Master's thesis

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

Odin Linga

Automatic landing of multi-rotor on a floating maritime platform

Master's thesis in Cybernetics and Robotics

Supervisor: Tor Arne Johansen

June 2020

NTNU
Norwegian University of Science and Technology

Odin Linga

Automatic landing of multi-rotor on a floating maritime platform

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

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

PROJECT DESCRIPTION SHEET

Name:	Odin Linga
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Automatisk landing av multirotor på flytende maritim plattform
Thesis title (English):	Automatic landing of multi-rotor on a floating maritime platform

Thesis Description: Automatic landing of unmanned aircraft, including multi-rotors and fixed-wing UAVs, is a necessary feature to make their use in the maritime industry widespread. The OASYS project proposes the development of a swarm of low-cost Micro Underwater Gliders (MUGs) that can be deployed/recovered using UAVs. The recovery of these gliders requires development of methods for automatic landing of a multi-rotor UAV on ships or small floating platforms. The purpose of the thesis is to implement and perform field testing of a system for automatic landing of a multi-rotor UAV on a floating platform. Moving-base Real-Time-Kinematic (RTK) Global Navigation Satellite System (GNSS) positioning with the base receiver placed on the platform is to be used for navigation. The software should be implemented using the LSTS toolchain, controlling a UAV using the ArduPilot flight software.

The following tasks should be considered:

1. Perform a literature review on automatic landing systems of multi-rotor UAVs on moving platforms, and the different navigation methods that can be used, including RTK GNSS.
2. Implement the necessary algorithms for automatic landing in DUNE, using an interface to an autopilot running ArduPilot, and a platform equipped with GNSS receiver.
3. Simulate a realistic moving platform in DUNE, and test the implemented software using software-in-the-loop simulation of ArduPilot.
4. Test the software in the field using a multi-rotor, landing on a platform.
5. Discuss your results.
6. Conclude and suggest future work.

Due date:	2020-06-22
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Tor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-supervisor:	PhD Cand. Martin L. Sollie, Dept. of Eng. Cybernetics, NTNU

Preface

This thesis concludes my master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The work on the thesis was carried out in the spring of 2020, and awards 30 credits, which is equivalent to one full semester. The thesis is a continuation of the specialization project that I wrote in the fall of 2019.

In the beginning of the semester most of the work in this project was carried out in the UAV lab at NTNU, but due to the outbreak of Covid-19, the rest had to be carried out at home, with the exception of some preparation for the flight test, and the test itself, which both had to be postponed by roughly a month. This also made it impossible to prepare a moving platform which was the plan, so the test had to be performed on a static target on the ground.

I would like to thank my supervisor Tor A. Johansen for giving me the opportunity to work on this project, and giving it direction. I also thank him for handling applications made necessary due to Covid-19. I would also like to thank my co-supervisor Martin L. Sollie for helping me with the project, and especially for preparing the hardware for the test. Lastly I would like to thank Pål Kvaløy helping during the flight test.

Odin Linga
Trondheim, June 2020

Abstract

The OASYS project aims to use Miniature Underwater Gliders (MUGs) to monitor the environment under the ocean, and then use multi-rotor UAVs to pickup the MUGs and bring them to autonomous surface vehicles where they can recharge. The goal of this project is to develop the algorithms necessary to automatically land these UAVs on a floating maritime platform, using the LSTS toolchain and ArduPilot.

Software is implemented in DUNE to execute the landing. A simple state machine, where the UAV progresses through the states after certain conditions are met, but with a possibility of entering manual control at any time, is created. The state machine will first take off, and reach a specified height before completing a circle motion. Then the UAV will use Constant Bearing Guidance to approach the platform, and then land with a constant vertical speed, until it is stopped by the platform.

A simulation of a landing platform is implemented using wave mechanics and Euler's method. The UAV and the platform is simulated using ArduPilot Software-in-the-loop simulation and DUNE. The simulation is successful, but ArduPilot does not simulate physical contact between the platform and the drone, so it's difficult to fully test landing on a moving platform.

A flight test is conducted at Udduvoll airfield. The flight test is done with a static target on the ground instead of the originally intended maritime platform, because of time constraints. The testing allowed for fixing a few issues with the initial software, like the drone tipping over on landing, and ultimately, successful landings where performed.

It is concluded that the project has been a success, and that the most immediate way forward should be to test the project with a moving target. It is also suggested to test with a smaller platform radius, to see how small of a platform it's possible to land on. Finally it is suggested to look into the possibility of simulating physical contact between the UAV and the platform.

Sammendrag

OASYS prosjektet forsøker å bruke Miniatyr Undervanns Glidere (MUGs) for å overvåke miljøet under havet, og bruke multirotor UAV-er til å plukke opp MUG-ene og frakte dem til autonome overflatefartøy hvor de kan lade. Målet med dette prosjekter er å utvikle nødvendige algoritmer for å automatisk lande disse UAV-ene på en flytende maritim platform, ved å bruke LSTS toolchainen og ArduPilot.

Programvare er implementert i DUNE for å utføre landingen. En enkel tilstandsmaskin, hvor dronen går gjennom tilstandene etter at visse vilkår er oppfylt, med en mulighet for å bytte til manuell kontroll, blir laget. Tilstandsmaskinen vil først ta av, og nå en gitt høyde, før den gjennomfører en sirkelbevegelse. UAV-en vil så nærme seg plattformen ved å bruke Constant Bearing Guidance, og så lande med konstant vertikal hastighet til den blir stoppet av platformen.

En simulering av en landingsplatform blir implementert ved å bruke bølgemekanikk og Eulers metode. UAV-en og platformen blir simulert ved å bruke ArduPilot Software-in-the-loop simulering og DUNE. Simuleringen er vellykket, men ArduPilot simulerer ikke fysisk kontakt mellom dronen og platformen, så det er vanskelig å teste landing på bevegende platform fult ut.

En flygningstest blir gjennomført på Udduvoll. Testen blir gjennomført med et statisk mål på bakken isteden for en flytende maritim platform som originalt tiltenkt på grunn av tidsbegrensninger. Testingen gjorde at noen problemer med den opprinnelige koden ble oppdaget og fikset, som om at dronen tippet over under landing, og til slutt ble velykkete landinger gjennomført.

Det blir konkludert med at prosjektet har vært vellykket, og at den første tingen som bør gjøres videre er å teste prosjektet med et bevegende mål. Det blir også foreslått å teste med mindre paltform radius, for å se hvor liten platform det er mulig å lande på. Til slutt blir det foreslått å se på muligheten for å simulere fysisk kontakt mellom UAV-en og platformen.

Contents

Preface **I**

Abstract **II**

Sammendrag **III**

Abbreviations **X**

1 Introduction **1**

1.1 Motivation 1

1.2 Specialization Project 1

1.3 Literature Review 2

1.4 Outline 3

2 Basic Theory **4**

2.1 Software 4

2.1.1 DUNE 4

2.1.2 Neptus 4

2.1.3 IMC 5

2.1.4 Glued 5

2.1.5 ArduPilot 5

2.2 Hardware 6

2.2.1 UAV 6

2.2.2 BeagleBone . . . 6
2.2.3 Landing Platform Instrumentation . . . 6
2.2.4 PixHawk . . . 8
2.2.5 GNSS Receivers . . . 8
2.3 Theoretical Concepts . . . 9
2.3.1 Constant Bearing . . . 9
2.3.2 Waves . . . 10
2.3.3 Euler's Method . . . 12
2.3.4 WGS84 . . . 12
2.3.5 Real Time Kinematic . . . 13

3 Implementation **14**
3.1 ArduPilot Communications Task . . . 15
3.2 Landing State Machine . . . 15
3.3 Platform State Machine . . . 18
3.4 Constant Bearing Controller . . . 18
3.5 Target Generator . . . 19
3.6 Landing Platform Simulation . . . 19
3.7 Configuration . . . 20

4 Simulation **23**
4.1 Simulation execution . . . 23
4.2 Simulation Results . . . 26

5 Flight Test **29**
5.1 Initial Testing . . . 30
5.2 Successful Tests . . . 31

6 Conclusion **34**
6.1 Further work . . . 34

Bibliography **35**

List of Figures

2.1 DJI S1000+ 6

2.2 Net-02 landing station 7

2.3 Illustration of Constant Bearing guidance 9

2.4 WGS84 reference frame 13

2.5 Illustration of RTK 13

3.1 Simplified illustration of the state machine 17

4.1 Running DUNE 23

4.2 Running ArduPilot 23

4.3 Screenshot of the "landing platform" drifting with the current 24

4.4 Screenshot of the UAV flying in a circle 24

4.5 Output of the DUNE console during simulation 25

4.6 3D plot of the path of the drone and landing platform in the ENU frame . . 26

4.7 Position error between drone and landing platform 27

4.8 ENU path of unstable simulation 28

4.9 Position error for horizontally static net-02 simulation 28

4.10 Position error for horizontally moving net-02 simulation 28

5.1 Picture of the van and the drone. Photo: Martin L. Sollie 29

5.2 Illustration of NTNU-NET-02 laid out on the ground, viewed from above . 30

5.3 Plot of Desired Velocity at the end of a test 30

5.4 Spike in target velocities. 30

5.5 Velocity and desired velocity from a successful landing 31

5.6 UAV path from a successful landing, in ENU 32

5.7 Position errors for the successful landing 32

5.8 Picture of the drone landed between the antennas. Photo: Martin L. Sollie 33

List of Tables

2.1 Sea states and corresponding K_ω 11

Abbreviations

AP-SIL ArduPilot Software-in-the-loop.

CB Constant Bearing.

CLI Command Line Interface.

CPU Central Processing Unit.

DOF Degrees of Freedom.

ECEF Earth Centered Earth Fixed.

EKF Extended Kalman Filter.

ENU East-North-Up.

GCS Ground Control Station.

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

IERS International Earth Rotation Service.

IMC Inter-Module Communication.

IMU Inertial Measurement Unit.

INS Inertial Navigation System.

LOS Line Of Sight.

LSTS Laboratório de Sistemas e Tecnologia Subaquática (Underwater Systems and Technology Laboratory).

MAV Miniature Aerial Vehicle.

MRA Mission Review and Analysis.

MSL Mean Sea Level.

MUGs Miniature Underwater Gliders.

NED North-East-Down.

NTNU Norges teknisk-naturvitenskapelige universitet.

RTCM Radio Technical Commission for Maritime Services.

RTK Real-Time-Kinematic.

SITL Software-in-the-loop.

UART Universal Asynchronous Receiver-Transmitter.

UAV Unmanned Aerial Vehicle.

UDP User Datagram Protocol.

WGS84 World Geodetic System 1984.

Chapter 1

Introduction

1.1 Motivation

Automatic landing of unmanned aircraft, including multi-rotors and fixed-wing UAVs, is a necessary feature to make their use in the maritime industry widespread. The purpose of this project is to implement the software architecture and control algorithms needed for automatic landing of a multi-rotor UAV equipped with an autopilot running the ArduPilot software. The software is implemented using the LSTS toolchain, using moving-base Real-Time-Kinematic (RTK) Global Navigation Satellite System (GNSS) positioning for navigation. The advantage of automatic landing is that it removes the need to have a trained pilot on board in order to operate the drone. It also allows for UAVs to operate from unmanned platforms. Another advantage is that it can be really difficult to manually land a drone if the ship is moving a lot due to waves.

