NTNU – Trondheim
Norwegian University of
Science and Technology

Path Planning for Search and Rescue Mission using Multicopters

Håvard Lægreid Andersen

Master of Science in Engineering Cybernetics (2 year))
Submission date: June 2014
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION SHEET

Name:	Håvard Lægreid Andersen
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Baneplanlegging for søk- og redningsoppdrag med multicopter
Thesis title (English):	Path Planning for Search and Rescue mission using multicopters

Thesis Description: Design and investigate methods for search and rescue operations performed by UAVs, particularly multicopters (quad/hex).

The following items should be considered:

1. Overall system and mission description with detailed module interaction schemes and protocols.
2. Field of view. Find/suggest algorithm to calculate which area that is covered by the on board camera when given the multicopter's position and attitude (with and without gimbal?). The algorithm should be implemented in the ground-station software, giving the operator a graphical presentation of which area the multicopter has covered.
3. Path planning for a Search & Rescue mission. Incorporate map-data and topography, and study optimal trajectories (or paths) for different scenarios. Discuss how the onboard camera should be guaranteed to observe the entire area of interest.
4. Optimal altitude. Discuss which altitude the search should be conducted at, considering the field of view, size of the subject in the camera frame and physical/technical limitations.
5. Implementation of the path-planning scheme in the ground station and on the UAV payload computer.
6. Verify the proposed path-planning scheme by simulations and experiments. Conduct missions for different stationary and moving subjects, and discuss the performance of different trajectories/pattern in different scenarios.
7. Conclude findings in a report. Include Matlab/C-code as digital appendices together with a user-guide.

Start date:	2014-01-20
Due date:	2014-06-16

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Tor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
Co-supervisor:	MSc Kristian Klausen, Dept. of Eng. Cybernetics, NTNU

Abstract

This thesis considers path planning for a low-cost multicopter used in the search part of a search and rescue mission. Search patterns or trajectories are considered and evaluated through simulations in MATLAB. How to place the onboard camera in order to cover as much area as possible and which altitude that gives the most area coverage without making the subjects too small to detect is discussed.

The proposed search patterns are implemented in the existing software structure used in this project. The implementation is tested using a ArduPilot software in the loop simulator, and by flight tests using a hexacopter.

Sammendrag

Norwegian translation of the abstract

Denne masteroppgaven tar for seg baneplanlegging for et lavkost multicopter som kan brukes til å hjelpe til i søket under et søk- og redningsoppdrag. Forskjellige søkemønstre, eller baner, vurderes og evalueres gjennom simuleringer utført i MATLAB. Hvordan kameraet ombord skal plasseres for a dekke så mye av søkeområdet som mulig, og hvor høyt multicopteret skal fly for å dekke mest mulig med et bilde, uten at målene blir for små til at det kan gjennkjennes, drøftes.

De forslåtte søkemønstrene implementeres i den eksisterende software-strukturen som brukes i dette prosjektet. Implementasjonen blir testet gjennom en *"ArduPilot software in the loop"*-simulator, og gjennom flytester med et hexacopter.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree MSc. at The Norwegian University of Science and Technology.

I would like to thank my supervisor Professor Tor Arne Johansen for giving me an interesting and challenging project to work on. I would also like to thank Ph.D. student Kristian Klausen for valuable help and feedback during the project, and Lars Semb for help with the flight tests.

Lastly, I want to thank my family for their support throughout my studies.

Trondheim, June 15, 2014

Håvard Lægreid Andersen

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Search and rescue operations can greatly benefit from the use of autonomous UAVs to survey the environment and collect evidence about the position of a missing person [29]. When searching for a missing person, factors such as time and manpower to search large areas, may often be limited. An autonomous UAV that has the ability to scan large areas and recognize persons could be of great assistance in such a scenario, and can thereby increase the possibility of a successful outcome.

The Norwegian Joint Rescue Coordination Centres (JRCC) handle more than 1500 maritime incidents each year in the Norwegian Sea and surrounding waters. Of these incidents, a substantial part involves both search and rescue [4].

1.1 Background and Motivation

This thesis is a part of a project where the possibility helping a search and rescue mission with a low cost UAV is considered. The work in this project is motivated by increasing the possibility for a successful outcome of a search and rescue mission.

In [12], a search and rescue scenario where a severely autistic young man is lost in the Monongalia National Forest when hiking with his parents is described. This search required a large number of searchers. At the height of the operation, more than 400 searchers were involved. It took four days of searching before the subject was found 350 meters East of the point last seen.

This shows the need, and potential for optimization of the search phase of a search and rescue mission, as this frequently is the element of the search and rescue mission that has proven to be most difficult [12].

In a typical search and rescue scenario, UAVs can be deployed in the area of interest, perform sensory operations to collect evidence of the presence of a victim, and report their collected information to a remote ground station or rescue team [29]. A complete mission scenario, of a UAV search and rescue scenario can be found in [10]. The mission is here divided into two separate legs. The first leg is the search part of the scenario when the UAVs should cooperately scan large regions in attempt to identify injured persons. During this leg, a saliency map is created that can be given to emergency services, or used as a basis for the second leg. In the second leg several UAVs are used to deliver food, water and medical supplies to the injured persons discovered in the first leg, that are awaiting help.

In this project, the focus is on the search phase of a search and rescue mission, i.e leg 1 from [10]. It was chosen to use a multicopter for searching as this costs less than a fixed-wing UAV, it is easier to operate and requires less space for take-off and landing. It also has the advantage of a better maneuverability; it can quickly change directions and thus it is more capable of tracking or following a detected subject.The multicopter will be equipped with an infrared camera, and use the computer vision algorithm developed by [17] for detection and tracking of subjects.

This thesis deals with path planning or mission planning for the search phase of a search and rescue mission. Different search patterns are found and their ability to cover the entire area of interest with the onboard camera, as well as their performances in different simulated scenarios, have been tested. These patterns have been implemented and integrated into the existing software and control structure used in this project. Finally, test-flights have been conducted using a hexacopter to test the implementation.

1.2 Previous Work

Research on searching and search planning was first introduced during World War II to assist with detection of enemy submarines with B.O Koopman as the leading expert on the field [16], [15]. Path planning for UAVs continues to be an important field of study for military purposes, such as minimizing the chance of the UAV being detected by an enemy radar [3], [8]. In [3], a Voroni graph with weighted edges is used to represent the possible routes, and the optimal path is found using dynamic programming. Other algorithms that have been used for UAV path planning includes A* [24], MILP [2], [9] and probability roadmaps [21].

The research of supporting search and rescue operations with UAVs is a growing

field, and path planning or search planning is an important part of this field. A search and rescue scenario with UAVs is found in [10], where two unmanned helicopters are used to cover a search area of 290m× 185m, and locate eleven subjects.

In [25], a mode-switching path planner for an UAV is developed to let a fixed-wing UAV fly side by side with a helicopter, and thereby increase the area coverage significantly. For Wilderness Search and Rescue (WiSAR), [18] proposes a path planner modelled as a discretized combinatorial optimization problem .

In [30], it is discussed how the use of heuristic search patterns can benefit a search and rescue mission, and how the use of these may improve the effectiveness of the search. Performance of different search patterns are tested in [7], by a simulation of German submarines in the Bay of Biscay 1942-1943.

1.3 Contribution and Scope of this Thesis

This thesis considers path planning for the UAV for the search part of a search and rescue mission. Different trajectories or search patterns are studied and how the on-board camera should be placed on the UAV to ensure the best possible area coverage is discussed. The search patterns' strong and weak sides are found through MATLAB-simulations.

The proposed patterns are implemented in the ground station and onboard software used in this project. The implemented scheme is tested by simulations using an ArduPilot software in the loop simulator, and finally a flight test is conducted.

1.4 Organization of this Thesis

This thesis is structured in the following manner. In Chapter 2, some background theory is explained. Chapter 3 details the various hardware components and software used for the hexacopter used in this thesis. Chapter 4 details a method for finding the area a multicopter covers with a single camera frame. This method is described for placing the camera on the multicopter with or without using a gimbal for roll and pitch stabilization. This method was implemented in the ground station software for live visualization of the area covered by the multicopter during a flight.

Chapter 5 studies different trajectories or search patterns. Algorithms for these are

created and presented. In Chapter 6, the optimal height for conducting a search is discussed. To find how large a human will appear in a picture, a model based on the theory from Chapter 4 is used.

Chapter 7 tests the different patterns performance in different scenarios by a series of simulations in MATLAB. For these simulations, a model of a multicopter with unlimited acceleration and angular rate is used.

Chapter 8 details the implementation of the search patterns as maneuvers in the onboard software Dune and the ground control station software Neptus. These maneuvers were tested using an ArduPilot software in the loop simulator.

Results from flight tests done at Agdenes airport, are shown in Chapter 9. The final conclusions, and some suggestions for future work are found in Chapter 10.

Chapter 2

Theory

2.1 Reference Frames

In this section, the coordinate frames used in this thesis and the implementation are described. This thesis follows definitions used in [13].

Earth-Centered Reference Frames

ECI: The Earth-centred inertial (ECI) frame $\{i\} = (x_i, y_i, z_i)$ is an inertial frame for terrestrial navigation, that is a non-accelerating reference frame in which Newton's law of motion apply. The frame's origin o_i is located in the center of the Earth.

ECEF: The Earth-centred Earth-fixed (ECEF) reference frame $\{e\} = (x_e, y_e, z_e)$ has its origin o_e in the center of the Earth. ECEF is rotating with respect to the ECI with an angular rate of rotation of $\omega_e = 7.2921 \times 10^{-5}$ rad/s.

Geodetic Frame

Geodetic frame: It is the reference system used by the GPS. A point in this frame is described by *latitude*, *logitude* and *height*, i.e (μ, l, h)

Geographic Reference Frames

NED: The North East Down (NED) $\{n\} = (x_n, y_n, z_n)$ frame is a local tangent plane used for local navigation. The x-axis is pointing to true North, the y-axis to the East and the z-axis points downwards normal to the Earth's surface.

NAV: The navigation frame (NAV), is in this thesis used for a rotated NED-frame. That is, a frame where the z-axis points downwards normal to the Earth's surface, but the x-axis and y-axis does not necessarily point towards true North and East. The angle between the x-axis and true North is called *bearing*, and denoted ψ.

Vehicle Coordinate Systems

BODY: The body-fixed frame $\{b\} = (x_b, y_b, z_b)$ with origin in the craft's *centre of origin* (CO) is a moving coordinate frame that is fixed to the craft. The x-axis is pointing in the craft's forward direction. The z-axis is pointing straight down relative to the vehicle and thus the y-axis points to the right.

CAM: The camera frame $\{c\} = (x_c, y_c, z_c)$ is a moving coordinate frame fixed to a camera. Using the ideal pinhole model, the origin of the frame is located inside the camera, with the z-axis pointing in the direction of the camera, and hits the center of the picture frame taken. The x-axis and y-axis points in the height and width direction of the picture taken respectively.

2.1.1 Vector Notation

The position of $\{b\}$ relative to the $\{n\}$ given in the $\{n\}$ is given by the vector $p^n_{b/n}$.

2.1.2 Rotation Matrices

To rotate from one frame of reference to another, rotation matrices are used. A rotation matrix between a and b is denoted $\mathbf{R}^{\mathbf{a}}_{\mathbf{b}}$, and it is an element in SO(3) or the *special orthogonal group of order 3.* A definition of SO(3) is found in [11]:

$$SO(3) := \{\mathbf{R} | \mathbf{R} \in R^{3\times 3}, \mathbf{R}^T\mathbf{R} = \mathbf{I} \text{ and } \det \mathbf{R} = 1\} \tag{2.1}$$

where $R^{3\times 3}$ is the set of all 3×3 matrices.

