. Furthermore, the delay build-up has a larger impact on the pose estimation method as this method begins after color tracking has finished.

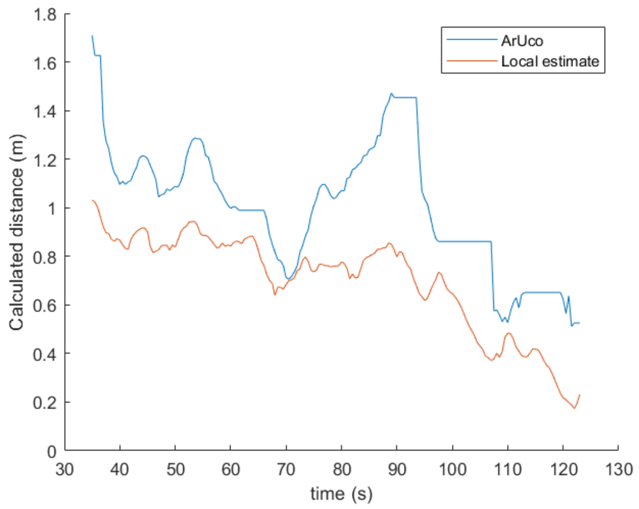


Figure 7.23: Comparison of DUNE's height estimate and the height estimate based on ArUco detection.

Figure 7.23 shows a clear discrepancy between DUNE's built-in position estimate and the height estimate based on ArUco detection. The height trends are somewhat overlapping, but overall the similarity of the two plots is less than ideal.

Chapter 8

Discussion and Conclusion

8.1 Discussion

Several elements involved in a real flight mission have been simplified or disregarded in simulations. Among these is the wave-element. Ocean waves were modeled as noisy sine waves, which is a significant simplification of natural waves in all sea states. Furthermore, Stoke's drift velocity for deepwater waves was used to model wave-induced velocity, with the added modifier C. This is also a crude estimate of actual wave-induced velocity. Additionally, the simulated MUG's z-position was set to follow the z-position of the wave at its location exactly, whereas there would be discrepancies between the two in real life.

When calibrating a camera with OpenCV and a CharUco board, it is necessary to use several frames from varying angles and distances to achieve a low re-projection error. As previously mentioned, OpenCV states that the closer the re-projection error is to zero, the better the calibration process was. However, it is still difficult to determine what a sufficient or good re-projection error value is. One can visually inspect the undistorted image as a rough evaluation, but a more precise evaluation method is currently absent.

The automatic HSV filter provides accurate and desired filtering in simulations, in tests with the HD Facetime camera and in some tests with the GoPro Hero 4. The automatic filter adapts better to varying light conditions than most static filters, and it removes more image noise on average. However, the automatic filter was not able to perform the desired filtering consistently across varying light conditions with the GoPro camera. Perhaps more accurate tuning is all that is required, but at this time the automatic filter is deemed insufficient for field operations.

A major issue with both filters is the poor handling of light reflection. Even small reflections can interfere with the filter, and since OpenCV's detection algorithm has rigorous demands, a slight interference with the ID code of the marker can cause detection to fail. This raises questions to the robustness and viability of using laminated markers as the primary basis of navigation. Additionally, the detection process is yet to be conducted at sea, and it is not clear how water and water droplets on top of the marker will affect light reflection, but this could have a large impact on detection and pose estimation. A potential improvement on the current marker is to use tear- and waterproof paper, instead of using lamination, as this would reduce the glare and intensity of light reflection, and still maintain the markers structural integrity at sea. Attaching the waterproof marker to the MUG such that it can not twist or bend would be an essential task using this alternative.

Results in Section 7.3.1 show that the calculated z-translation becomes increasingly inaccurate with increasing distance. At a distance of $0.53\,\mathrm{m}$, the average error is 12.89%. The ArUco codes in the CharUco board have a side length of $1.2\,\mathrm{cm}$. The ArUco code within the custom marker has a side length of $8\,\mathrm{cm}$. Extrapolating the error from the CharUco test would yield an error in the calculated z-translation of $0.45\,\mathrm{m}$ at $3.53\,\mathrm{m}$ with the custom marker. Having a 12-13$\%$ error at this height is not critical for intended operations in the scope of this thesis, but it is important to be aware of for other potential uses. Section 7.3.1 also illustrates that as the distance between the drone and the marker decreases, so does the error in the calculated z-translation. The field tests do not verify this relationship between variables, but the data is limited.

The color detection and tracking scheme implemented in this thesis is a simple, yet effective method. Still, its performance is dependant on the performance of the HSV filtering process. Moreover, objects of similar color to aerospace orange, noise in the filtered image and incorrectly filtered objects can all negatively affect the output of the color detection and tracking method. However, as described in Section 6.4, the drone will move towards the centroid of the combined area of interest (AOI). This means that so long as the combined non-marker AOI is smaller than the marker AOI, the marker will remain relatively close to the center of the camera view. Non-marker AOI of greater size than the marker AOI can move the marker further away from the center, and as the drone descends the marker may be lost from the camera view before the switch from color tracking to ArUco tracking is triggered. This could be especially problematic if an orange buoy, a common object on the sea surface, is in the camera view.