The OASYS project is a collaboration between NTNU, OsloMet, the Norwegian Polar Institute and others, aiming to develop a fully automated Ocean-Air coordinated robotic system for ocean observation and monitoring [1]. The idea of the project is to use Miniature Underwater Gliders (MUGs) to monitor the environment under the ocean, and then use drones to pickup the MUGs and bring them to autonomous surface vehicles where they can recharge [2].

1.2 Specialization Project

This master's thesis is a continuation of the specialization project completed in the fall of 2019 [3]. The specialization project built upon a master's thesis from NTNU in 2019 [4], which was about using a drone to pick up an object from the sea, using DUNE, ArduPilot and computer vision. The specialization project focused on simulating a landing of a drone on a static target using DUNE and ArduPilot. Most of the time spent on the specialization project was used to understand DUNE and the code from [4], and since the project was not using computer vision, removing the parts of the code using CV. In the specialization project the UAV was simulated using DUNE and ArduPilot, where the drone took off, flew in a circle generated by a DUNE task, then landed, however there were not a proper system of landing detection, just that the drone was within a certain distance of the landing point. This master's project expands on the specialization project

by creating a simulation of a moving landing platform, by implementing proper landing detection, and by testing the project with hardware, which required a fair amount of changes. The specialization project report suggest the following further work:

- Implement simulation of moving platform
- Automatic takeoff when guided mode is entered or at least ensure consistent heights
- Potentially look at different control algorithms
- Conduct a field test

With the exception of looking at different control algorithms, all these things are done in this project.

1.3 Literature Review

There has been a fair amount of research into the topic of automatic landing of UAVs. Many of the articles are looking at landing UAVs using computer vision. [5], uses computer vision and Model Predictive Control. The article describes Hardware in the loop simulations, but does not conduct a flight test. In [6], the Robot Operating System (ROS) is used to combine computer vision and traditional navigation sensors, to land on a moving vehicle. Landing tests are performed successfully on both static and moving targets. In [7], GPS and IMU measurments are used along with computer vision to land a Miniature Aerial Vehicle (MAV) on a moving ground vehicle. Landings are performed with speeds of up to 50 km/h.

GPS/INS is used together with a radar altimeter for automatic takeoff and landing in [8]. A Kalman filter is used to integrate the altitude measurements from the radar altimeter and successful tests are performed. In [9] a fixed-wing UAV is landed using only DGPS. No computer vision or inertial sensors are used. The solution works quite well, but is sensitive to wind effects.

There has also been some other work on landing UAVs using the LSTS toolchain. [4], which is described more in Section 1.2, uses computer vision to land on small objects in the sea, but no actual landing tests are performed, only tests in the air, with manual landing. The master thesis [10] is about the use of DUNE for landing on a moving platform. RTKLIB is used for RTK GNSS positioning, where the estimated position is used directly without integration with inertial sensors. A PID controller is used to control the drone, which seems to work fine in the simulations, but not as well in the field test, so it is suggested to look into the use other control algorithms.

1.4 Outline

Chapter 2 - Basic Theory: Outlining the software and hardware used in the project, and explaining theoretical concepts used.

Chapter 3 - Implementation: Presenting the DUNE code used and created for this project.

Chapter 4 - Simulation: Describing some simulations of the project and discussing the results from the simulations.

Chapter 5 - Flight Test: Describing the flight test and going though the results.

Chapter 6 - Conclusion Concluding and suggesting further work.

Chapter 2

Basic Theory

2.1 Software

The software section is mostly reused from the specialization project, but has some additions and corrections.

The software described in this section, with the exception of ArduPilot, is developed by the Laboratório de Sistemas e Tecnologia Subaquática (Underwater Systems and Technology Laboratory) (LSTS), and is part of the LSTS toolchain. LSTS is a part of the University of Porto, Portugal.

2.1.1 DUNE

DUNE is a runtime environment for unmanned on-board software. It is used to write generic embedded software. DUNE can be used for interaction with sensors, actuators, and also communication, navigation and control for a wide variety of autonomous vehicles. DUNE is written in C++, and can be used with different CPU architectures and operating systems [11]. DUNE is divided into tasks that are running concurrently, potentially in different threads. There are two main types of DUNE tasks, Task and Periodic. A task of the type Task will typically not do anything unless it receives a message that it is subscribed to. A task of the type Periodic will run with a set frequency, regardless of whether or not it receives a message.

2.1.2 Neptus

Neptus is a distributed command and control infrastructure for the operation of unmanned vehicles. Neptus supports planning and execution of plans on site and analysis after the mission [12]. Neptus has a Mission Review and Analysis (MRA) console which allows plotting of various data from the mission, and it's possible to watch a replay of the mission. MRA also allows for the conversion of logs to different formats, including the .mat format used by MATLAB. The specialization project [3] only used the Neptus Mission Review and Analysis (MRA) console, to plot data from the simulation after it was over. In this

project Neptus will also be used for testing, in order to look at the UAV and the target on the same map at the same time, and Neptus also allows for the user to update the configuration parameters of the vehicle while DUNE is running which is very practical during testing, as it reduces the need to restart.

2.1.3 IMC

The Inter-Module Communication (IMC) protocol is a message oriented protocol for communication between heterogeneous vehicles, sensors and human operators[13]. IMC is used to send messages between different DUNE tasks, and also to send messages to and from Neptus.

In DUNE, the function `bind<IMC::[type]>(this);` is used to subscribe to an IMC message of the type [type]. Then `consume(IMC::[type])` is used to program what is done when a [type] message is received. An IMC message is sent to the IMC bus by using the `dispatch()` function.

2.1.4 Glued

Glued is a lightweight Linux operating system, designed by LSTS to make it easy to cross compile LSTS software to be used on an embedded system [14]. In this project DUNE is cross compiled for Glued, and Glued is the operating system used for the BeagleBone on-board computer.

2.1.5 ArduPilot

ArduPilot is an open source autopilot for autonomous, unmanned vehicles [15]. ArduPilot can control airplanes, multi-rotors, helicopters, boats and submarines [16]. In this project ArduPilot is used to simulate the multi-rotor drone using the ArduPilot software in the loop simulator, shortened SITL. SITL uses a flight dynamics model in a flight simulator to simulate sensor data for the UAV [17]. DUNE can then retrieve the estimates via the DUNE task `Control.UAV.Ardupilot`, which sends data to ArduPilot through MAVLink messages. MAVLink is a light weight message protocol for communication with drones [18]. When using the ArduPilot SITL mode, the user can control the drone by using MAVProxy, a CLI based GCS, which is included with ArduPilot. MAVProxy uses MAVLink to communicate, and is developed by CanberraUAV [19]. ArduPilot is used both for Simulations and for the field tests, so there doesn't have to be significant changes to DUNE between the two.

Figure 2.1: DJI S1000+

2.2 Hardware

2.2.1 UAV

The UAV used for the flight test is a DJI S1000+, which is an octocopter designed for aerial photography [20]. A picture of the drone is seen in Figure 2.1. The specific drone that is used, however, should not matter, and the software used in this project should work for any multi-rotor drone, provided that it has a flight controller that can run ArduPilot.

2.2.2 BeagleBone

BeagleBone Black is a single board computer produced by the non-profit corporation "The BeagleBoard.org Foundation" [21]. In this project the BeagleBone Black is used to run DUNE on the UAV and on the ground station. The BeagleBone is also used to recieve RTCM corrections data used for RTK, and sends this to the GNSS receiver using UART.

2.2.3 Landing Platform Instrumentation

NTNU-NET-02 is a landing case, created for an other project at NTNU, which acts as the landing platform vehicle. Like the drone it runs DUNE on a Beagle Bone. Two GNSS antennas are connected too the case, and computes the position in the middle of them. The case will then use DUNE to send the GNSS data to the drone by an EstimatedLocalState message which is explained in further detail in Chapter 3.

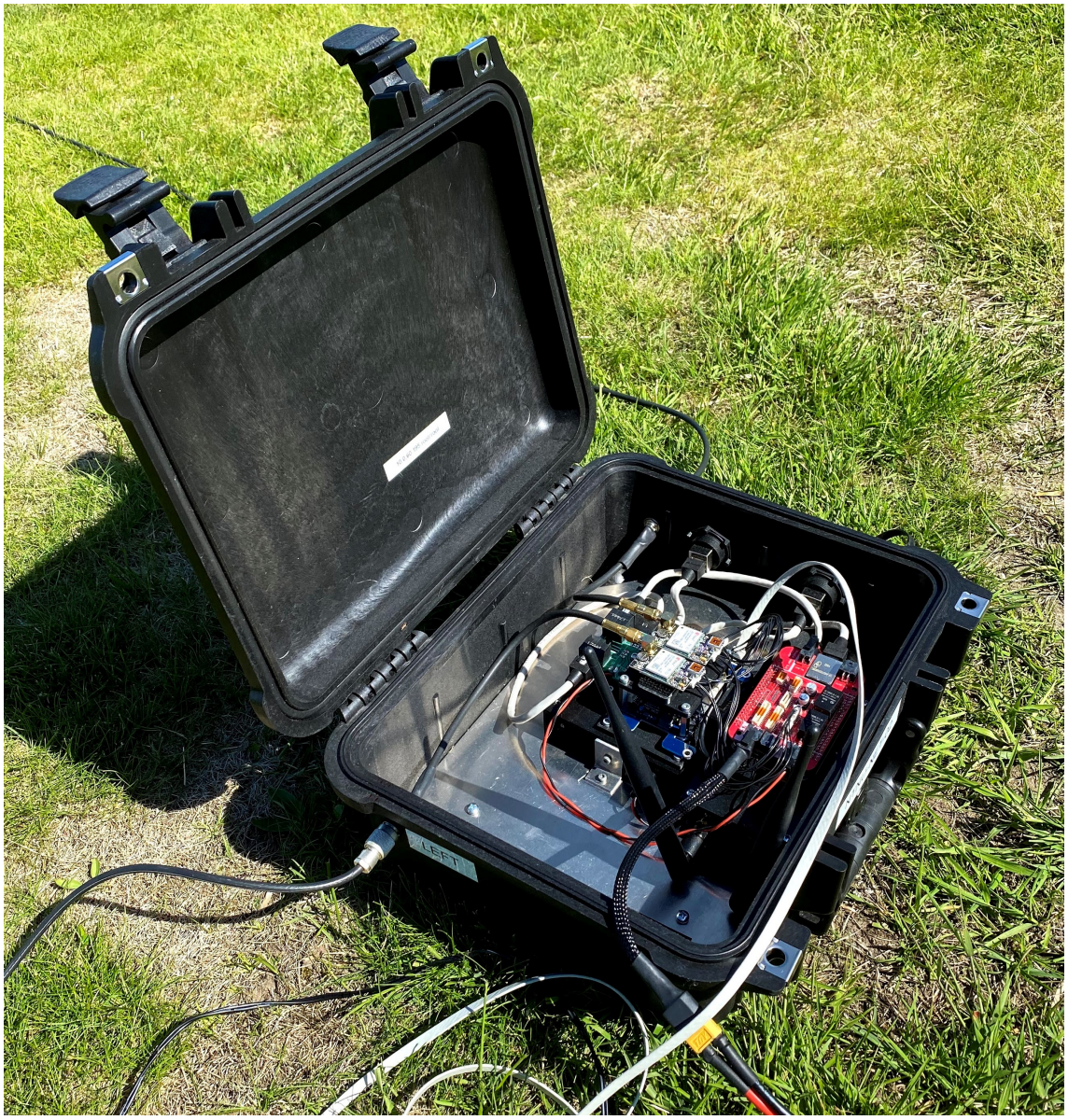


Figure 2.2: Net-02 landing station

2.2.4 PixHawk

The Pixhawk 1 is an open hardware flight controller, originally manufactured by 3DR [22]. The Pixhawk runs ArduPilot, and is used to control the UAV. The Pixhawk is connected to the BeagleBone via UART, which allows DUNE and ArduPilot to communicate.