A rotation of a vector from one reference frame to another is expressed as

$$v^{to} = \mathbf{R}^{to}_{from} v^{from} \tag{2.2}$$

A rotation around a fixed axis is called a simple rotation. The rotation matrices around each axis are given by:

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.3}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.4}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Rotation between BODY and NED

When the vehicle's attitude is given in Euler angels: roll (ϕ), pitch (θ) and yaw (ψ), the rotation from BODY to NED is given by [11]:

$$\mathbf{R}^n_b := \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \tag{2.6}$$

When calculating this, $\mathbf{R}^n_b$ is found as:

$$\mathbf{R}^n_b = \begin{bmatrix} s(\theta)c\psi & -c(\phi)s(\psi) + s(\phi)s(\theta)c(\psi) & s(\phi)s(\psi) + c(\phi)s(\theta)c(\psi) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\theta)s(\psi) & -s(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c\theta) \end{bmatrix} \tag{2.7}$$

where $s\cdot = \sin(\cdot)$ and $c\cdot = \cos(\cdot)$.

2.2 Guidance, Navigation and Control

A motion control system can be constructed as three independent blocks denoted as the guidance, navigation and control (GNC) system. These systems interact with each other through data and signal transmissions [13]. This is illustrated by figure 2.1, where an autopilot for a multicopter is shown. Below, the tasks of each subsystems are listed, based on the definitions given by [13]:

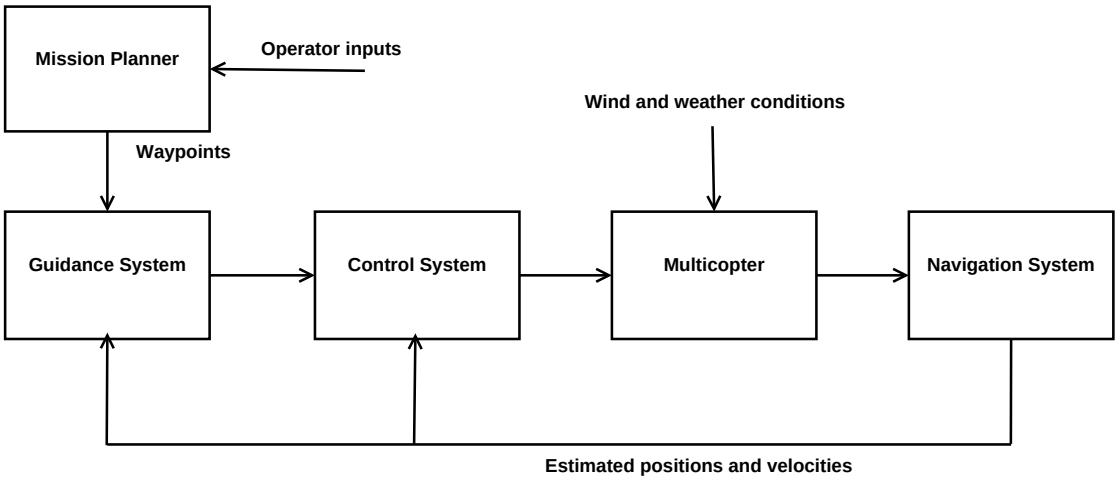


Figure 2.1: GNC signal flow. Based on figure 9.1 from [13]

Guidance is the action of the system that continuously computes the reference (desired) position, velocity and acceleration of the multicopter to be used by the motion control system.
Navigation is the science of directing a craft by determining its position/attitude, course and distance travelled. In some cases, velocity and acceleration are determined as well. This is usually done by using a global navigation satellite system (GNSS) combined with motion sensors such as accelerometers and gyros. An advanced navigation system is the inertial navigation system (INS).
Control is the action of determining the necessary control forces and moments to be provided by the craft in order to satisfy a certain control objective. The desired control objective is usually set by the guidance system, and examples of control objectives for a multicopter used in a search and rescue mission can be path-following, target tracking and setpoint regulation. The control laws can include both feedback, with outputs from the navigation system, and feedforward with signals from the guidance system.

2.2.1 Guidance Systems

This section contains some of the equations used for simulations and implementation later in this thesis. This theory is found in [13].

Lookahead-Based Stearing

For lookahead-based steering, the course angle assignment is separated into two parts:

$$\chi_d = \chi_p + \chi_r(e) \tag{2.8}$$

where $\chi_p = \alpha_k$ and e is the cross-track error given by the following equations:

$$\alpha_k = \arctan\left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right) \tag{2.9}$$

$$e(t) = -\{x(t) - x_k\}\sin(\alpha_k) + \{y(t) - y_k\}\cos(\alpha_k) \tag{2.10}$$

when the waypoints are denoted (x_k, y_k) and (x_{k+1}, y_{k+1})

For calculating $\chi_r(e)$, [13] offers some variants. The one used in this thesis is:

$$\chi_r = \arctan(-K_p e) \tag{2.11}$$

where K_p is a design variable given by

$$K_p(t) = \frac{1}{\Delta(t)} > 0 \tag{2.12}$$

and $\Delta(t)$ is found from:

$$\Delta(t) = \sqrt{R^2 - e(t)^2} \tag{2.13}$$

where R is chosen.

Head for next waypoint

When moving along a piece wise linear path made up of n straight-line segments connected by $n + 1$ waypoints, the decision to select the next waypoint can be made on the basis of whether or not the craft lies inside a *circle of acceptance* with radius R_{k+1} around the waypoint (x_{k+1}, y_{k+1}), i.e. if the following expression is true:

$$[x_{k+1} - x(t)]^2 + [y_{k+1} - y(t)]^2 \leq R_{k+1}^2 \tag{2.14}$$

Another criteria for switching waypoints can be the along-track distance, given by the following expression:

$$s(t) = [x(t) - x_k]\cos(\alpha_k) + [y(t) - y_k]\sin(\alpha_k) \tag{2.15}$$

where α_k is given by Equation 2.9.

The switch of waypoints should then occur if the multicopter has travelled the whole (or close to the whole) distance between the two waypoints, i.e. if the following expression is true:

$$s(t) + ds \geq \sqrt{(x_k - x_{k+1})^2 + (y_k - y_{k+1})^2} \tag{2.16}$$

where ds is a design variable indicating how close to the next waypoint the multicopter must be before a switch is requested.

When simulating, one of equations 2.14 and 2.16 can be used, or a combination of both. If the sampling time of the position is high, and the value of R_{k+1} low, using only Equation 2.14 may lead to none of the measured positions being within the circle of acceptance. This will not be experienced using Equation 2.16 as this criteria does not have an upper limit for the waypoint switch.

Chapter 3

System Description

This chapter presents an overview of the different parts of the system used in this thesis. This includes the multicopter, the payload, the software and the necessary communication protocols. A photograph of the multicopter used in this project is shown at the end of this chapter, in Figure 3.9.

3.1 The Multicopter

For this thesis, a hexacopter was chosen. The hexacopter is a ArduCopter 3DR Hexa B from 3DRobotics. It is delivered as a ready-to-fly kit featuring an ArduPilot Mega autopilot, fixed-pith propellers, 850kv brush-less motors, SimonK Electronic Speed Controllers (ESC), aluminium arms, fiberglass mounting boards, a power distribution board, GPS and a 3DR radio telemetry kit. The hexacopter is powered by a Hyperion 3s 4000mAh 25C battery. The hexacopter is shown in Figure 3.1.

In this project, the hexacopter is chosen to fly in X-configuration. The other alternative is +-configuration. These configurations are shown in Figure 3.2, where the blue and green arrows indicate rotor configuration. In X- configuration, the hexacopter flies with two arms first, and the heading of the craft is in the middle of those two arms, while in +-configuration, one arm flies first, and indicates the heading of the hexacopter. The numbers on the motors in the drawing indicate which output port on the autopilot the motor should be connected to. The configuration is normally shown by using a separate color on the arms flying first; thus it can be seen that the hexacopter in Figure 3.1 is in X-configuration.

Figure 3.1: ArduCopter 3DR Hexa B. *Image courtesy 3drobotics.com*

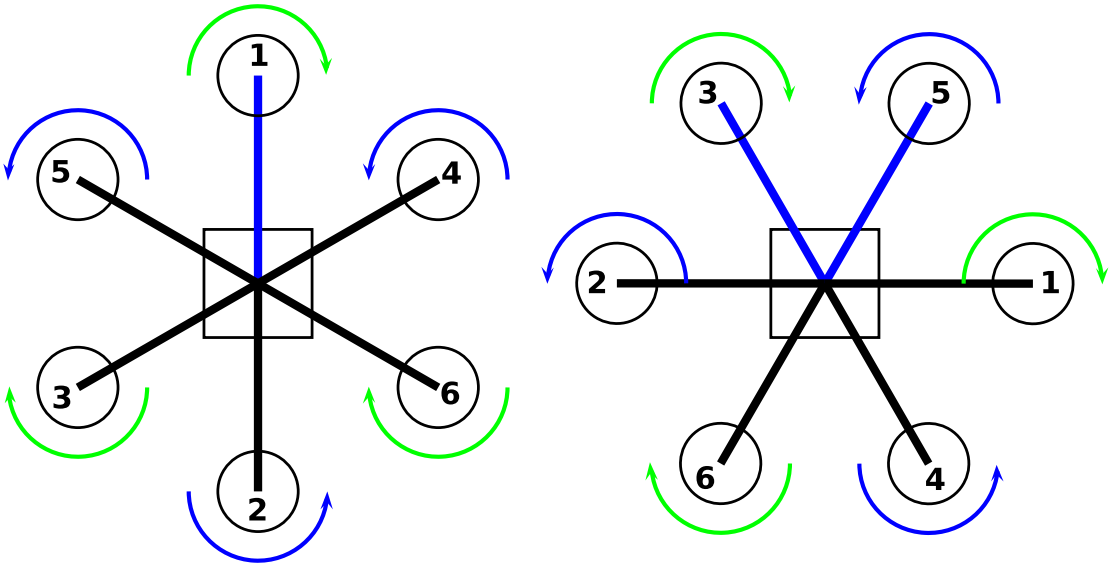


Figure 3.2: The two possible hexacopter configurations, +-configuration to the left and X-configuration to the right. Green arrows indicate clockwise direction of the rotors, while blue arrows indicate counter-clockwise direction.

3.1.1 Autopilot

The ArduPilot Mega 2.6 is a complete open source autopilot system capable of controlling angular rotations and altitude and performing programmed GPS missions with waypoints. It has an Atmel ATMEGA2560 chip for processing, and an Atmel ATMEGA32U-2 chip for USB-functions.

The ArduPilot includes an Inertial Measurement Unit (IMU) with the following sensors:

- InvenSense MPU-6000, 3-axis Gyro / 3-axis Accelerometer
- Honeywell HMC5883L-TR 3-axis Digital Compass
- Measurement Specialities MS5611-01BA03 Barometric Pressure Sensor

Figure 3.3: ArduPilot Mega 2.6. *Image courtesy 3drobotics.com*

3.1.2 Radio Controller

For safety and other practical reasons a radio controller is used to manually arm and disarm motors and to do the take-off and landing. The radio controller chosen is a Spektrum DX7S Radio Controller,which is shown in Figure 3.4. It features 7 channels and a 2.4GHz DSm spread spectrum telemetry system.