Simulation results with wave-induced velocity, but without the camera system, indicate that the drone has little trouble landing on a slowly moving target, missing the center of the marker by $0.0368\,\mathrm{m}$. Still, the constant bearing guidance system received perfectly accurate marker positions during this simulation, there was no wind present, and an error of $0.0368\,\mathrm{m}$ on a $0.10\,\mathrm{x}0.20\,\mathrm{m}$ surface could be considered insufficiently accurate. The simulation was run with a C value of 0.1, considerably reducing the wave-induced velocity compared to the original Stoke's drift model. If sea conditions cause higher wave-induced velocities, or if a strong ocean current is present, the accuracy in the landing phase will be of great concern.

Simulations implementing the camera vision system and a stationary virtual marker yield highly accurate landings. However, the simulation conditions are not realistic; there is no wind present, there are no other objects than the marker, there is no light that can cause glare, and the background is ideal.

Field tests conclude that the automatic HSV filter does not perform consistently enough, although it does succeed in several conditions. However, the final static HSV filter range is found to be surprisingly versatile with respect to varying conditions. In terms of navigation there are issues with the accuracy in the pose estimation method and the camera feed delay. The cause of the delay build-up is yet to be determined, but it is likely an issue with the newly added DUNE tasks, and should be avoidable. In field tests, the pose estimation is only compared to DUNE's built-in estimate, which itself is not a perfect estimate, but is accurate over short time periods. In Section 7.3.2, highly accurate estimates are achieved using the FaceTime camera and the ArUco marker at a distance of $0.5\,\mathrm{m}$. Potential explanations for the inaccuracies in field tests is an inaccurate camera calibration, inaccurate filtering, or image noise that can cause the marker to appear slightly distorted.

8.2 Conclusion

A camera calibration method utilizing OpenCV functionality has been presented, and its performance has been verified at a superficial level. An automatic HSV filter intended to detect and filter through aerospace orange has been developed. However, inconsistent performance rendered it unsuitable for field operations. Instead, a static HSV filter is optimized based on specific data. This filter acts as the basis in the implemented color detection scheme, which also incorporates the OpenCV library. An ArUco marker detection and pose estimation method is presented and realized using HSV filtering and OpenCV. All of the above-mentioned processes have been integrated into DUNE.

A state machine governing all stages of the autonomous flight, excluding takeoff, has been formulated and implemented in DUNE. A pre-existing constant bearing guidance system has been integrated and modified to allow both relative- and absolute-position targets.

FlightGear was used as a 3D simulation environment, incorporating both custom terrain to emulate the color the ocean surface, and the custom marker. Simulation results produced accurate landings with calculated errors as low as $0.25\,\mathrm{cm}$ and $0.04\,\mathrm{cm}$ in x- and y-translation, respectively. However, as previously stated, the conditions in FlightGear are not realistic.

Field tests were performed on land using a 3DR Solo drone and the custom NTNU marker. Navigation based on color detection and pose estimation of the ArUco marker is executed. However, a delay build-up in the video feed prevents real-time control and causes oscillations in the 3DR Solo's movement. Due to this, landing conditions were never met, and the lowest height the 3DR reached was $17.34\,\mathrm{cm}$ according to DUNE's state estimate, and $51.13\,\mathrm{cm}$ according to ArUco pose estimation.

8.3 Further Work

As the end goal of this project is to pick up a MUG from the ocean surface autonomously, several challenges still remain. An important step to achieve the end goal, is to add functionality in DUNE that allows for autonomous take-off and mode switching.

To improve the relevance of simulation results, a wind model, potentially the one described in 5.4, and a wave model should be integrated with the camera vision based simulation. The relative orientation between the drone and the MUG as a result of ocean waves should be taken into account during the landing phase.

It should be considered whether the laminated marker ought to be replaced with waterproof paper or another solution that produces less light reflection.

Implementing a Kalman filter as described in Section 3.5 to model the motion of the MUG can potentially improve landing accuracy as the camera loses sight of the complete ArUco marker in the final descent. A Kalman filter is also useful if light reflecting off the marker causes intermediate disruptions in the detection and pose estimation process.

As of now, there are no further instructions given to the drone after it has successfully landed on the marker and disarmed. An electropermanent magnet should be acquired and mounted onto the multi-rotor, and necessary code to pick up the MUG and return it to launch using this magnet should be implemented.

Bibliography

[1] International orange.
https://encycolorpedia.com/ff4f00, 2013.
Accessed: Feb 2020.