2.2.5 GNSS Receivers

uBlox ZED-F9P GNSS receivers are used both for the UAV and the landing platform. The receivers have built in RTK functionality [23]. One of the receivers in the landing case acts as the RTK base of the other receiver in the case and for the UAV, which makes relative positioning very accurate. Two Harxon GPS1000 antennas are connected to the landing case.

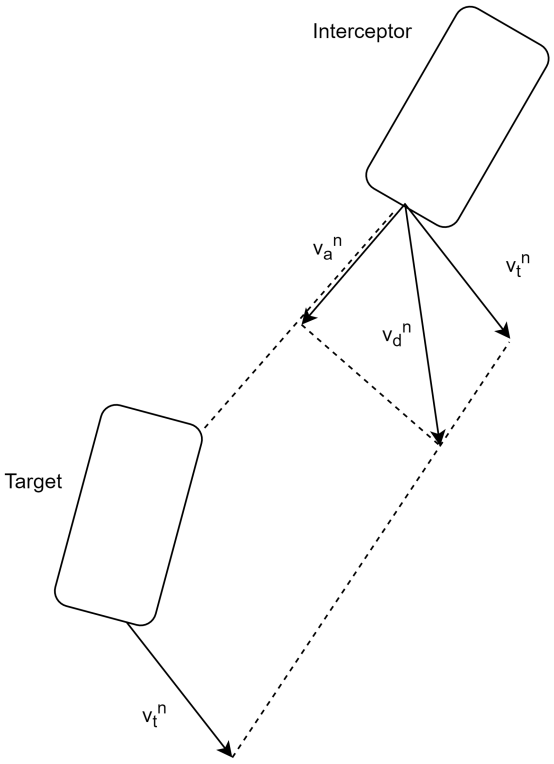


Figure 2.3: Illustration of Constant Bearing guidance

2.3 Theoretical Concepts

This section is mostly new from the specialization project, but Section 2.3.1 on Constant Bearing is reused.

2.3.1 Constant Bearing

In this project Constant Bearing (CB) guidance is used to control the movement of the drone. The reason CB is used is that there already exist a task implementing CB in DUNE from the master's thesis [4].

Constant Bearing guidance is a two-point guidance scheme, where the interceptor is supposed to align the interceptor target velocity $\boldsymbol{v}_a^n$ along the Line Of Sight (LOS) vector between the interceptor and the target. An illustration of CB guidance can be seen in Figure 2.3.
The constant bearing desired velocity is given by:

$$\boldsymbol{v}_d^n = \boldsymbol{v}_t^n + \boldsymbol{v}_a^n \tag{2.1}$$

$$\boldsymbol{v}_a^n = -\kappa \frac{\tilde{\boldsymbol{p}}^n}{||\tilde{\boldsymbol{p}}^n||} \tag{2.2}$$

Where $\boldsymbol{v}_t^n$ is the velocity of the target and $\boldsymbol{v}_a^n = [\dot{N}_a, \dot{E}_a]^\top$ is the approach vector specified

such that the desired approach speed $U_a = ||\boldsymbol{v}_a^n||$ is tangential to the LOS vector and

$$\tilde{\boldsymbol{p}}^n := \boldsymbol{p}^n - \boldsymbol{p}_t^n \tag{2.3}$$

is the LOS vector between the interceptor and the target, $||\tilde{\boldsymbol{p}}^n|| \geq 0$ is the euclidean length of this vector and

$$\kappa = U_{a,max} \frac{||\tilde{\boldsymbol{p}}||}{\sqrt{(\tilde{\boldsymbol{p}}^n)^\top \tilde{\boldsymbol{p}}^n + \Delta_{\tilde{p}}^2}} \tag{2.4}$$

where $U_{a,max}$ is the maximum approach speed toward the target and $\Delta_{\tilde{p}} > 0$ affects the transient interceptor-target rendezvous behaviour. The CB guidance law computes the velocity commands needed to track the target. The superscript ^{n} indicates that the velocity is represented in the NED frame. Note that for the purposes of the specialization project, the target is stationary, making CB guidance equal to Pure Pursuit Guidance [24].

2.3.2 Waves

In order to simulate realistic movement of the moving platform, wave movements must be simulated. A linear wave response approximation can be described as the following transfer function: [25]

$$\eta_{\omega,i}(s) = \frac{K_{\omega,i} s}{s^2 + 2\lambda\omega_e s + \omega_e^2} \omega_i(s) \tag{2.5}$$

Where $\eta_{\omega,i}$ is the wave induced position, $K_{\omega,i}$, λ and ω_e are wave parameters, and w_i is zero mean white noise. The $i = (1, 2, ..., 6)$ indicates that this transfer function apply to all six degrees of freedom. In this project this is simplified to only use three degrees of freedom, x, y, and z.

This can be written in the state space form:

$$\dot{\boldsymbol{x}}_{\omega,i} = \boldsymbol{A}_{\omega,i} \boldsymbol{x}_{\omega,i} + \boldsymbol{E}_{\omega,i}\, \omega_i \tag{2.6}$$
$$\eta_{\omega,i} = \boldsymbol{C}_{\omega,i} \boldsymbol{x}_{\omega,i} \tag{2.7}$$

Which expands to:

$$\dot{\boldsymbol{x}}_{\omega,i} = \begin{bmatrix} 0 & 1 \\ -\omega_e^2 & -2\lambda\omega_e \end{bmatrix} \boldsymbol{x}_{\omega,i} + \begin{bmatrix} 0 \\ K_{\omega,i} \end{bmatrix} \omega_i \tag{2.8}$$
$$\eta_{\omega,i} = \begin{bmatrix} 0 & 1 \end{bmatrix} \boldsymbol{x}_{\omega,i} \tag{2.9}$$

The wave parameters K_ω, λ and ω_e depend on different sea states. Sea state codes are defined in [25] from 0 to 9, where 0 is a quiet sea, and 9 has an observed wave height over $14m$. The gain constant K_ω is defined as:

$$K_\omega = 2\lambda\omega_0\sigma \tag{2.10}$$

Sea State	H_s	K_ω
0	0	0
1	0.05	0.0057
2	0.3	0.0413
3	0.875	0.1205
4	1.875	0.2582
5	3.25	0.4475
6	5	0.6884
7	7.5	1.0326
8	11.5	1.5833
9	14	1.9275

Table 2.1: Sea states and corresponding K_ω

where σ is a wave intensity constant, λ is a damping coefficient, and ω_0 is the dominating wave frequency. The encounter frequency is:

$$\omega_e = |\omega_0 - \frac{{\omega_0}^2}{g} U \cos\beta| \tag{2.11}$$

In order to simplify things, it is assumed that the forward speed $U = 0$ making $\omega_e = \omega_0$. [25] suggest that it is a good approximation to use $\omega_0 = 0.8$ and $\lambda = 0.2573$. Doing this leaves only σ to be calculated. This is done using a modified version of the MATLAB script on page 266-267 of [25] to calculate K_ω for each sea state. The version used in this project is shown in Listing 1. The results of the calculations can be seen in Table 2.1. Note that the average value of H_s for each sea state is used, that is the lowest value plus the highest value divided by two, except for sea state 9, where the minimum value is used, since there is no maximum value.

```
clc
global sigma wo

wo = 0.8;  To = 2*pi/wo;
Hs = 5;
wmax = 3;
w = (0.0001:0.01:wmax)';

% Modified PM
S = wavespec(3,[Hs,To],w,1);  sigma = sqrt(max(S));

lambda = lsqcurvefit('Slin', 0.1, w, S)
Kw=2*lambda*wo*sigma
```

Listing 1: MATLAB script to calculate wave parameters based on [25]

2.3.3 Euler's Method

In order to simulate the wave movements, a simulation method must be used. In this project, Euler's method is used, because it is a relatively simple method to implement.

Euler's method is a numerical integration scheme where the solution is computed from

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(\boldsymbol{y}_n, t_n) \tag{2.12}$$

Where $\boldsymbol{y}_n$ is the value at time t_n, h is the time step and $\boldsymbol{f}(\boldsymbol{y}_n, t_n)$ is the derivative function $\dot{\boldsymbol{y}}$ [26]. Applying this to the wave equation, Equations (2.8) and (2.9):

$$x_{\omega,1_{n+1}} = x_{\omega,1_n} + hx_{\omega,2_n} \tag{2.13}$$

$$x_{\omega,2_{n+1}} = x_{\omega,2_n} - h\omega_e^2 x_{\omega,1_n} - h2\lambda\omega_e x_{\omega,2_n} + hK_{\omega,2}\omega_2 \tag{2.14}$$

2.3.4 WGS84

World Geodetic System 1984 (WGS84) is an Earth Centered Earth Fixed (ECEF) reference system defined by the U.S. Department of Defence. WGS84 is the reference system used by GPS [27]. As the name suggests an ECEF frame has an origin in the center of Earth, and is fixed relative to the Earth. WGS84 ha a z axis pointing towards the International Earth Rotation Service (IERS) Pole and an x axis that points towards the IERS Reference Meridian, i.e. the prime meridian. The y axis completes the right hand frame. The WGS84 reference frame is shown in Figure 2.4 [28]. The reason that WGS84 is important in this project, is because the GPS receivers will send the position in WGS84 coordinates in the form of latitude, longitude and height. The latitude is an angle from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$, from the South Pole to the North Pole. The longitude is an angle from $-\pi$ to π, where 0 is the Prime Meridian, and the negative angle are to the west, and positive to the East. The height is the length of distance between the position and the WGS84 ellipsoidal model.