When flying, mainly two flightmodes on the controller are used, *Stabilize* and *Loiter*. When in Stabilize, the operator is in full control of the hexacopter, and controls the hexacopter with the sticks. When in Loiter mode, the hexacopter

loiters; now Dune has control of the hexacopter, and waypoints and plans can be sent. The operator can at any time switch between these modes using a switch on the controller, and abort any plan running in Dune.

Figure 3.4: Spektrum DX7S Radio Controller. *Image courtesy spektrumrc.com*

3.2 Payload

This section describes the payload proposed for a search and rescue mission. The payload is designed by [17]. A drawing of how the payload is connected with the rest of the system is shown in Figure 3.5. For powering the payload, a LiPo 11.1V 2200mAh battery is used.

For a hexacopter low weight is one of the highest priorities of the payload, as the weight limits the flight time. The following list contains the components in a prioritized order, these should be added one by one until the maximum weight of the payload for the hexacopter is reached.

- Battery
- Single Board Computer

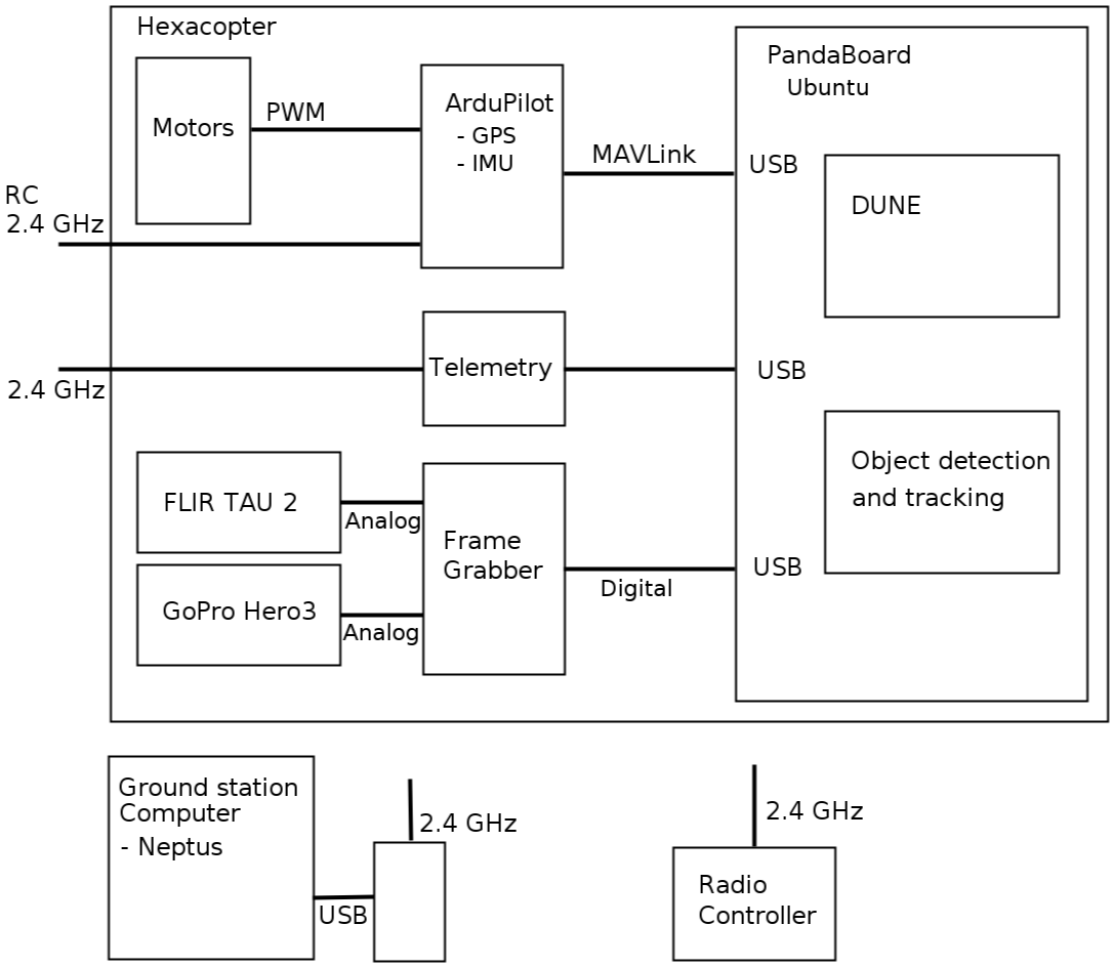


Figure 3.5: System drawing

- Infrared Camera
- Frame Grabber
- Video Camera

3.2.1 Single Board Computer

The PandaBoard ES is a low-power, low-cost single-board computer development platform. It is based on the Texas Instruments OMAP4430 SoC (System on Chip). The PandaBoard ES is shown in Figure 3.8. Some of its most important properties are listed in Table 3.1. [17]

Table 3.1: Properties PandaBoard ES

Overview	**PandaBoard ES**
CPU	1.2GHz Dual Core
Processor Type	ARM Cortext-A9
RAM	1GB - GPU
GPU	304 MHz Power VR SGX540
Physical Attributes	
Size	11.4cm × 10.1cm × 3cm
Weight	82g
Interfacing	
Display Port	HDMI
Power	DC Jack / USB OTG - 5V 1A
Storage	SD Card - Ethernet WiFi
Ethernet	RJ45 and WiFi
Environment	
Operating Temperature	-20° - 70°
Humidity	N/A

3.2.2 Infrared Camera

The infrared camera chosen for this project is a FLIR Tau 2 336 which can be seen in Figure 3.6. A lens with focal length 9mm was chosen by [17], which gives a field of view of 35° × 27° with a frame rate of 8.3 Hz. The camera's weight is 72g.

Figure 3.6: FLIR Tau 2 336 Infrared Camera. *Image courtesy flir.com*

Figure 3.7: GoPro Hero3. *Image courtesy allthingsd.com*

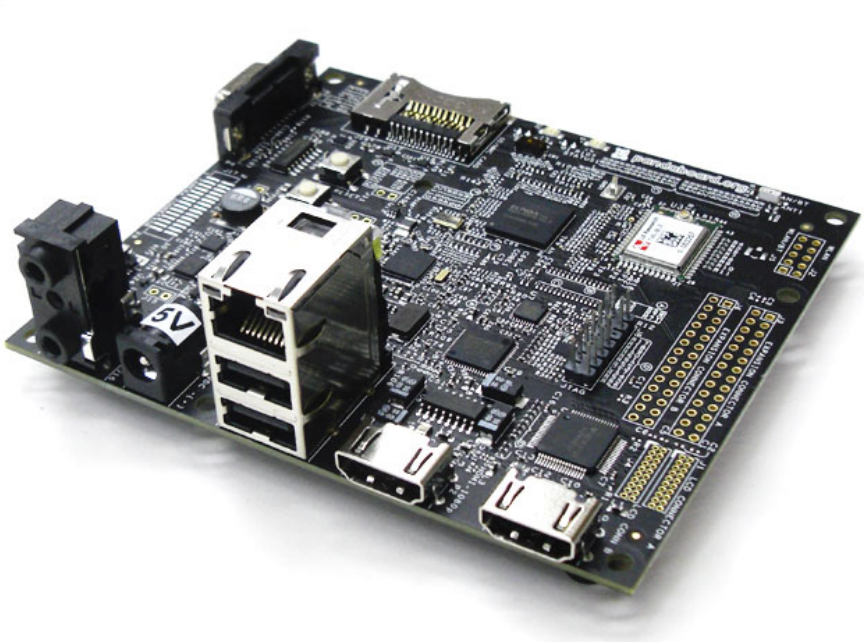


Figure 3.8: Pandaboard ES. *Image courtesy blog.digit-parts.com*

3.2.3 Gimbal

For placing the camera, the gimbal chosen was a Tarot T-2D. It is used for roll and pitch stabilization of the infrared camera such that the best possible quality of the pictures is achieved.

3.2.4 Frame Grabber

A frame grabber is an electronic device used to capture individual, digital still frames from an analog video signal, such as the output from the IR-camera. The frame grabber chosen by [17], is an EasyCap DC60. The frame grabber connects to the infrared camera by a RCA-connector, and to the Pandaboard by USB. It has a weight of 57g.

3.2.5 Video Camera

For video camera, a GoPro Hero3 was chosen. It is a compact and solid camera often used in similar applications. The weight of the camera is 74g. It is powered

by an internal battery, making it very easy to use. The GoPro Hero3 is shown in Figure 3.7.

3.3 Software

3.3.1 Ubuntu

Ubuntu is an open source operating system based on the Linux kernel. It is one of the most popular Linux distributions, and therefore it has the advantage of being frequently updated and having support for many different USB-components such as frame grabbers and different GPS devices. Ubuntu is also widely used for platforms such as the PandaBoard, resulting in a variety of distributions that are optimized with respect to the ARM architecture. All of these factors make Ubuntu an ideal operation system for the PandaBoard [17].

3.3.2 Computer Vision

For detection and tracking of objects with the infrared camera, an algorithm developed by [17] will be used. The object detection algorithm utilizes two types of pre-trained classifiers to perform detection. The object tracking algorithm is based on an estimate-and-measure tracking approach. A standard Kalman filter is used for estimation. This algorithm is implemented using OpenCV.

OpenCV

Intel Open Computer Vision Library, referred to as OpenCV, is a function library mainly aimed at real-time computer vision. The library is open-source and cross-platform, and receives frequent updates from its large user base.

3.3.3 DUNE

DUNE is short for Unified Navigational Environment. It is developed by the *Underwater Systems and Technology Laboratory* (LSTS), located in Porto. DUNE is the onboard software running on the vehicle which is responsible for interaction with sensors, payload and actuators, and also for communications, navigation, control, maneuvering, plan execution and vehicle supervision. It is CPU architecture

independent as well as operating system independent, and can be used in ASVs, ROVs, AUVs and UAVs [22].

3.3.4 Neptus

Neptus is a ground station software developed by the creators of Dune. Neptus can be used during all the stages of the mission. Before the mission, Neptus is used to create a mission plan. The plan can then be validated through simulations, and edited.

During a mission, Neptus can be used to receive and visualize information from one or many unmanned vehicles. Commands such as waypoints and new mission plans can also be send to the craft. After mission, Neptus can be used to review and analyse the mission. Neptus includes a Mission Review and Analysis tool that can be used to closer examine data collected during the mission [22].

3.4 Communication Protocols

3.4.1 IMC

Dune and Neptus use the Inter-Module Communication (IMC) protocol for communication. IMC defines a common message set that is understood by all network nodes, Dune tasks and Neptus plugins. IMC is fully defined and documented in a single XML file, which using by XSLT can be translated into different language bindings [22]. IMC is used for communication between Dune and Neptus, and between different Dune-tasks.

3.4.2 MAVLink

MAVLink or Micro Air Vehicle is a communication protocol for communicating with small unmanned vehicles [23]. Normally, it is used for communication between an ArduPilot Mega, and the Ground Control Station. In this project however, MAVLink is used for communication between Dune-tasks, onboard the multicopter, and the ArduPilot Mega.