[2] Neptus - home.
https://github.com/LSTS/neptus/wiki, 2013.
Accessed: Nov 2019.

[3] Distortion.
https://www.edmundoptics.com/knowledge-center/application-notes/imaging/distortion/, 2014.
Accessed: Feb 2020.

[4] Dune - home.
https://github.com/LSTS/dune/wiki, 2014.
Accessed: Oct 2019.

[5] Dune - unified navigation environment.
https://lsts.fe.up.pt/toolchain/dune, 2015.
Accessed: Oct 2019.

[6] Neptus - command and control software.
https://lsts.fe.up.pt/toolchain/neptus, 2015.
Accessed: Nov 2019.

[7] 3dr solo.
https://www.tek.no/test/i/zGQG3q/3dr-solo, 2016.
Accessed: May 2020.

[8] Ardupilot - software.
https://ardupilot.org/index.php/about, 2016.
Accessed: Oct 2019.

[9] Ocean forecast.
https://www.yr.no/place/Ocean/63.44180_10.36698/, 2016.
Accessed: Feb 2020.

[10] S1000 - features.
https://www.dji.com/no/spreading-wings-s1000/feature, 2016.
Accessed: Nov 2019.

[11] Opencv - aruco marker detection.
https://docs.opencv.org/master/d9/d6a/group__aruco.html#ga84dd2e88f3e8c3255eb78e0f79571bd1, 2017.
Accessed: Mar 2020.

[12] Opencv - calibration with aruco and charuco.
https://docs.opencv.org/master/da/d13/tutorial_aruco_calibration.html, 2017.
Accessed: Sep 2019.

[13] Opencv - camera calibration and 3d reconstruction.
https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, 2017.
Accessed: Sep 2019.

[14] Opencv - detection of aruco markers.
https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html, 2017.
Accessed: Aug 2019.

[15] Opencv - installation in linux.
https://docs.opencv.org/3.3.0/d7/d9f/tutorial_linux_install.html, 2017.
Accessed: Aug 2019.

[16] Withrobot - project history.
http://withrobot.com/en/projecthistory/, 2017.
Accessed: Oct 2019.

[17] Oasys - home.
https://blogg.hioa.no/oasys/, 2018.
Accessed: Oct 2019.

[18] Tello - camera.
https://www.cnet.com/products/ryze-tello/, 2018.
Accessed: Nov 2019.

[19] Tello - specifications.
https://www.ryzerobotics.com/tello/specs, 2018.
Accessed: Nov 2019.

[20] Randal W. Beard and Timothy W. McLain. *Small Unmanned Aircraft, Theory and Practice*. Princeton University Press, 2012.

[21] Claudia Cenedese and Arnold L. Gordon. Ocean current. `https://www.britannica.com/science/ocean-current`, 2018. Accessed: Feb 2020.

[22] Peter Corke. *Robotics, Vision and Control*. Springer, 2011.

[23] Yi Feng, Cong Zhang, Stanley Baek, Samir Rawashdeh, and Alireza Mohammadi. Autonomous landing of a uav on a moving platform using model predictive control. *MDPI*, 2018.

[24] Douglas Fields. Galilean relativity. University of New Mexico, 2015.

[25] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011.

[26] Jens Ludvik Grytnes Joberg. Multirotor pickup of object in the sea. Master's thesis, Norwegian University of Science and Technology, 2019.

[27] Ara N. Knaian. *Electropermanent magnetic connectors and actuators : devices and their application in programmable matter*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2010.

[28] Vuk Krivokapic. Automatic landing of multi-rotor on moving platform. Master's thesis, Norwegian University of Science and Technology, 2019.

[29] Yunting Li, Jun Zhang, Wenwen Hu, and Jinwen Tian. Laboratory calibration of star sensor with installation error using a nonlinear distortion model. *Applied Physics B: Lasers and Optics*, 115, 2013.

[30] Kevin Ling, Derek Chow, Arun Das, , and Steven L. Waslander. Autonomous maritime landings for low-cost vtol aerial vehicles. In *2014 Canadian Conference on Computer and Robot Vision*, 2014.

[31] Mayank Mittal, Rohit Mohan, Wolfram Burgard, and Abhinav Valada. Vision-based autonomous UAV navigation and landing for urban search and rescue. *CoRR*, 2019.

[32] Willard J. Pierson and Lionel Moskowitz. A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii. *Journal of Geophysical Research*, 1964.

[33] Riccardo Polvara, Sanjay Sharma, Jian Wan, Andrew Manning, and Robert Sutton. Vision-based autonomous landing of a quadrotor on the perturbed deck of an unmanned surface vehicle. *MDPI*, 2018.

[34] Marwan Shaker, Mark N.R. Smith, Shigang Yue, and Tom Duckett. Vision-based landing of a simulated unmanned aerial vehicle with fast reinforcement learning. In *2010 International Conference on Emerging Security Technologies*, 2010.