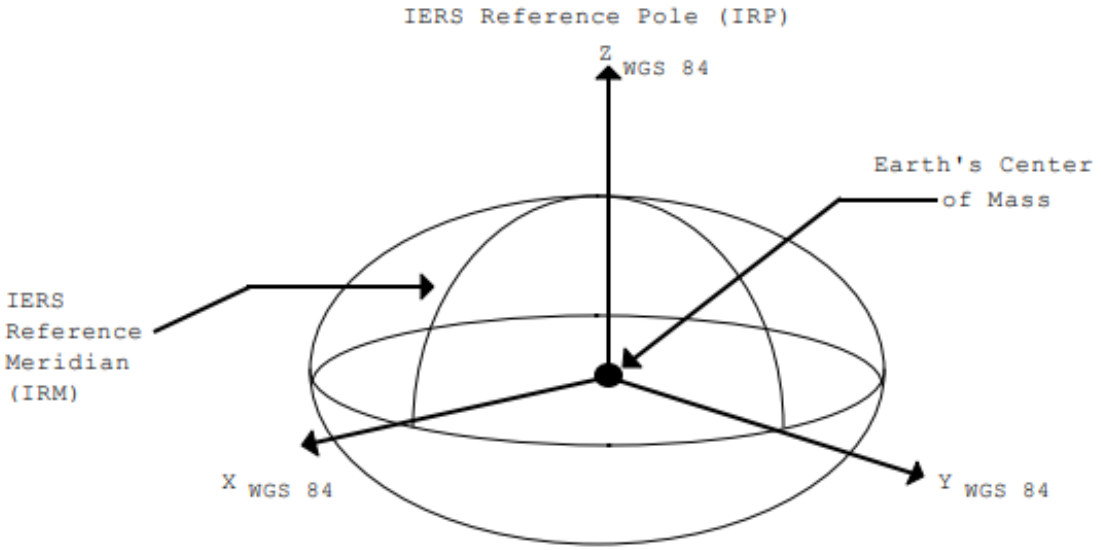


Figure 2.4: WGS84 reference frame

2.3.5 Real Time Kinematic

Real-Time-Kinematic (RTK) is a technique used to enhance the accuracy of position measurements of GNSS signals. RTK is a type of differential positioning, using data from a base station with a known position to correct the measurements [29]. In this project moving base RTK is used. Moving base RTK is a form of RTK where the base is moving, causing less accuracy in absolute position, but retaining good accuracy in relative position, which is what is relevant to this project. With RTK the rover receives raw data from the base via the Radio Technical Commission for Maritime Services (RTCM) format. If there is a long distance between the base station and the rover errors will become significant [30]. An illustration of RTK can be seen in Figure 2.5 [30].

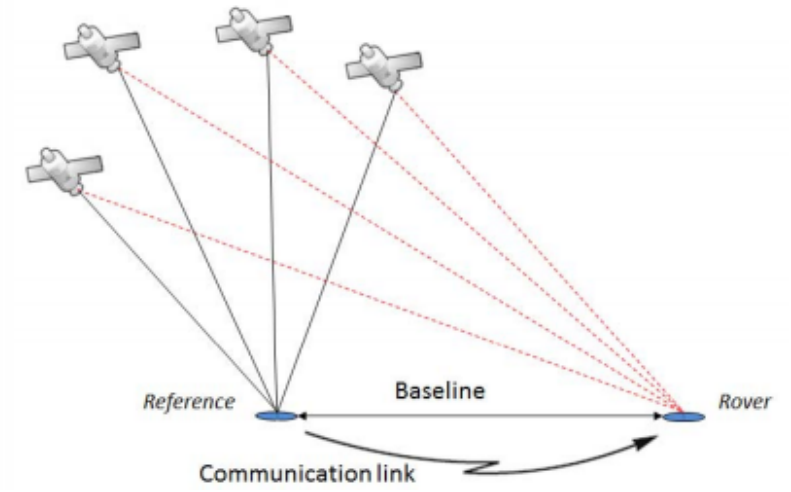


Figure 2.5: Illustration of RTK

Chapter 3

Implementation

Since the core idea of the project remains unchanged from the specialization project, this chapter will have some similarities with it's counter point from there. However none of the sections are entirely reused as there have been updates to every DUNE task. The illustration of the statemachine in Figure 3.1 is reused.

The practical implementation of this project is a continuation of the specialization project [3]. In the specialization project the landing was performed on a stationary target with predefined coordinates. In this project the UAV gets the target position from a landing platform, either simulated or real. In order to receive the position of the landing platform in DUNE, the platform is made its own vehicle. This vehicle will get it's position from GNSS antennas, during the flight test, and for the simulation it will get it from ArduPilot. The landing platform vehicle will then transfer it's position to the UAV. This is done by adding the estimated state of the landing platform to an EstimatedLocalState message and sending it to the UAV. DUNE does not transfer IMC messages between vehicles by default. In order to transfer EstimatedLocalState from one vehicle to another, three tasks are needed. First is Transports.DiscoverVehicle is used for the two vehicles to discover each other, second is Transports.UDP, which transfers messages to another vehicle using UDP. The third task is called Transports.LocalStateTransport, and is a task developed at NTNU which takes EstimatedState messages and adds the to an EstimatedLocalState message that can be sent to another vehicle. It must be noted that vehicle IDs are not persistent between vehicles, so it is not possible for the drone to know which vehicle the message is received from, so the drone should only receive EstimatedLocalState from one vehicle at a time.

In addition to the above the project consists of five DUNE task: "Supervisors.Landing", "Supervisors.Platform", "Control.Path.ConstantBearing", "Simulators.TargetGenerator" and "Simulators.LandingPlatform". The state machine, Supervisors.Landing, handles activation of the other tasks, and controls the overarching logic. Supervisors.Platform does the same for the simulated landing platform. Control.Path.ConstantBearing handles the control of the UAV, and Simulators.TargetGenerator generates targets for the UAV to fly in a circle. Simulators.LandingPlatform is similarly used to generate a path for the simulated landing platform to follow. In addition the task Control.UAV.Ardupilot is used to transmit data between DUNE and ArduPilot.

3.1 ArduPilot Communications Task

The DUNE task that handles communications between DUNE and ArduPilot is called Control.UAV.ArduPilot, and is developed by LSTS and it is included in DUNE. It is a very large task, so it will not be explained in it's entirety, but only the parts that are relevant to this project. One of the things the task does, is that it receives navigation data from ArduPilot and dispatches it as an EstimatedState message. The EstimatedState message contains the estimated position in latitude, longitude and height, in addition to the estimated velocity in the NED frame. The estimates are sent from ArduPilot, which uses an EKF, which takes measurements from the GNSS receivers and IMU. It also contains more data that is not used in this project, like the rotation of the drone. The task is also used to send velocity commands to ArduPilot, to move the drone, as described in Section 3.4.

In the newest version of DUNE, this task has an automatic take off function. However this project is based on an older branch that does not yet have this. This presented two choices, upgrade to the newest version of this file, or change the old version to be able to take off. Because an upgrade could lead to unintended consequences that could potentially take a long time to fix, it was decided to modify the file already in use. Even though there is no dedicated take off function, there is a take off function in the start of the function that is activated by the task receiving a DesiredPath message, if the drone is on the ground. since this function is not used for anything else in this project, it has been modified, so that is only does this take off, and nothing else. The takeoff function will tell ArduPilot to take off, and reach a specified altitude if the drone is on the ground. If it is not it will do nothing. The MAVLink command that is sent is MAV_CMD_NAV_TAKEOFF, which takes in the altitude. This command is described in the MAVLink documentation [31].

3.2 Landing State Machine

The task that includes the state machine governing the landing process is called Supervisors.Landing. The task is subscribed to five messages: IMC::EstimatedState, IMC::AutopilotMode, IMC::VehicleCommand, IMC::CLI and IMC::EstimatedLocalState. IMC::EstimatedState, is a message containing the estimated position, orientation and velocity of the UAV, and is received from ArduPilot. Every time the task receives an estimated state, the switch case structure of the state machine is entered. The state machine in `Supervisors.Landing` has five states: MANUAL, TAKEOFF, TRACK_TARGET, LAND and LANDED. The initial state is MANUAL, which does not do anything with the message, and the UAV is controlled manually by ArduPilot via the MAVLink console. The AutopilotMode message is a message from ArduPilot saying that the autopilot mode has been changed. If the autopilot mode is changed to manual, and the state is not MANUAL, the state is changed to MANUAL. If the autopilot mode is changed to GUIDED, and the state machine is in MANUAL, the state is changed to TAKEOFF.

When the statemachine first enters TAKEOFF, an arming command and then a takeoff command will be sent to ArduPilot, which makes the UAV ascend until it reaches the height specified in the configuration file. In addition the control loops for speed control and the Constant Bearing (CB) task in Section 3.4 are activated, and a Constant Bearing

Target message, with the position at the specified height is sent to the control task. If the drone takes off from the ground, the target is unnecessary, but the take off command doesn't work in the air, so it is necessary when starting the program after takeoff. Once the height is reached, the state TRACK_TARGET is entered, which activates the target generator. The target generator generates targets in a circle, and sends the targets via a ConstantBearingTarget message. The state machine will remain in this state until it receives a CLI message with the text "start landing", or the target generator sends a VehicleCommand message with the flag VC_SUCCESS, both of which will set the state to LAND. In this project the CLI message has not been looked at, but instead a time limit for the target generator is implemented, so that when the target generator has worked for a set time, it will stop generating targets, and send the VC message to initiate the landing.

When entering the LAND state, the target is set as the position of the landing platform, which is received in the form of an EstimatedLocalState message, plus an offset that can be specified in the configuration file. This is described in more detail in Section 3.4. To avoid the drone hitting the platform from the side, the landing is done in two phases. In the first phase the target is set to be 1 m over the platform, i.e. 1 is subtracted from the z-value of the target. This value was originally 0.5 m, but this was changed during testing. When the UAV "lands" in this phase, a Boolean variable, m_approach, is set to false, and phase two is entered, and the real target is used. In the second phase the drone will also use a constant vertical speed, this is described in more detail in Section 3.4. The drone is considered landed in the first phase if the error in the (x, y)-plane is less than a specified radius r divided by 2, and the error in the z direction is less than 50 cm. If the drone leaves the radius r during phase two, m_approach is set to true, and the first phase is reentered. To be considered landed in the second phase the error in the (x, y)-plane needs to be within the radius r and the error in the z direction must be less than 50 cm. A third criterion must also be reached, which is that the vertical speed of the drone must have an absolute value of less than 0.1 m/s. When the UAV lands in phase two, the statemachine enters the state LANDED. When entering this state, the control loops of the drone are deactivated, and an IdleManeuver message is sent to the ArduPilot task, in order to change the autopilot mode to LOITER. When the mode changes to LOITER, the throttle is disarmed, and the state machine is set to MANUAL again. The reason for changing the autopilot mode to LOITER is that ArduPilot does not allow to disarm in GUIDED mode. An illustration of the state machine can be seen in Figure 3.1.

While the task is running it will also dispatch data that will be used for plotting in IMC::ControlParcel messages. ControlParcel messages have three fields of 32 bit floating point numbers, and is not used for anything else in this project, making it convenient for this purpose. The messages being sent contains the position error between the drone and the landing platform, the latitude, longitude and height of the landing platform, the NED position of the drone in relation to the starting point, and the same for the target. Using 32-bit for the latitude and longitude messages means they are not accurate enough to be used for anything important, but they give a general idea of the position. Since the NED positions require a starting point, they are not sent before GUIDED mode is entered. This means that the NED data for the platform is not sent before the drone takes off.