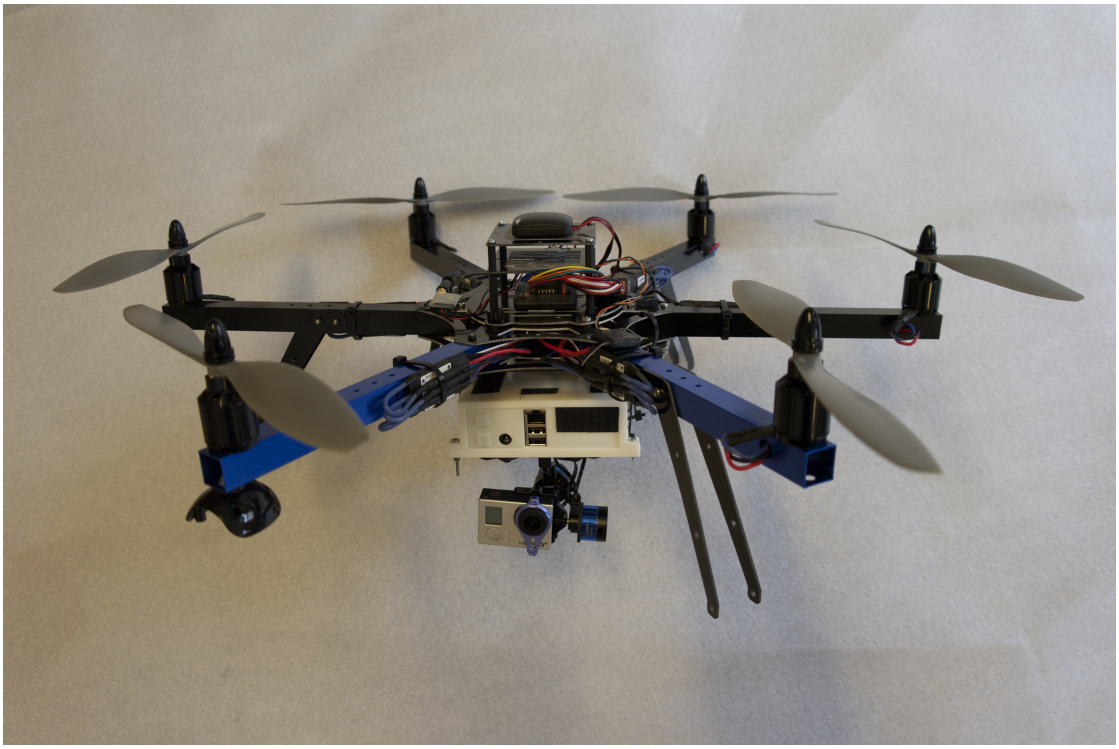


Figure 3.9: Photograph of the hexacopter. Here, the gimbal is used to stabilize the GoPro Hero3, as the infrared camera is not used. The Pandaboard is located within the white box hanging under the hexacopter. *Photo by Thor Audun Steen*

Chapter 4

The Onboard Camera's Field of View

4.1 Camera Placement

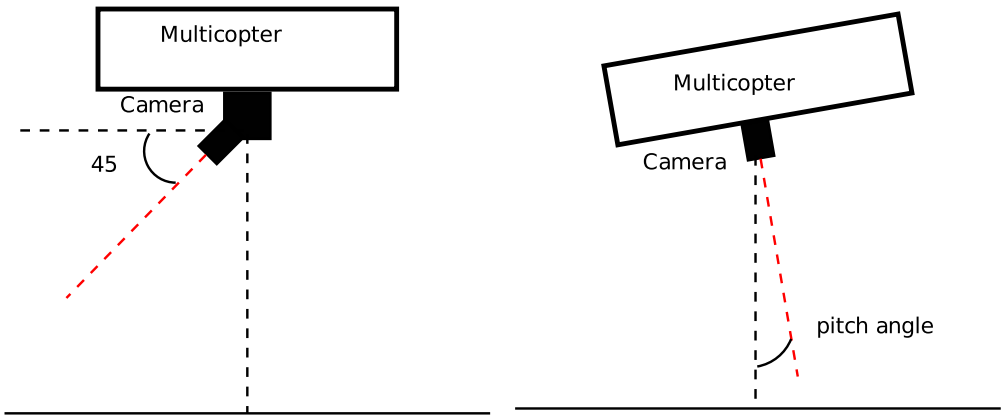
In this thesis, two methods for placing the camera are considered. These are shown in Figure 4.1. The first approach uses a gimbal for roll and pitch stabilization. The gimbal is set to point in a predefined direction of $\alpha = 45°$ forward. The camera will then cover an area in front of the multicopter.

The other approach is a strapdown solution without a gimbal. The camera will then point straight down when the multicopter has no roll or pitch. When moving, the roll and pitch will affect which area is covered.

4.2 Projected Camera Image

To see which area that is covered by the camera, the corner coordinates of the projected camera image frame relative earth must be found. Assuming approximately flat ground, a method for this is found in [27]. This method, with some minor adjustments and simplifications, is detailed in this section.

For the following calculations, what is known as the ideal pinhole camera model is used. The model is ideal as it is completely undisturbed, and without any distortions. A projected image plane is created between the ground and the camera.



(a) With gimbal (b) Without gimbal

Figure 4.1: Methods for placing the camera on the multicopter.

The center of this image is given by:

$$C^c_{proj} = [0, 0, -\frac{1}{2}z]^T \tag{4.1}$$

which is half of the UAV's altitude. The projected image plane is shown in Figure 4.2 as the blue plane.

The coordinates of the projected image's corners can then be calculated using the angles for field of view from the camera's datasheet; α_h and α_w. These angles represent the whole field of view, and must therefore be halved when calculating the coordinates of the corners; $H = \alpha_h/2$, $W = \alpha_w/2$. The coordinates are denoted by the body-fixed names Front/Rear and Starboard/Port.

$$FS^c_{proj} = \left(|C^c_{proj}| \tan(H), |C^c_{proj}| \tan(W), -|C^c_{proj}|\right) \tag{4.2}$$

$$FP^c_{proj} = \left(|C^c_{proj}| \tan(H), -|C^c_{proj}| \tan(W), -|C^c_{proj}|\right) \tag{4.3}$$

$$RS^c_{proj} = \left(-|C^c_{proj}| \tan(H), |C^c_{proj}| \tan(W), -|C^c_{proj}|\right) \tag{4.4}$$

$$RP^c_{proj} = \left(-|C^c_{proj}| \tan(H), -|C^c_{proj}| \tan(W), -|C^c_{proj}|\right) \tag{4.5}$$

The transformation from BODY-frame (and CAM-frame) to NED-frame is given by a series of rotations, and the transformation matrix $\mathbf{M}$ is given by:

$$\mathbf{M} := \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi)\mathbf{R}_y(\alpha) \tag{4.6}$$

where the angles $[\phi, \theta, \psi]^T$ are the UAVs attitude. When using the gimbal with roll and pitch stabilization, $\mathbf{M}$ is reduced to:

$$\mathbf{M} = \mathbf{R}_z(\psi)\mathbf{R}_y(\alpha) \tag{4.7}$$

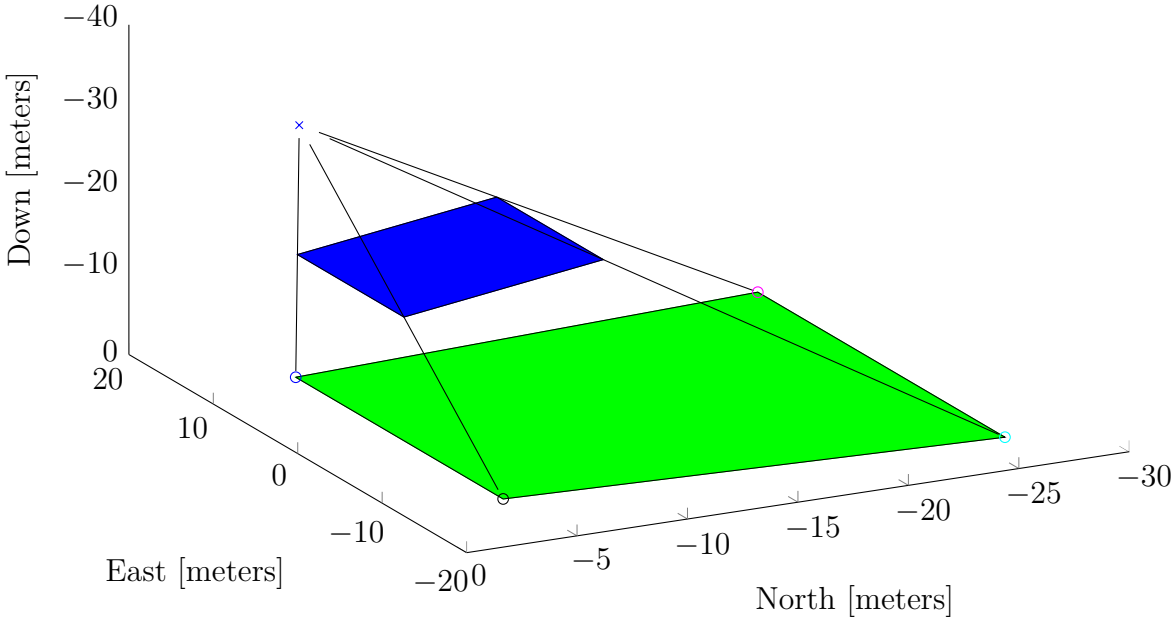


Figure 4.2: The area covered by the camera. The multicopter is placed in $[0, 0, -40]^T$, and has a pitch angle of 20°.

Without the gimbal, $\mathbf{M}$ equals the the rotation matrix from NED to BODY.

$$\mathbf{M} = \mathbf{R}_b^n = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi) \tag{4.8}$$

The simple rotations $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ are found in Equations 2.3 to 2.5, and $\mathbf{R}_b^n$ is found in Equation 2.7.

Then, the corners, and center, of the projected image in NED-coordinates are given by the following equations:

$$C_{proj}^n = \mathbf{M}\left(C_{proj}^c\right) + r_{c/n}^n \tag{4.9}$$
$$FS_{proj}^n = \mathbf{M}\left(FS_{proj}^c\right) + r_{c/n}^n \tag{4.10}$$
$$FP_{proj}^n = \mathbf{M}\left(FP_{proj}^c\right) + r_{c/n}^n \tag{4.11}$$
$$RS_{proj}^n = \mathbf{M}\left(RS_{proj}^c\right) + r_{c/n}^n \tag{4.12}$$
$$RP_{proj}^n = \mathbf{M}\left(RP_{proj}^c\right) + r_{c/n}^n \tag{4.13}$$

with $r_{c/n}^n$ as the position of the camera's center of orientation in NED-coordinates. This is found from:

$$r_{c/n}^n = r_{b/n}^n + \mathbf{R}_b^n r_{c/b}^b \tag{4.14}$$

where $r^n_{b/n}$ is the UAV's position in NED. For the setups described in this chapter, the camera is placed at the bottom of the multicopter as seen in Figure 4.1. Then the camera's position in the BODY-frame is given by:

$$r^b_{c/b} = [0, 0, d]^T \tag{4.15}$$

where d is the measured distance from the multicopter's center of orientation (CO) to the center of the camera's frame.