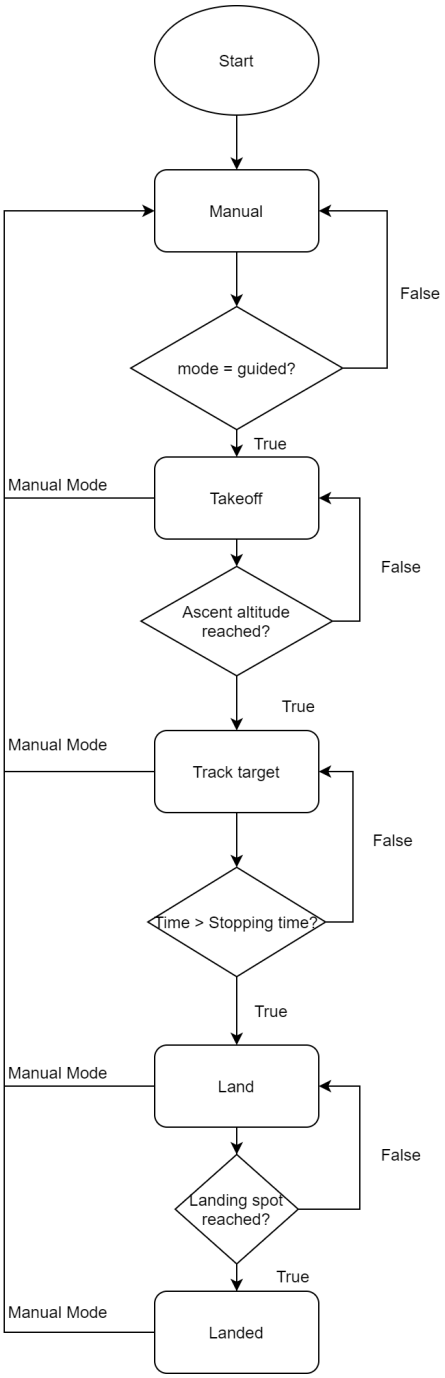


Figure 3.1: Simplified illustration of the state machine

3.3 Platform State Machine

The Landing platform is simulated with its own ArduPilot simulation instance, and its own instance of DUNE. For the purposes of simulation, a supervisor, with a similar structure to Supervisors.Landing, called Supervisors.Landing, is made for the landing platform vehicle. This supervisor has the same structure as the supervisor in Section 3.2, but when it enters the target generation state, called PLATFORM_SIM, it will instead activate the landing platform simulator described in Section 3.6, and follow the target from that task. Once in this state it will never leave, unless the autopilot mode is changed to something other than GUIDED.

3.4 Constant Bearing Controller

The task called Control.Path.ConstantBearing is a periodic task which implements the control law from Section 2.3.1. As long as the task has not been activated or there is no target to follow, the task will not do anything. The task is activated by receiving a VehicleCommand message with the info set to "cbg" and the command set to VC_EXEC_MANEUVER. The target is received via a ConstanBearingTarget message, and the state of the vehicle from an EstimatedState message. When the task is activated, it will send the calculated velocities to `Control.UAV.Ardupilot` via a DesiredControl message. The Ardupilot task sends the velocities to ArduPilot via a set_positon_target_local_ned MAVLink message, with the mask need in order for the message to be interpreted as a velocity. The mask is a set of bits that sets the velocities as active, and position and acceleration as inactive. More information about this can be found in the MAVLink message documentation [31].

In the specialization project [3] this task was unchanged from the previous master's thesis [4], but for this project it has some changes. First it was made to stop if the task does not receive an Estimated State message for more than 2 seconds. However this functionality is difficult to test in the simulator, and has not been sufficiently tested. The second and biggest change is a change to the target message, so that it sends longitude, latitude and height instead of x, y and z coordinates in the NED frame. In an EstimatedState message, the position is given by a coordinate in latitude, longitude and height, and an offset in NED. This change was necessary because the target and the UAV will not have the same NED frame. In addition the GPS hardware will typically output longitude, latitude and height, and not give out NED coordinates. In order to do this targets must first be converted to the WGS84 frame before being sent by using the function Coordinates::WGS84::displace, this function takes in latitude, longitude, height plus an offset in NED and outputs the new latitude, longitude, height coordinate. This task will use the same function on the received Estimated State, in order to ensure that the target and the Estimated State is in the same reference frame. Then the function Coordinates::WGS84::displacement can be used to calculate the difference in position between the two points in NED coordinates.

Another change is that the task allows for keeping constant vertical speed during landing as mentioned in Section 3.2. This is achieved by the task receiving an IMC::CLI message with the text "cbland" which sets a Boolean variable to true, that makes the z value of the

desired velocity equal to the speed specified in the configuration file. If a CLI message with the text "cbmove" is received the computed desired velocity is used again. The reason for why a CLI message is used, is because it is a message that doesn't do anything else, which has a text field, so it is convenient to use for communication between tasks, without having to create a new IMC message. The task also has a configuration parameter that can disable the constant speed functionality, making it always use the computed z velocity. This is needed for the simulation, since the simulated moving target is generated in the air, and there would be nothing making the UAV stop at the target if the constant speed method was used. After testing, this was updated so that only the feed forward velocity is used in the horizontal plane when landing, to avoid the drone tipping over on the ground after landing.

3.5 Target Generator

The target generator is a periodic task, called Simulators.TargetGenerator, that gets activated by receiving a VehicleCommand message with info set to "track", and command set to VC_EXEC_MANEUVER. When the task is activated it will initialise a time t_0. The task loop generates targets in a circle based on the time $t = t_{clock} - t_0$, until the task is deactivated. The x value of the target is given by $r(\cos\phi)$ and the y value is given by $r(\sin\phi)$. Here r is the radius of the circle, and ϕ is given by $\frac{t}{s_r}$ where s_r is the speed reduction parameter. The velocities are given by $v_x = -\sin(\phi)\frac{r}{s_r}$ and $v_y = \cos(\phi)\frac{r}{s_r}$. The circle is generated in the xy-plane, so $v_z = 0$. The targets have to be displaced in relation to the starting position using Coordinates::WGS84::displace as described in Section 3.4. The task is deactivated by receiving a VehicleCommand message with info = "track" and command = VC_STOP_MANEUVER. The task will send this message to itself, if the time t becomes larger than a set parameter t_{stop}. The motivation for including this task, is to avoid having the drone only moving straight up and down, as that would not be very exciting.

3.6 Landing Platform Simulation

This periodic task is used to simulate the movement of a landing platform. The task is called Simulators.LandingPlatform. The position of the target is made up of the simulated wave position from Equations (2.13) and (2.14) in addition to an optional current specified in the configuration file. Since each simulation in this project will take a relatively small time to complete, it is assumed that the current will not change during the simulation, simplifying the current induced position to be just $p_c = v_c t$, but if it should become necessary to change the current during the simulation this would be a relatively simple change to make. The target is also generated a couple of meters above ground, because the simulated drone will crash if it reaches ground level. This height can be changed in the configuration file.

The task is subscribed to two IMC messages, VehicleCommand and EstimatedState. VehicleCommand is used for activation and deactivation of the task, and EstimatedState is used to check the difference in position of the simulated drone and the generated targets for debug purposes. When the task first is activated, the state variables are initialized, and

the time step is stored in a variable by dividing 1 by the task's execution frequency. When the task is initialized, the wave parameters are set, according to Table 2.1 in Section 2.3.2. The wave parameters are also updated if the sea state parameter is changed. The main loop starts with displacing and sending the target for the current time t_n, then random numbers for each of the three DOF are generated. After that the state space variables for the next time t_{n+1} are calculated using Euler's method. Finally the state space variables for t_n are set as the value computed for the next variables, and the loop starts over again.

3.7 Configuration

There are two important configuration files in this project. First and most important is the configuration file for the UAV. A simplified version of this file is shown in Listing 2. At the top, the main ArduPilot configuration file is included, which starts the communication with ArduPilot. Data is sent from ArduPilot at the default rate, which is 10Hz. The next line includes the tasks necessary to receive EstimatedLocalState from the landing platform. After that the main supervisor task is started, with parameters for the initial ascent, the radius of the landing platform, and an offset of the landing target. Next the Constant Bearing Guidance controller is started. The task has two tuning parameters, max approach speed and transient speed modifier. It also has one parameter for whether or not to use constant landing speed, and one for the value of the constant landing speed. The last of the tasks included is the target generator, which has parameters for reducing the speed of the generated circle, the radius of the generated circle and for how long the targets should be generated. Then some settings for the AP-SIL profile of `Control.UAV.Ardupilot` is set. "Convert MSL to WGS84 height" must be true, because ArduPilot outputs data in Mean Sea Level height by default, and the data from the landing case comes in WGS84. Lastly settings for TCP communications between the Beagle Bone and the Pixhawk is set for the Hardware profile.

The second configuration file is for the simulated landing platform and is shown in Listing 3. This is very similar to the configuration file for the drone, but includes the landing platform simulator instead of the target generator. The parameters for the platform simulator are the sea state, currents in x and y direction and randrange. Randrange is a parameter which determines the range of possible values generated by the number generator. There is also a parameter for the base altitude of the generated path. Multiple instances of ArduPilot can be run on the same machine, but for DUNE to connect to the correct instance, the port must be changed so that the landing platform connects to ArduPilot with instance 2 instead of the default instance 0.

```
## ntnu-hexa-Land config file.
# Main ardupilot file
[Require uav/arducopter.ini]
#Transfer of EstimatedLocalState
[Require uav/RCFormation/common.ini]
#============================================================
# Created DUNE tasks
[Supervisors.Landing]
Enabled                             = Always
Entity Label                        = Supervisor_Landing
Initial ascent                      = 20
Platform Radius                     = 2
Offset x                            = 0
Offset y                            = 0
Offset z                            = 0
Debug Level                         = Debug

[Control.Path.ConstantBearing]
Execution Frequency                 = 10
Enabled                             = Always
Entity Label                        = Constant_Bearing_Guidance
Max approach speed                  = 1
Transient speed modifier            = 1
Landing Speed                       = 0.5
Use constant landing speed          = True
Debug Level                         = Debug

[Simulators.TargetGenerator]
Execution Frequency                 = 10
Enabled                             = Always
Entity Label                        = CBG_Target_generator
Speed reduction                     = 5
Radius                              = 10
Stopping time                       = 30
Debug Level                         = Debug
# ============================================================
# ArduPilot Options
[Control.UAV.Ardupilot/AP-SIL]
Enabled                             = AP-SIL
Ardupilot Tracker                   = False
Debug Level                         = Debug
Convert MSL to WGS84 height         = True
TCP - Address                       = 127.0.0.1
TCP - Port                          = 5762 #Instance 0, ArduPilot default
[Transports.SerialOverTCP]
Enabled                                 = Hardware
Serial Port - Device                    = /dev/uart/5
Serial Port - Baud Rate                 = 921600
```

Listing 2: Simplified version of configuration file for the drone

```
##ntnu-hexa-003 Config file
# Main ardupilot file.
[Require uav/arducopter.ini]
#Transfer of EstimatedLocalState
[Require uav/RCFormation/common.ini]
#============================================================
[Supervisors.Platform]
Enabled                                 = AP-SIL
Entity Label                            = Supervisors_Platform
Initial ascent                          = 2
Debug Level                             = Debug

[Control.Path.ConstantBearing]
Execution Frequency                     = 10
Enabled                                 = AP-SIL
Entity Label                            = Constant_Bearing_Guidance
Max approach speed                      = 2
Transient speed modifier                = 5
Debug Level                             = Debug

[Simulators.LandingPlatform]
Execution Frequency                     = 100
Enabled                                 = AP-SIL
Entity Label                            = Sim_LandingP
Seastate                                = 4
Current x                               = 1
Current y                               = 0.5
Randrange                               = 4
Altitude                                = 3
Debug Level                             = Debug
#============================================================
[Control.UAV.Ardupilot/AP-SIL]
Enabled                                     = AP-SIL
TCP - Port                                  = 5782 # Instance 2
Ardupilot Tracker                           = False
Convert MSL to WGS84 height                 = True
```

Listing 3: Simplified version of configuration file for the simulated landing platform

Chapter 4

Simulation

4.1 Simulation execution

In order to start the simulation of the drone, two console windows are opened and DUNE is run with the configuration file ntnu-hexa-Land.ini, and the profile AP-SIL in the first window, like in Figure 4.1. In the second window, ArduPilot is run from the folder ArduCopter like in Figure 4.2. The console and map arguments are optional. To start the simulation of the landing platform, the same is done in two new console windows, but dune is run with ntnu-hexa-003, and ArduPilot is run with the additional argument -I2, so that the correct instances of DUNE and ArduPilot connects with each other.

```
odin@odin-VirtualBox:~/uavlab/build$ ./dune -c ntnu-hexa-Land -p AP-SIL
```

Figure 4.1: Running DUNE

```
odin@odin-VirtualBox:~/uavlab/ardupilot/ArduCopter$ sim_vehicle.py --console --map
```

Figure 4.2: Running ArduPilot

The simulation starts at the default location, which is at a small runway outside of Canberra, Australia. At this location there is a hill southwest of the runway, so when simulating here, either the current should be set to go north and/or east, or a high altitude offset should be used. When both DUNE tasks have initialized, each DUNE console will display a message that the other vehicle has been discovered. It takes some time for the GPS measurements to converge. When `APM: EKF IMU 0 is using GPS` and `APM: EKF IMU 1 is using GPS` shows up in both consoles, the UAV and the landing platform is ready for takeoff. The next step is to start the landing platform by typing `mode guided` into the MAVProxy console, which makes the platform arm the throttle and take off, and then it starts the simulation, which will cause the platform to "float" with the current like in Figure 4.3. After that `mode guided` should be entered into the other MAVProxy console, which will arm the UAV and cause it to take off. When the initial ascent height is reached, the target generator is switched on, and the drone will fly in a circle, like in

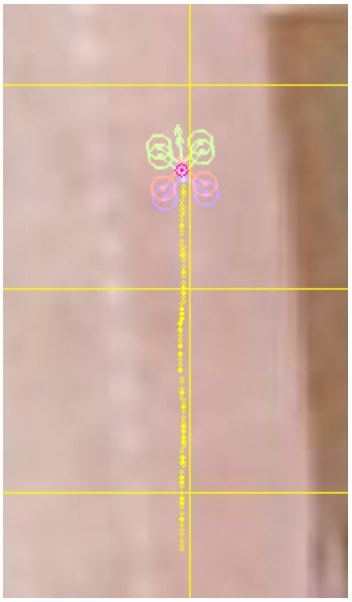


Figure 4.3: Screenshot of the ”landing platform” drifting with the current

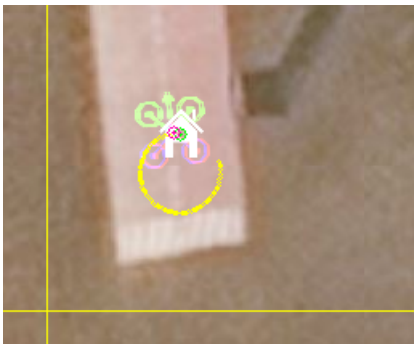


Figure 4.4: Screenshot of the UAV flying in a circle

Figure 4.4. When the stopping time is reached, the UAV will start to follow the landing platform, by EstimatedLocalState messages. When the target is reached, the LANDED state is entered, however because the landing target has to be generated above ground, the drone can't be disarmed, so when simulating the drone will just stop in the air. A screenshot of the console output of the drone's DUNE console can be seen in Figure 4.5.

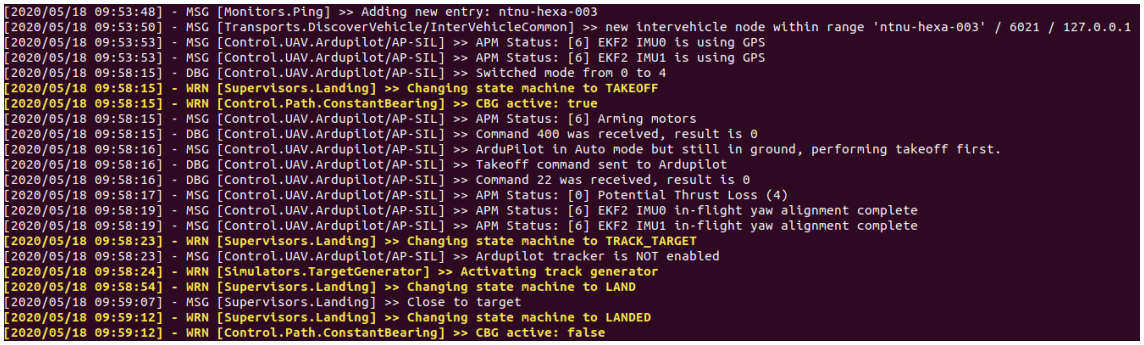


Figure 4.5: Output of the DUNE console during simulation

In order to ensure that the program also works for the flight test, some simulations were also performed with a simulated version of the ntnu-net-02 case. The net-02 simulation is started using DUNE like the other simulations, but does not use ArduPilot. The ntnu-net-02 simulation is developed at the NTNU UAVLab, and has not been changed in any way for this project, except for the start location, which has been changed to Australia, like the UAV simulation. The simulation replays real data recorded from the Motion Reference Unit of a ship at sea. Since this simulator does not use ArduPilot, it does not crash when it goes into the ground, meaning it can be simulated to travel roughly at ground level. This means that the constant landing speed can be tested for the moving target, provided that the terrain is more or less flat, which it happens to be northeast of the runway, and the simulated log travels in the a northeast direction. The simulator also has an option to simulate without horizontal movement, causing the platform to go up and down at the starting position.

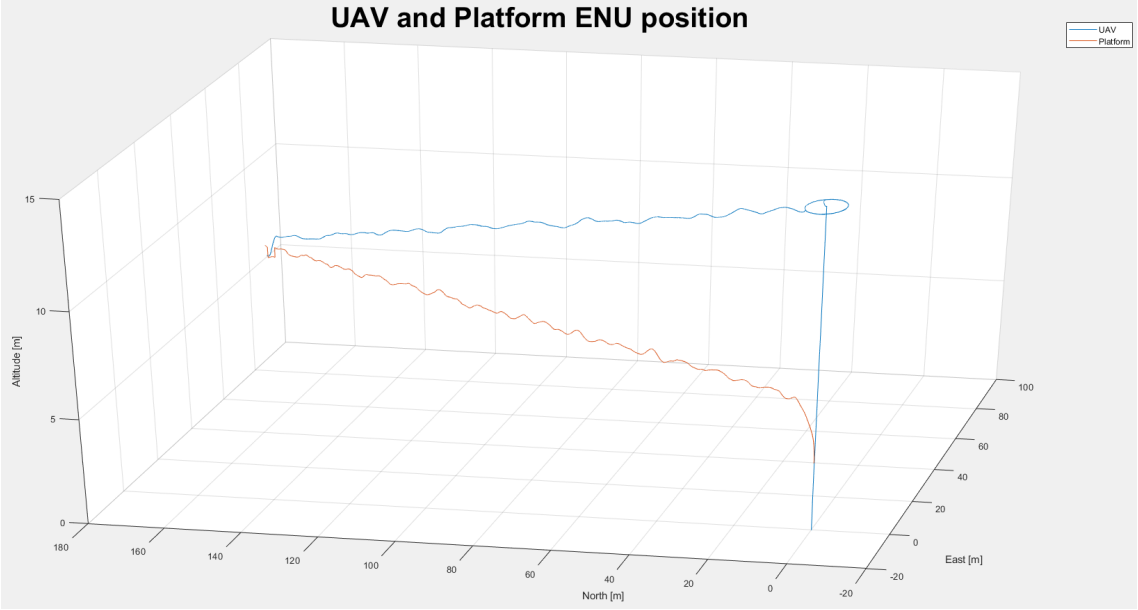


Figure 4.6: 3D plot of the path of the drone and landing platform in the ENU frame

4.2 Simulation Results

The simulation results are reviewed by using the Neptus MRA console to convert the logs to the .MAT file format, readable by MATLAB. Then a MATLAB script is used to plot the data sent with the ControlParcel messages. Figure 4.6 shows a 3D plot of the path of the UAV and the landing platform during the simulation in the East-North-Up (ENU) frame. ENU is used so that the drone moving upwards shows as moving upwards in the plot. In this particular simulation, the configuration had a generated circle radius of 5 m, an initial ascent altitude of 15 m. The landing platform had an altitude of 3 m, a sea state of 4 and a current of 1 m/s and 0.5 m/s in the x and y direction respectively. The plot shows the drone taking off, flying in a nice circle, and then flying towards the landing platform and "landing" in the air when it reaches the platform.

In Figure 4.7 one can see the position error between the UAV and the landing platform over time. The reason that the error in the z axis starts out at -1 m is the initial offset to the target, which is in place as long as the drone is not close to landing. This causes a jump in the z error when the true position of the platform is used, and a second jump, back when the drone has "landed".

While simulating the UAV different tuning parameters where tried to see the effects. It was noticed that circular motions with a velocity of larger than 2 m/s, that is $\frac{r}{s_r} > 2$ m/s, would become unstable. Figure 4.8, shows the ENU path of UAV during a simulation with $r = 10$ and $s_r = 2$ giving a velocity of 5 m/s. The circle motion lasted for 30 s. The plot shows that the drone spirals out of control, and is not able to perform the circle motion properly. Fast circle motions are unlikely to be necessary for the task of picking up and returning the MUGs however, so this shouldn't be a big problem.

Figure 4.7: Position error between drone and landing platform

Now the simulation of ntnu-net-02 will be looked at. Figure 4.9 shows the position error for a simulation of ntnu-net-02 without horizontal movement. The plot shows that the simulation went well, and without problems, landing nicely on the ground before disarming.

Looking at the plot for the simulation with horizontal movement in Figure 4.10, it is seen that the landing was not completed successfully, but there is instead a zigzag movement at the end. The plot for the z axis shows that this zigzag movement takes place between the landing platform and five meters below it. This happens because the ground at the location the landing platform has reached when the drone is attempting to land is five meters below the landing strip. This causes the drone to land on the ground five meters below the platform, which means that the landing is not registered. When the UAV has hit the ground and is stopped, the platform will quickly travel away, causing the drone to leave the platform radius, which makes the drone go above the platform again. This problem is difficult to do anything with, because of the varying ground height in the simulator, however it is also a problem that only exists in the simulator, as in a real flight the UAV would land on the platform instead of on the ground below it.