With this, the projected images can be found in the NED-frame, as shown in Figure 4.2, and the next task is to calculate the lines through the projected pyramid and down to the ground. Since two points on this line is known, a parametric equation can be made. This is done by first finding the slopes in each axis;

$$\Delta(C^n_{proj})_x = (C^n_{proj})_x - (r^n_{c/n})_x \tag{4.16}$$
$$\Delta(C^n_{proj})_y = (C^n_{proj})_y - (r^n_{c/n})_y \tag{4.17}$$
$$\Delta(C^n_{proj})_z = (C^n_{proj})_z - (r^n_{c/n})_z \tag{4.18}$$

Then, the parametric equation for the line through the center of the projected pyramid starting in the camera's center is given by:

$$C(t) = \begin{pmatrix} (r^n_{c/n})_x + \Delta(C^n_{proj})_x \cdot t \\ (r^n_{c/n})_y + \Delta(C^n_{proj})_y \cdot t \\ (r^n_{c/n})_z + \Delta(C^n_{proj})_z \cdot t \end{pmatrix} \tag{4.19}$$

On ground, the z-component of $C(t)$ equals zero, and thus, t can be found from:

$$t = \frac{-(r^n_{c/n})_z}{\Delta(C^n_{proj})_z} \tag{4.20}$$

Then, the center coordinates on ground is given by:

$$C = \begin{pmatrix} (r^n_{c/n})_x + \frac{-(r^n_{c/n})_z \Delta(C^n_{proj})_x}{(C^n_{proj})_z} \\ (r^n_{c/n})_y + \frac{-(r^n_{c/n})_z \Delta(C^n_{proj})_y}{(C^n_{proj})_z} \\ 0 \end{pmatrix} \tag{4.21}$$

Then, the same steps are used to find the four corners (FS,FP,RS and RP). All of the corners will then be available in the NED-plane, and can be seen forming the green shape in Figure 4.2. These can then be converted to ellipsoidal coordinates using the method described in Appendix A.2.

4.3 Implementation

A Neptus plugin called *Picture outline on map* using the algorithm for finding the picture corner points was developed. This plugin listens for new *IMC::EstimatedState*-messages, from which it gets the multicopter's position and attitude.

Using the algorithm explained in the previous section, the corner points of a picture are found in the NED-frame relative to the position of the multicopter, and transformed to ellipsoidal coordinates using the method in Appendix A.2. Using an existing Neptus-function, these points, in ellipsoidal coordinates, can be translated into screen positions and drawn on the screen map.

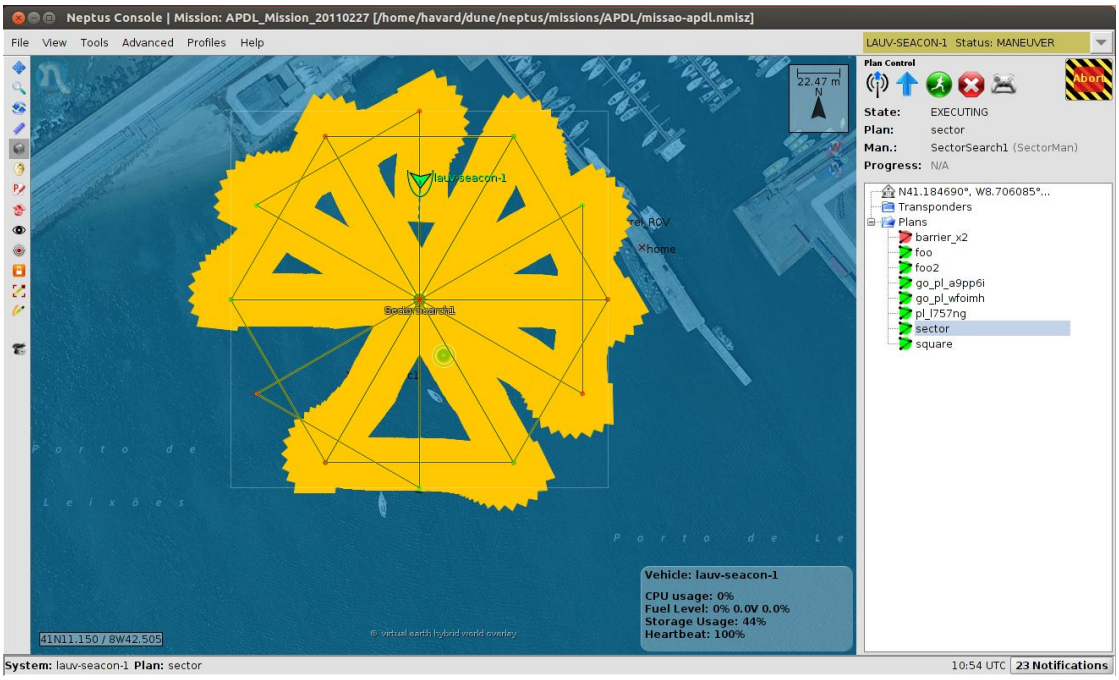


Figure 4.3: A screenshot from Neptus when simulating with the *Picture outline on map* plugin active.

Chapter 5

Path Planning

5.1 Search and Rescue

A definition of Search and Rescue (SAR) is given by [30]: *Search and Rescue is the act of searching for, rescuing, or recovering by means of ground, marine, or air activity any person who becomes lost, injured, or is killed while outdoors or as a result of a natural or man-made disaster.* The US Coast Guard defines search and rescue as the use of available resources to assist persons and property in potential or actual distress [14].

Search and rescue can be broken down into two defining disciplines: search and rescue. Locating a missing subject or object is the first element of any SAR mission and must take place before a rescue can occur. The second phase, rescue, takes place once the subject of the search is located. This consists of first accessing the subject, providing initial care to prevent further distress and finally returning the subject to a more stable environment [12].

5.2 Search Patterns

In this section, five search patterns from the US National Search and Rescue Manual [26] are described. Detailed descriptions can also be found in [7]. The starting point of each pattern is denoted CSP (commence search pattern), and the probable location of the subject is called datum.

5.2.1 Parallel and Creeping Line Search Patterns

When the datum is not known with a high degree of certainty and the search area is large, either the parallel (Figure 5.1) or the creeping line search pattern (Figure 5.2) is preferable. When the missing subject is equally likely to occupy any part of the search area, the parallel pattern is desirable, while the creeping line is the best choice if the subject is more likely to be in one end of the search than the other. The dimensions of the search area can also be the tipping point for deciding which of these patterns to use.

When the search area is quadratic, the search patterns will be equal, but rotated 90°. When the search area in NED is longer in East than in North, the parallel search pattern will have a slightly longer total distance than the creeping line, but it will have fewer and longer lines, and thus better navigational properties. Similar, an increase in North will give the same advantages for the creeping line search pattern.

These patterns have the best navigational properties as they have long straight lines to follow, and few turns. Long straight lines also means it is possible for the multicopter to travel with higher speed between each turn, and thus the flight time can be lowered. The parallel search pattern can be created using Algorithm 5.1 and the creeping line search pattern using Algorithm 5.2.

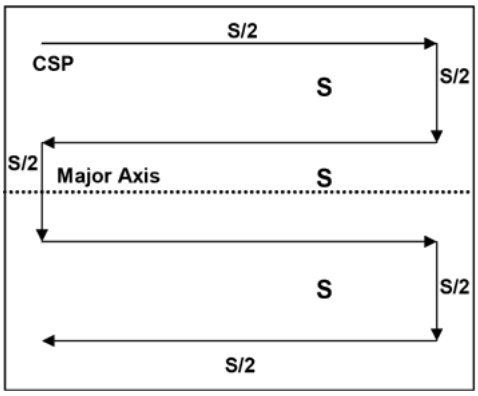


Figure 5.1: The parallel search pattern. Picture source [7]

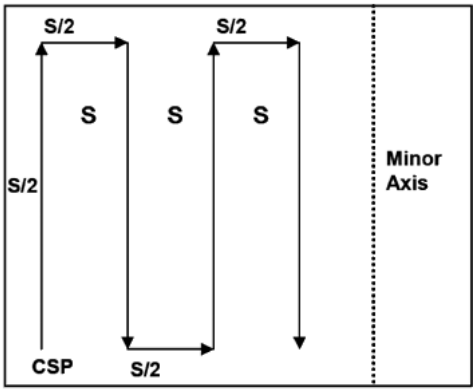


Figure 5.2: The creeping line search pattern. Picture source [7]

5.2.2 Square and Sector Search Patterns

When the datum is well known or established within close limits, the square (Figure 5.3) and sector (Figure 5.4) search patterns are preferable. Then, the datum and

Algorithm 5.1 The parallel search pattern

```
i=1, px = CSP(1), py=CSP(2), h=height
while px - S + (S+1)/2 < CSP(1) + distanceNorth do
  Waypoints(i) = [px,py,h]
  if mod(i,2) == 0 then
    px = px + S
  else if mod(i-1,4) == 0 then
    py = CSP(2) + distanceEast - S/2
  else
    py = CSP(2)
  end if
  i = i +1
end while
```

Algorithm 5.2 The creeping line search pattern

```
i=1, px = CSP(1), py=CSP(2), h=height
while py - S + (S+1)/2 < CSP(2) + distanceEast do
  Waypoints(i) = [px,py,h]
  if mod(i,2) == 0 then
    py = py + S
  else if mod(i-1,4) == 0 then
    px = CSP(1) + distanceNorth - S/2
  else
    px = CSP(1)
  end if
  i = i +1
end while
```

the CSP is the same point. The square pattern is used when uniform coverage of the search area is desired, while the sector search is used in scenarios where the target is difficult to detect having the advantage that it crosses the CSP several times from different directions.

The square search pattern consists of straight lines and turns. Every other search line is increased by one track space S. All turns are 90 degrees to the right. A possible algorithm for the square search is found in Algorithm 5.3.

The sector search pattern is created within a circle. When the path hits the circle, a turn 120 degrees to the right is made. After three sectors have been created, the path returns to the CSP point in the center. Then, the same pattern is created one more time with a 30 degrees offset relative to the previous pattern. In Algorithm 5.4, a possible algorithm for the sector search pattern is proposed. This algorithm creates a path that loops through the three sectors k_{max} times, where k_{max} is defined by the operator.

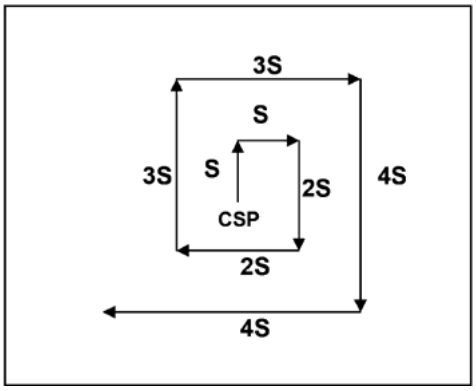


Figure 5.3: The square search pattern. Picture source [7]

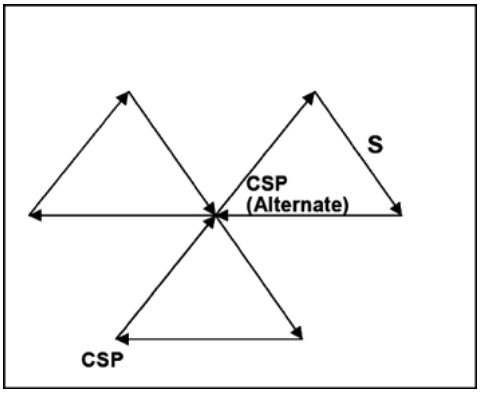


Figure 5.4: The sector search pattern. Picture source [7]

5.2.3 Barrier Patrol Search Pattern

The barrier patrol search pattern is preferable when the subject is fast moving, or for instance in sea with a strong water current. The barrier patrol search pattern consists of 12 waypoints with a fixed distance from the CSP. These waypoints can be created from Table 5.1.