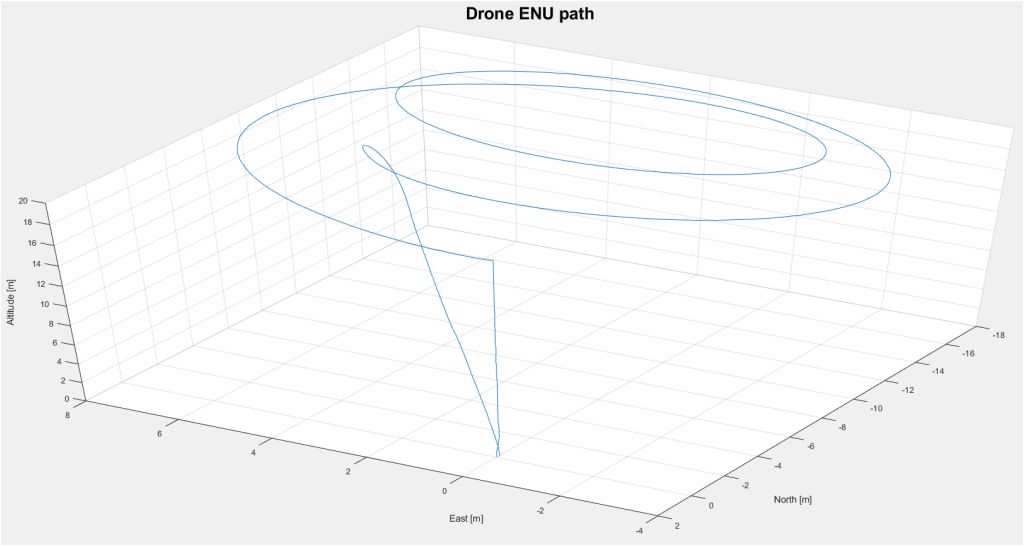


Figure 4.8: ENU path of unstable simulation

Figure 4.9: Position error for horizontally static net-02 simulation

Figure 4.10: Position error for horizontally moving net-02 simulation

Chapter 5

Flight Test

The flight test took place at Udduvoll airfield, in Melhus south of Trondheim. The ground station PC was set up in a van, with a router, and an antenna to transfer data over the network to the UAV. The van can be seen in Figure 5.1. The ntnu-net-02 case with it's GNSS antennas was laid out on the airfield as shown in Figure 5.2, where the target is in the middle between the antennas. A laptop running Neptus was used to set necessary configuration parameters, and to see that a connection had been established between net-02 and the drone ntnu-hexa-Land. The test was executed by having the pilot take off manually, and switching to guided mode when in the air. Once in guided mode, the drone would reach the specified height, fly in a circle, and then attempt to land. All the tests described in this thesis used a platform radius of 2 m, an initial ascent altitude of 20 m, a circle radius of 10 m and a speed reduction of 5.

Figure 5.1: Picture of the van and the drone. Photo: Martin L. Sollie

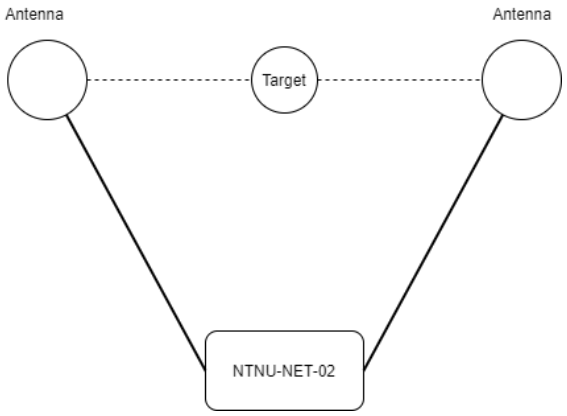


Figure 5.2: Illustration of NTNU-NET-02 laid out on the ground, viewed from above

5.1 Initial Testing

During the first few tests, the drone would tip over during landing. By looking at the logs, it was discovered that this was due to the drone having a desired horizontal velocity. Because of this the CB controller task was changed, so that it does only use the feed forward velocity $\boldsymbol{v}_t^n$, and not the proportional velocity $\boldsymbol{v}_a^n$, in the final landing phase. Because of noise in the position estimates, and the fact that the landing legs on the drone are relatively long, the constant landing phase would start a bit late, so the program was changed to start the constant landing speed 1 m above the target, instead of the original 0.5 m. Figure 5.3, shows a plot of the desired velocities sent to ArduPilot by DUNE at the end of a test before these changes were made. The plot shows that there is a desired velocity in the x direction, for the entire constant velocity phase until the drone tips over, and the pilot switches modes, and turns off the throttle, disabling the program, and setting the desired velocity to 0. The plot also shows a spike in desired velocity near the end. Figure 5.4 shows that this is due to a spike in the target velocity sent from the net-02 case.

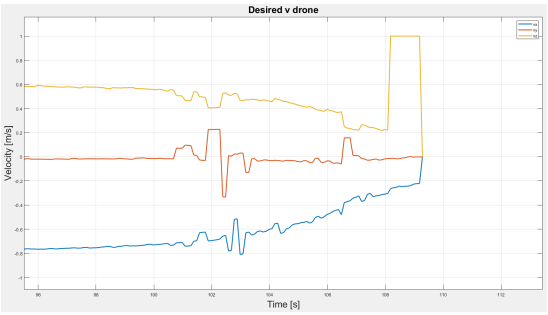


Figure 5.3: Plot of Desired Velocity at the end of a test

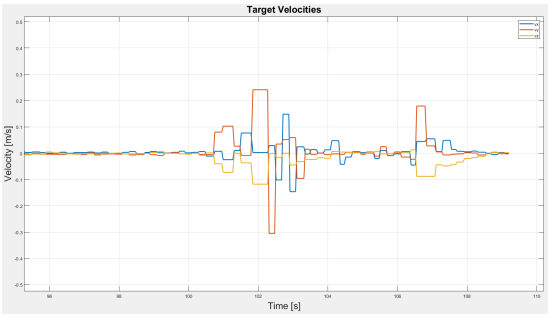


Figure 5.4: Spike in target velocities.

5.2 Successful Tests

After the changes were implemented, three tests were performed. For all three tests the UAV would land properly without tipping over, and then disarm properly. The results from one of these tests will be discussed now. Figure 5.5 shows the actual velocity and the desired velocity of the drone, during a successful test. First the sinusoidal velocity of the circle movement is shown. During this simulation the circle movement lasted for thirty seconds, which was roughly one revolution. After the circle motion, the drone moves towards the point 1 m above the landing position. When this point is reached the x and y velocities goes close to zero, and the z velocity goes to the constant 0.5 m/s, until it is stopped by the ground. There are still some spikes in desired velocity from the target velocity, but they didn't affect the landing noticeably. This is probably because the spikes last for a relatively short time, so they are corrected before they could cause any harm.

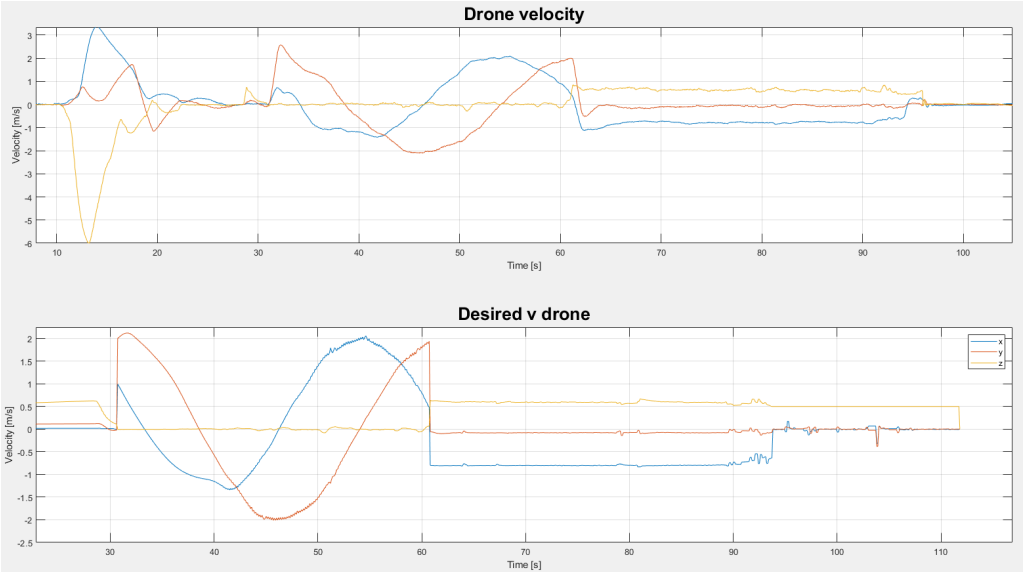


Figure 5.5: Velocity and desired velocity from a successful landing

Figure 5.6 shows the ENU path of the drone for this test. Guided mode was entered slightly above the initial ascent altitude of 20 m, and since the zero position of the dispatched NED frame is in the start position, most of the test takes place with a positive z value in NED, which shows up as negative in the plot, since the z axis is flipped for ENU.

This particular test took place after another successful test, which means that the UAV was standing in the landing point before the pilot took off. The plot of the position errors in Figure 5.7 shows this since it also shows data from before take off. The configuration for this test used a circle radius of 10 m, a speed reduction of 5, a stopping time of 30 s, a max approach speed of 1 m/s, a transient speed modifier of 1 and a constant landing speed of 0.5 m/s. The plot shows that guided mode was entered about 15 m north, 5 m east and roughly 20 m above the landing point. Then the UAV completed the circle motion for thirty seconds, before going in for landing. When the point 1 m above the landing is reached, the constant speed phase is entered, and the drone lands on the ground and disarms. A picture of the drone after landing is seen in Figure 5.8.