Algorithm 5.3 The square search pattern

```
i = a = 1, px = CSP(1), py=CSP(2), h=height
while a · S < distance + S do
  Waypoints(i) = [px,py,h]
  if mod(i-1,4) == 0 then
    px = px + a · S
  else if mod(i-2,4) == 0 then
    py = py + a · S
  else if mod(i-3,4) == 0 then
    px = px - a · S
  else
    py = py - a · S
  end if
  if mod(i,2) == 0 then
    a = a+1
  end if
  i = i +1
end while
```

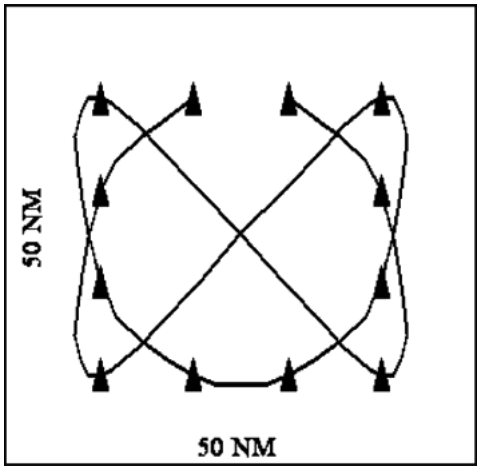


Figure 5.5: The barrier patrol search pattern. Picture source [7]

Algorithm 5.4 The sector search pattern

i=1, j=0, px = CSP(1), py=CSP(2), h=height
Waypoints(i) = [px,py,h]
d = length/2
while j < k_{max} **do**
  if i == 1 **then**
    px = CSP(1) + d cos(30 + $j \cdot 30$)
    py = CSP(2) + d sin(30 + $j \cdot 30$)
  else if i == 2 **then**
    px = CSP(1) + d cos(90 + $j \cdot 30$)
    py = CSP(2) + d sin(90 + $j \cdot 30$)
  else if i == 3 **then**
    px = CSP(1) + d cos(270 + $j \cdot 30$)
    py = CSP(2) + d sin(270 + $j \cdot 30$)
  else if i == 4 **then**
    px = CSP(1) + d cos(330 + $j \cdot 30$)
    py = CSP(2) + d sin(330 + $j \cdot 30$)
  else if i == 5 **then**
    px = CSP(1) + d cos(150 + $j \cdot 30$)
    py = CSP(2) + d sin(150 + $j \cdot 30$)
  else if i == 6 **then**
    px = CSP(1) + d cos(210 + $j \cdot 30$)
    py = CSP(2) + d sin(210 + $j \cdot 30$)
  else if i == 7 **then**
    px = CSP(1)
    py = CSP(2)
    j = j+1
    i = 0
  end if
  i = i +1
  Waypoints(i) = [px,py,h]
end while

Table 5.1: Waypoints for the barrier patrol search pattern

Waypoint	x-coordinate	y-coordinate
1	CSP(1) + d/2	CSP(2) + d/6
2	CSP(1) + d/6	CSP(2) + d/2
3	CSP(1) - d/2	CSP(2) + d/2
4	CSP(1) + d/2	CSP(2) - d/2
5	CSP(1) - d/6	CSP(2) - d/2
6	CSP(1) - d/2	CSP(2) - d/6
7	CSP(1) - d/2	CSP(2) + d/6
8	CSP(1) - d/6	CSP(2) + d/2
9	CSP(1) + d/2	CSP(2) + d/2
10	CSP(1) - d/2	CSP(2) - d/2
11	CSP(1) + d/6	CSP(2) - d/2
12	CSP(1) + d/2	CSP(2) + d/6

5.3 Cover Area

This section outlines a method for finding the optimal lawnmower pattern that covers a non-rectangular area, as the area shown in Figure 5.6. As the multicopter has limited flight range, the optimal path will be the shortest path that covers the area. This approach is based on the work presented in [20], with some simplifications and adjustments. The method outputs waypoints in the NED-frame, and these can be translated into GPS-positions using the method outlined in Appendix A.1. Additional examples of paths created by this method is found in Appendix B.

First, a bounding box is created around the shape, by getting the minimum and maximum height and width. The bounding box is shown in red in Figure 5.7. It is assumed that the multicopter starting position is in $[0, 0]^T$, and that the first waypoint is the closest one to this point.

The number of rounds required to cover the shape is given by

$$n = \frac{W_{shape}}{2 \cdot W_{camera}} \tag{5.1}$$

where W_{shape} is the width of the shape, and W_{camera} is the area covered on the side of the craft from a given altitude, given by Equation 5.2, where α_w is the sideways field of view of the camera, found in the camera's data sheet. n must

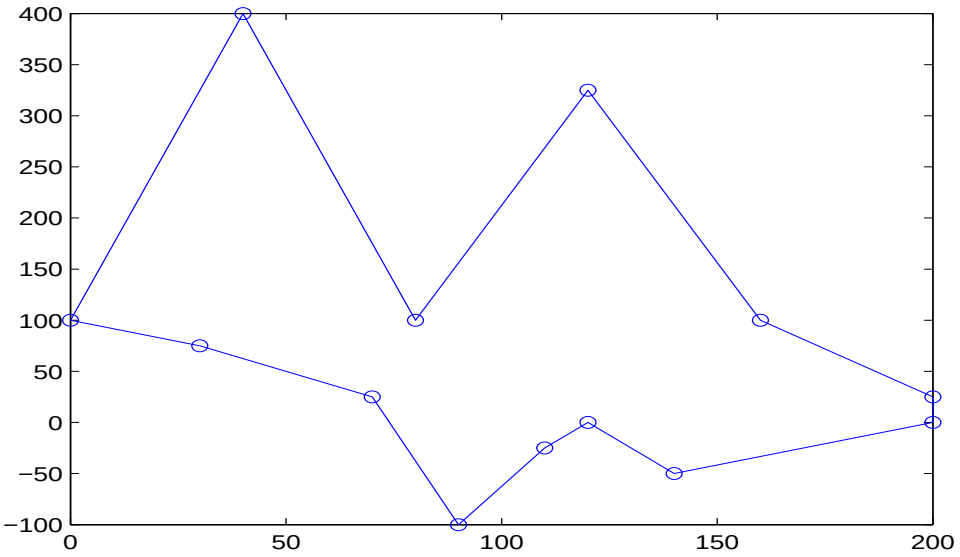


Figure 5.6: The area to be searched

then be rounded up to the nearest integer.

$$W_{camera} = altitude \cdot \tan\left(\frac{\alpha_w}{2}\right) \tag{5.2}$$

The distance between each line is given by

$$W_{round} = \frac{W_{shape}}{n} \tag{5.3}$$

Since n is rounded up; $W_{round} < 2 \cdot W_{camera}$, and some overlap between each camera frame is achieved.

The first two waypoints are then placed on a line $\frac{1}{2} \cdot W_{round}$ from the West edge of the bounding box, and the rest are placed evenly spaced W_{round} from each other. This creates the search pattern shown in green in Figure 5.7. This pattern provides full coverage of the shape. The distance however is much longer than necessary, and as can be seen in the figure, large areas outside the shape are covered.

To reduce the search distance, post processing can be made. The first task is to cut the waypoint lines down to the shape. This is shown in Figure 5.8. Then, a much shorter path is found, but this does not cover the entire area.

To increase the coverage, more processing is made. Between each waypoint-line, a virtual line is placed inside the area. These lines are shown in Figure 5.8. If the

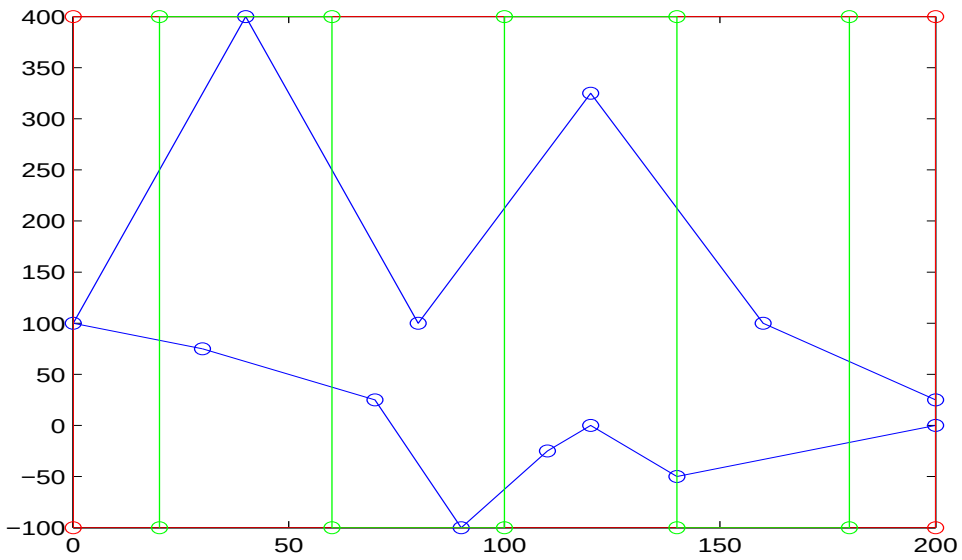


Figure 5.7: The area to be searched, with the bounded box and a lawnmower pattern.

path given by the waypoints does not cover the entire virtual line, the waypoints are adjusted. The changes are shown in Figure 5.9. Here, it can easily be seen that a larger part of the shape is covered, but is the whole shape covered?

This can be tested by drawing a circle with radius W_{camera} around each corner of the shape. This is done in Figure 5.10. If the path intersects with this circle, the area is covered by the search. Otherwise the area is not covered, and the path must then be further processed to complete the coverage.

For such uncovered areas, [20] proposes an algorithm that computes the distance from each waypoint line to the vertex, and inserts a new waypoint between the waypoints that makes up the waypoint line with the shortest distance to the vertex.

Implementation of this proved to be a bad solution, with new waypoints making straight lines crooked, and thus losing area coverage. Therefore, another approach had to be found.

The solution was to simply prolong the closest waypoint line (or lines) to the vertex not covered, similarly to what was done in the previous post-processing. This approach also has the advantage that it does not increase the number of turns, and it keeps the form of the search with long straight lines North-South intact. Figure 5.11 shows the path after the final step of post-processing. Here, it

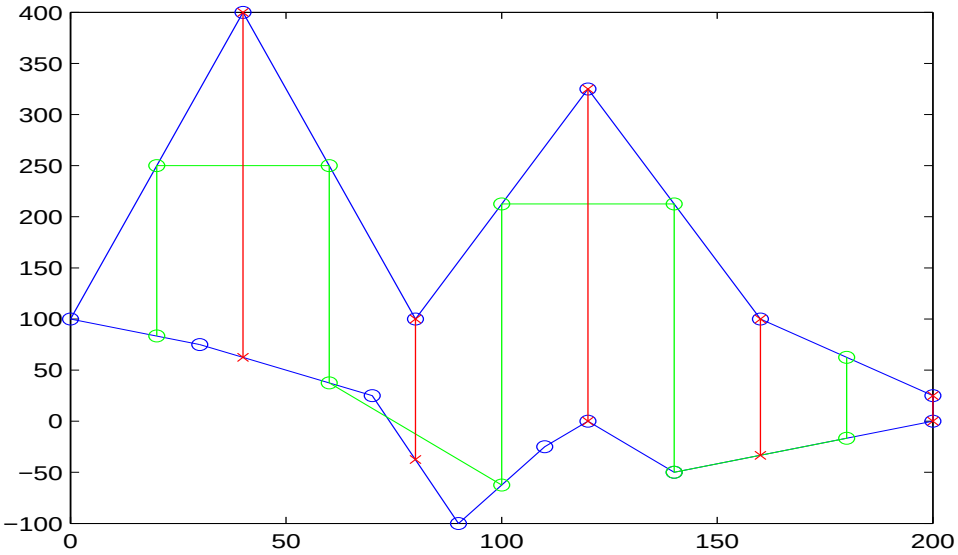


Figure 5.8: The path after the first step of processing. The virtual lines are shown in red.

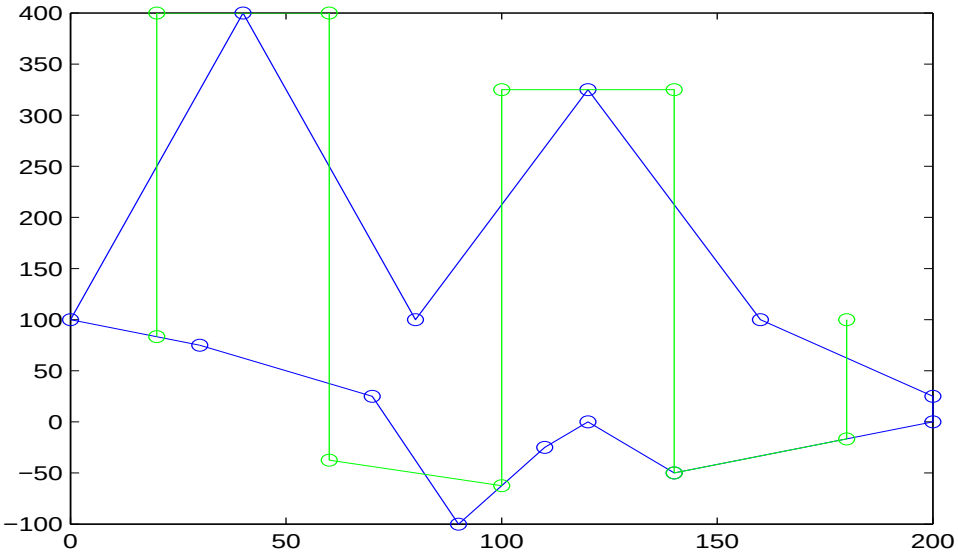


Figure 5.9: The path after the second step of processing.

can be seen that the waypoint previously located in $[-62.5, 100]^T$ is now moved to $[-100, 100]^T$, and thus the path will cover the entire shape.

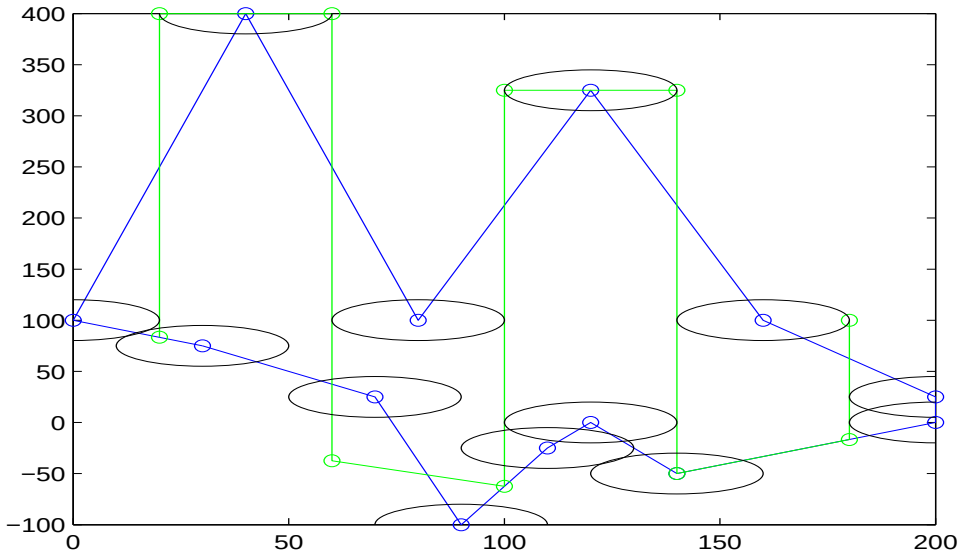


Figure 5.10: The path after the second step of processing. The circles around the corners of the shape show that a part of the shape, located at the bottom of the plot, is not covered. Note that the circles look ellipsoidal due to different scale on the axes of the plot.

5.3.1 Reduce Search Distance

When searching with a multicopter, or other crafts with limited flight time, it is always desirable to reduce the distance needed to complete the search. When using other crafts such as air-planes, it might also be desirable to reduce the number of turns. With a hexacopter, this is however of less importance, and therefore it is chosen to focus only on reducing the distance needed to complete the search. In [20], two possibilities are mentioned which are described in the following sections.

Picture height

The height of a picture is given by Equation 5.4.

$$H_{camera} = altitude \cdot \tan\left(\frac{\alpha_h}{2}\right) \tag{5.4}$$

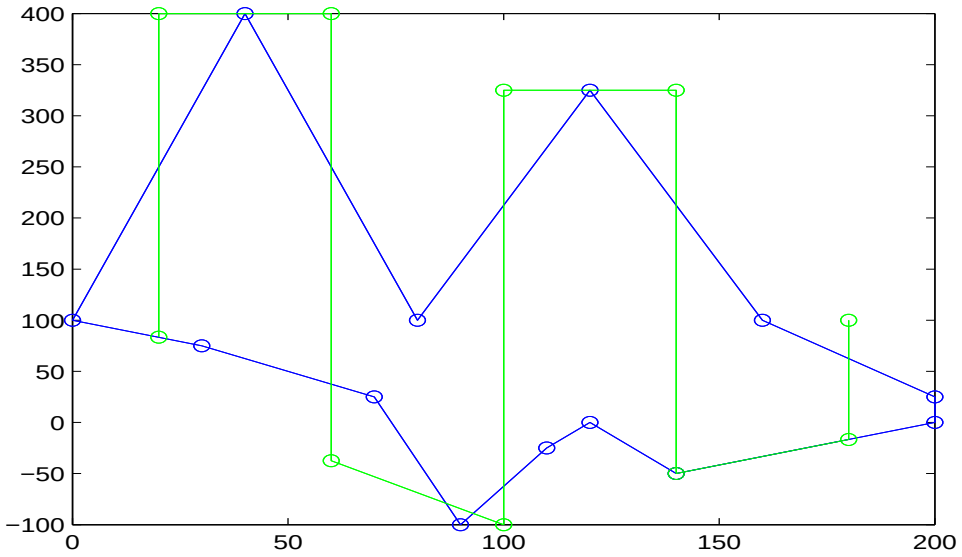


Figure 5.11: The path after the third and final step of processing. This path should cover the entire shape.

where α_h is the field of view in height direction found in the camera's data sheet.

Depending on the camera's position, the route can be reduced by shortening all the waypoint lines by H_{camera}. A reduced route is shown in Figure 5.12. The path shown here is 200 meters shorter than the path shown in Figure 5.11.

Rotation

The path can be shortened by trying different rotations. For this, Algorithm 5.5 can be used. This was applied to the path shown in Figure 5.12. The path with the shortest distance, 1794 meters, was found with a rotation of $\theta = 175°$, and is shown in Figure 5.13 The path with the longest search distance, 2324 meters, was found with a rotation of $\theta = 153°$, and can be seen in Figure 5.14. One might also note that the path with the longest distance has 12 waypoints while the shortest path has only 10, i.e the path with the shortest search distance has less turns than the path with the longest search distance. Few turns is also a desirable property.

How the search length varies with different rotations is shown in Figure 5.15. This shows that the search distance varies substantially for small changes in θ, and for this shape, there is no apparent trend for where the search distance is shortest. The

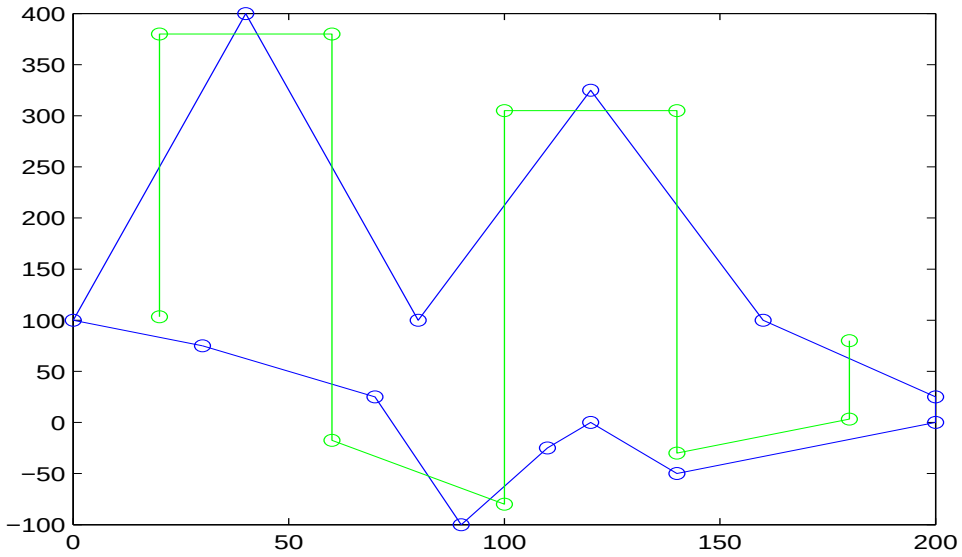


Figure 5.12: The path after each line is reduced with H_{camera}.

red curve shows the length of a lawnmower pattern that covers the entire shape with its surrounding area, as shown in Figure 5.7. This plot illustrates clearly how large the difference between the initial and the post-processed path is. For some rotations, the search distance can be more than halved by post-processing the path.

Algorithm 5.5 Cover area rotation

Do for different values of θ
1. Rotate shape by -θ
2. Apply cover area algorithm and compute waypoints.
3. Compute search distance, and check if better than previous best distance.
4. If better: Rotate shape and waypoints back again by rotating by θ.

5.4 Camera Placement

To get the maximum area coverage with the different search patterns, it is important to place the camera correctly. From an altitude of 50 meters, the two camera placing methods shown in Figure 4.1 covers the areas shown in Figure 5.16. The multicopter is placed in $[0, 0]^T$. The method with gimbal covers a large area in

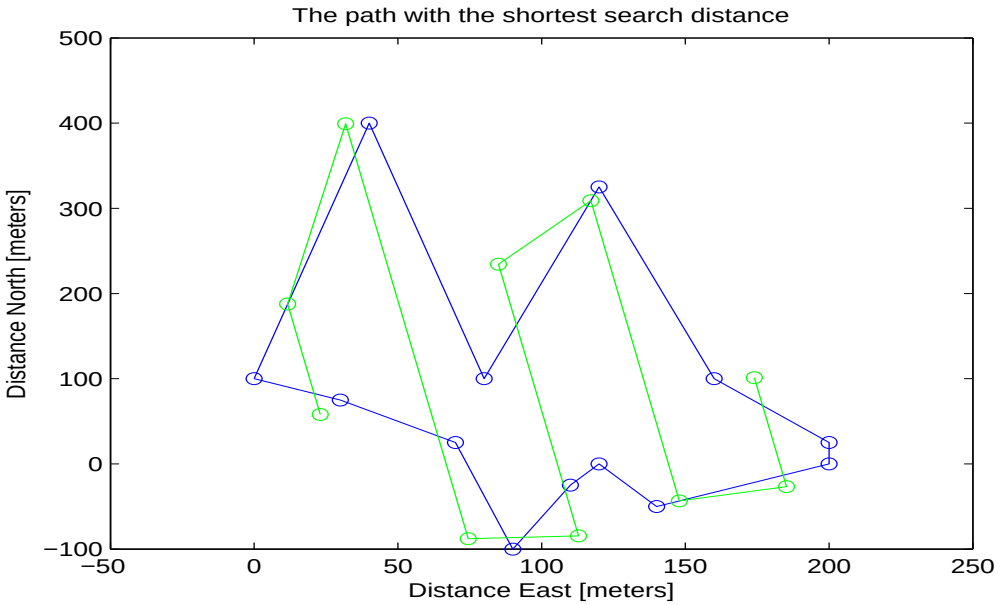


Figure 5.13: The best found path after rotations.