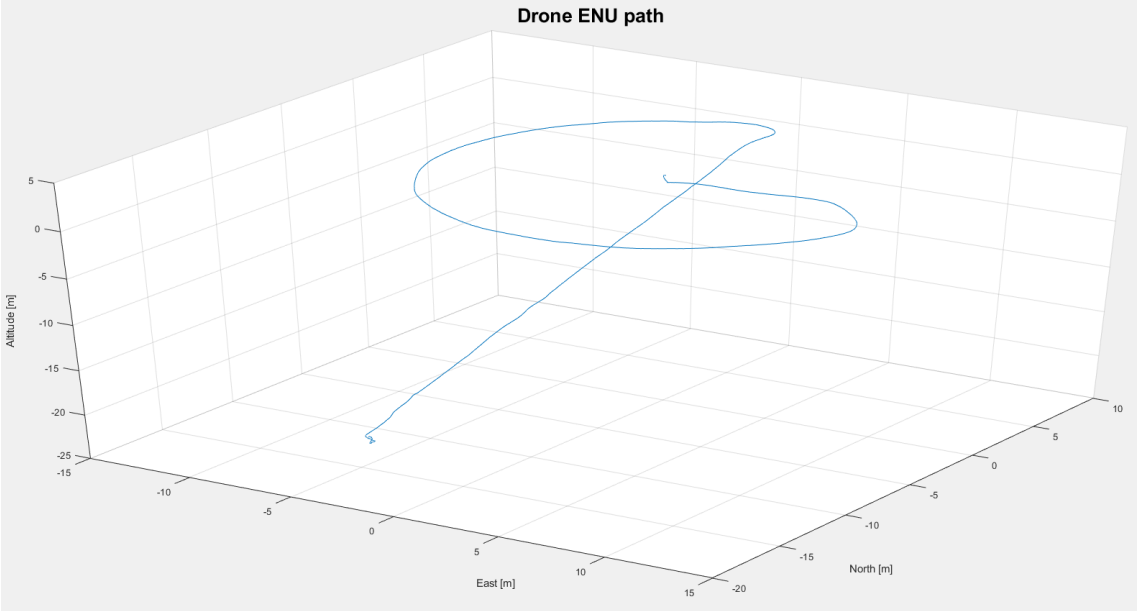


Figure 5.6: UAV path from a successful landing, in ENU

Figure 5.7: Position errors for the successful landing

Figure 5.8: Picture of the drone landed between the antennas. Photo: Martin L. Sollie

Chapter 6

Conclusion

The thesis presents the implementation of algorithms for automatic landing of a multi-rotor UAV, using the LSTS toolchain and ArduPilot. The algorithms are intended for automatic landing on a floating maritime platform, but could also be used for landing on other stationary or moving platforms.

A simulation of a floating platform is implemented in DUNE, and the landing algorithms are tested with the simulation. The results show that the landing works satisfactorily, but there are some inherent problems to using the ArduPilot simulator and DUNE to land at a moving target, as there is no simulation of physical contact between the two simulations.

A flight test is performed using the hardware described in Section 2.2. Most of the time used for testing was spent on fixing the problem of the drone tipping over during landing. Once the problems were fixed the UAV was eventually able to land successfully and repeatably. The tests were performed on a static target on the ground, instead of on a floating maritime platform as originally intended. Without the access to a moving platform for testing, certain aspects of problem could not be tested and evaluated, such as tracking the horizontal position of a moving platform and landing with significant heave, roll, and pitch motions from waves.

6.1 Further work

The most obvious thing that should be done is that a flight test should be conducted using a moving target, as was originally intended to be done in this project. It should be considered to test with a smaller platform radius, and test how small the radius can be while the UAV is still able to land. This would be best tested with a moving platform. In addition it could be useful to look into if it would be possible to simulate physical contact between the UAV and the landing platform in the ArduPilot simulation, or find some other way to simulate physical contact.

Bibliography

[1] "OASYS home page." `https://blogg.hioa.no/oasys/`. Online: Accessed: 2020-06-14.

[2] "Billigere og tryggere miljøovervåkning." `https://www.oslomet.no/forskning/forskningsnyheter/billigere-og-tryggere-miljoovervakning`. In Norwegian. Online: Accessed: 2020-06-14.

[3] O. Linga, "Automatic landing of multi-rotor on moving platform," 2019. Specialization Project, NTNU.

[4] J. L. G. Joberg, "Multirotor pickup of object in the sea," Master's thesis, NTNU, 2019.

[5] Y. Feng, C. Zhang, S. Baek, S. Rawashdeh, and A. Mohammadi, "Autonomous landing of a uav on a moving platform using model predictive control," *Drones*, vol. 2, no. 4, 2018.

[6] V. Line, "Autonomous landing of a multirotor uav on a platform in motion," Master's thesis, NTNU, 2018.

[7] A. Borowczyk, D.-T. Nguyen, A. P.-V. Nguyen], D. Q. Nguyen, D. Saussié, and J. L. Ny, "Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle**this work was partially supported by cfi jelf award 32848 and a hardware donation from dji.," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10488 – 10494, 2017. 20th IFAC World Congress.

[8] A. Cho, Y. Kang, B. Park, C. Yoo, and S. Koo, "Altitude integration of radar altimeter and gps/ins for automatic takeoff and landing of a uav," in *2011 11th International Conference on Control, Automation and Systems*, pp. 1429–1432, 2011.

[9] Am Cho, Jihoon Kim, Sanghyo Lee, Sujin Choi, Boram Lee, Bosung Kim, Noha Park, Dongkeon Kim, and Changdon Kee, "Fully automatic taxiing, takeoff and landing of a uav using a single-antenna gps receiver only," in *2007 International Conference on Control, Automation and Systems*, pp. 821–825, 2007.

[10] V. Krivokapic, "Automatic landing of multi-rotor on moving platform," Master's thesis, NTNU, 2019.

[11] "LSTS site about DUNE." `https://lsts.fe.up.pt/toolchain/dune`. Online: Accessed: 2019-12-16.

[12] "LSTS website about Neptus." https://www.lsts.pt/toolchain/neptus/. Online: Accessed: 2019-10-03.

[13] R. Martins, P. S. Dias, E. R. B. Marques, J. Pinto, J. B. Sousa, and F. L. Pereira, "IMC: A communication protocol for networked vehicles and sensors," in *OCEANS 2009-EUROPE*, pp. 1–6, May 2009.

[14] "LSTS site about Glued." https://lsts.fe.up.pt/toolchain/glued. Online: Accessed: 2020-05-22.

[15] "ArduPilot Documentation." http://ardupilot.org/ardupilot/. Online: Accessed: 2019-11-25.

[16] "ArduPilot About page." http://ardupilot.org/about. Online: Accessed: 2019-11-25.

[17] "ArduPilot page about Software in the Loop." http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Online: Accessed: 2019-12-02.

[18] "MAVLink Developer Guide." https://mavlink.io/en/. Online: Accessed: 2019-12-09.

[19] "ArduPilot page about MAVProxy." http://ardupilot.github.io/MAVProxy/html/index.html. Online: Accessed: 2019-12-05.

[20] "Spreading Wings S1000+ - dji.com." https://www.dji.com/no/spreading-wings-s1000-plus. Online: Accessed: 2020-06-17.

[21] "BeagleBoard about page." https://beagleboard.org/about. Online: Accessed: 2020-05-22.

[22] "Pixhawk 1 flight controller." https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html. Online: Accessed: 2020-06-17.

[23] "ZED-F9P module." https://www.u-blox.com/en/product/zed-f9p-module. Online: Accessed: 2020-06-21.

[24] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, 2011.

[25] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, 2020. Draft manuscript handed out fall 2019.

[26] O. Egeland and J. T. Gravdahl, *Modeling and Simulation for Automatic Control*. Marine Cybernetics AS, 2002.

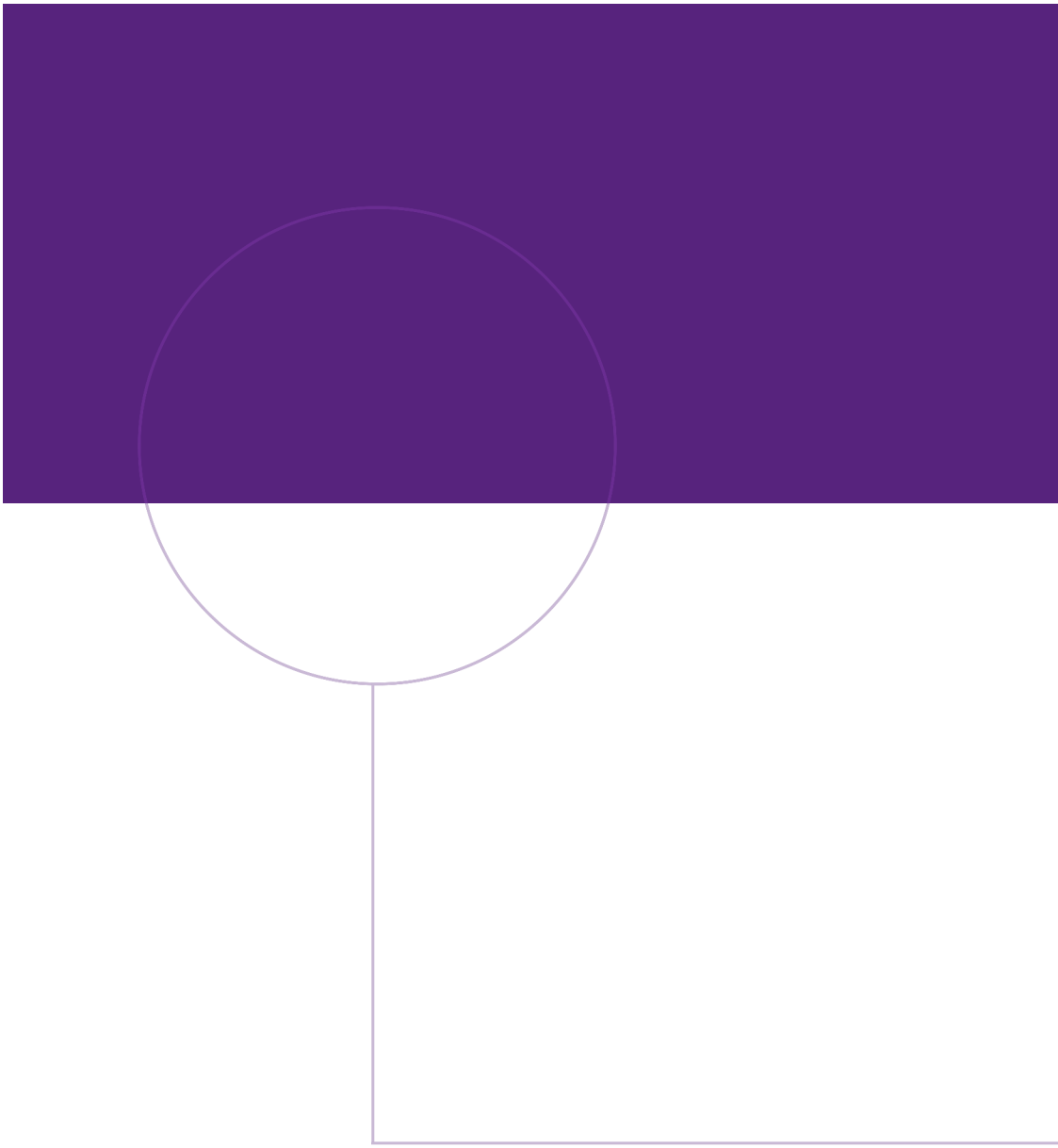
[27] "Qinsy World Geodetic System 1984 (WGS84)." https://https://confluence.qps.nl/qinsy/9.1/en/world-geodetic-system-1984-wgs84-182618391.html. Online: Accessed: 2020-06-08.

[28] National Imagery and Mapping Agency (NIMA), "NIMA TR8350.2 Third edition amendment 1," tech. rep., NIMA, January 2000.

[29] L. Mæhlum, "RTK - Real Time Kinematic." https://snl.no/RTK_-_Real_Time_Kinematic. Store norske leksikon på snl.no, Online: Accessed: 2020-06-20.

[30] N. Sokolova, "Lecture notes from TTK5 Kalman Filtering and Navigation," 2019. NTNU.

[31] "MAVLink Message Documentation." `https://mavlink.io/en/messages/common.html`. Online: Accessed: 2019-12-09.



NTNU

Norwegian University of
Science and Technology