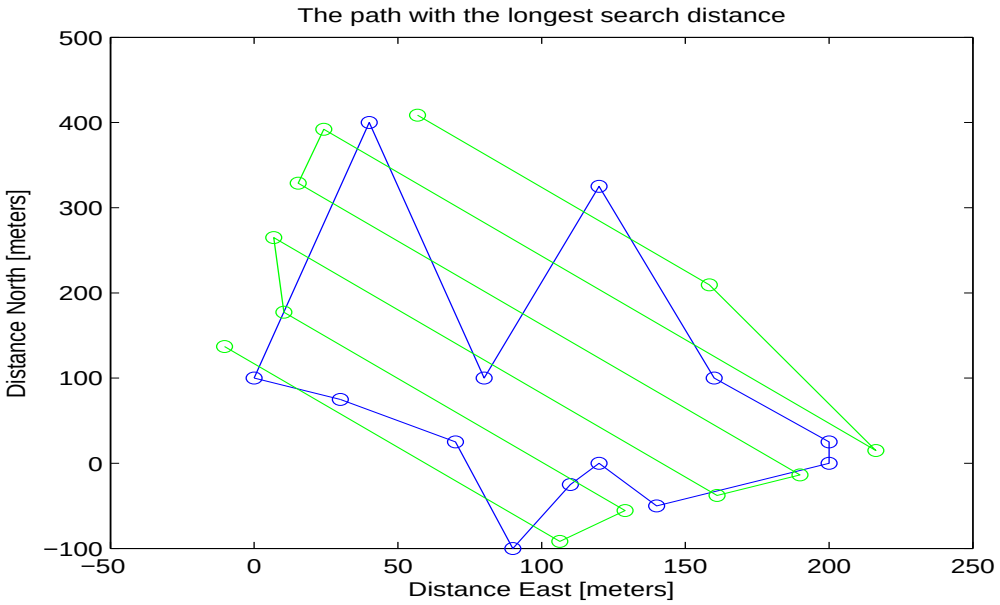


Figure 5.14: The worst found path after rotations.

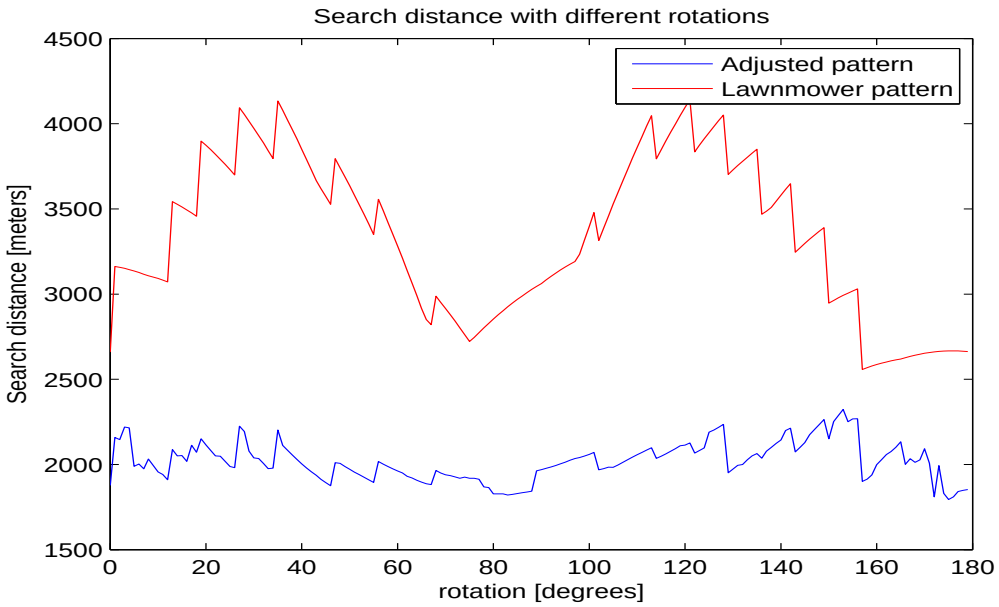


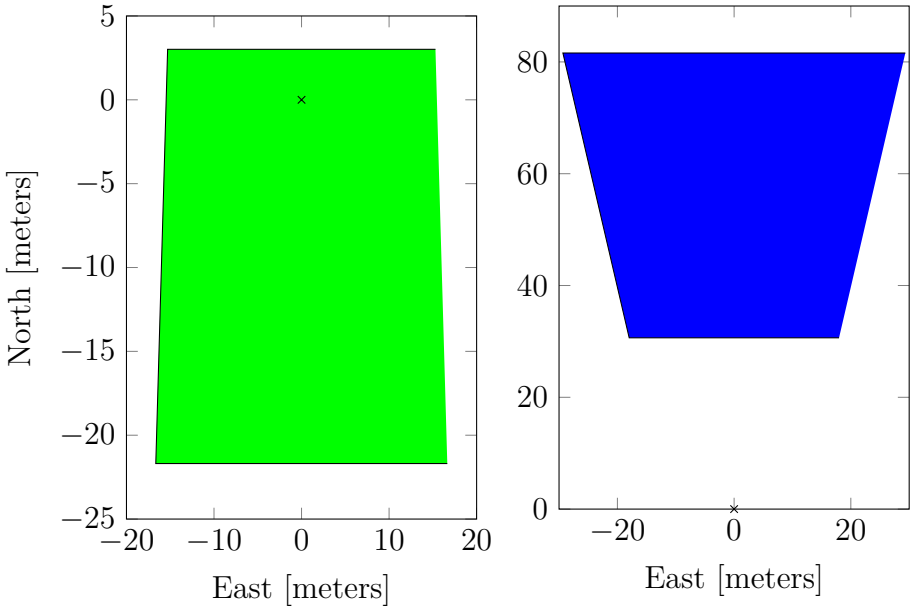
Figure 5.15: Plot showing search distance as a function of θ. The blue curve shows the distance of the post-processed search pattern (Figure 5.13), and the red curve shows the distance of the initial pattern, as in Figure 5.7

front of the multicopter, while the method without gimbal covers a smaller area below and behind the multicopter.

Both methods have advantages and disadvantages. A gimbal is heavy and uses power, but it gives better pictures, and it is easier to compute the area covered. The other method does not require extra weight, and no extra power consumption, but as the multicopter experiences roll and pitch, the pictures become blurry and it is harder to compute the area covered.

Another important factor is the angle of which a picture is taken. With the gimbal, the camera frame will always hit a human with an angle of 45°, assuming flat ground and a standing human. The other approach will have a different angle from time to time, but it will be considerably smaller, meaning the human will occupy less pixels in the picture.

Based on the reasons mentioned above, it was decided to use a gimbal for placing the camera. This was also the method that fits best with the design of the object detection and tracking algorithm from [17].



(a) Without gimbal

(b) With gimbal

Figure 5.16: The area covered by the camera with both methods for placing the camera

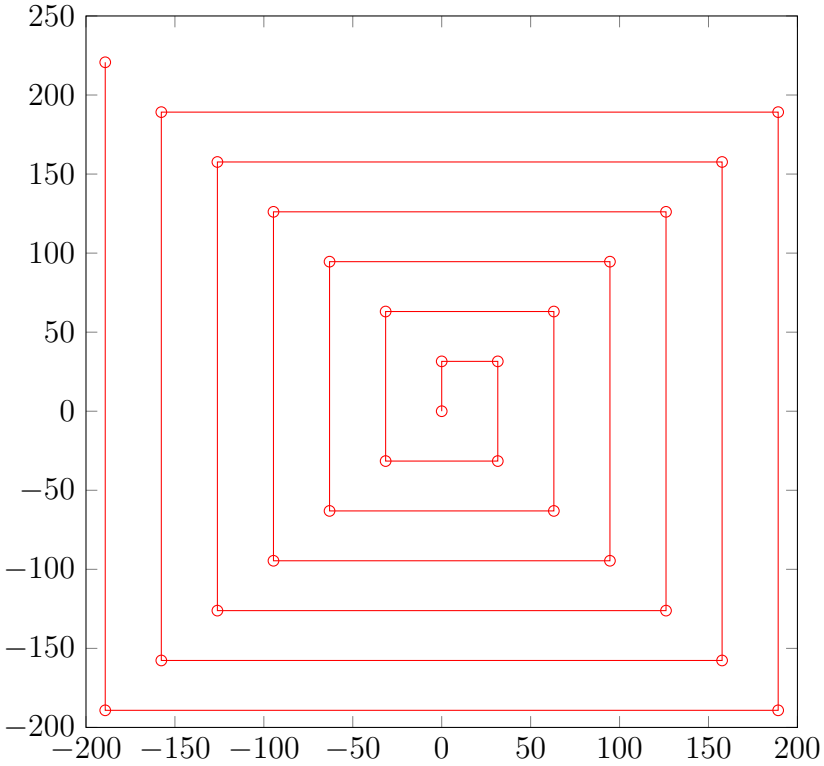
Sideways Coverage

It is also important to place the camera such that the coverage sideways is largest, i.e the largest angel sideways and the smallest angle a long track. This is important for getting the longest possible distance between the search lines when using one of the parallel and creeping line search patterns, or the square search pattern, and thereby getting the shortest overall search length to cover the requested search area.

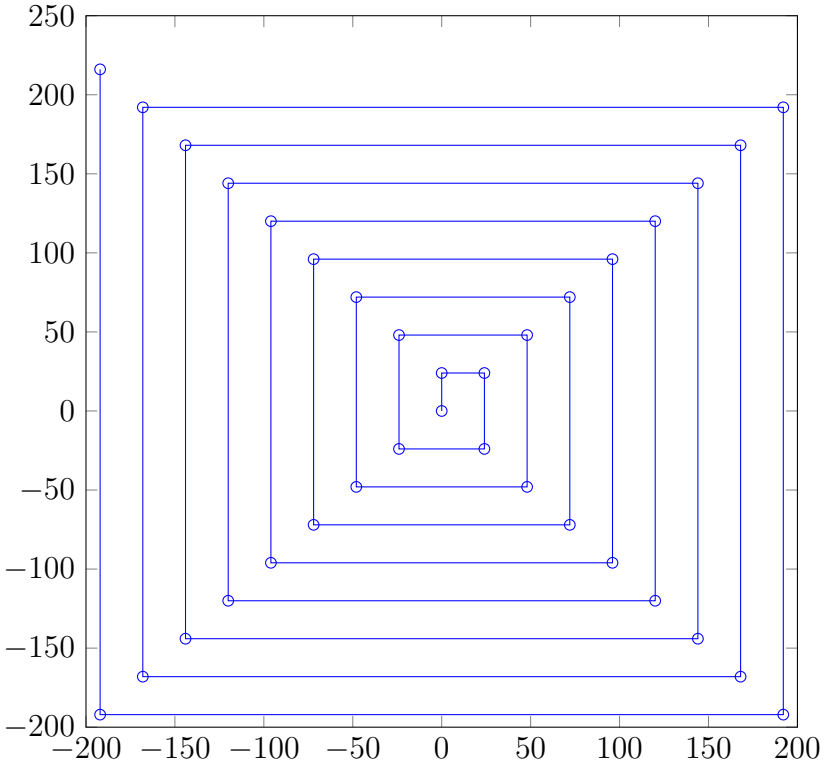
An example of this is shown in Figure 5.17a and 5.17b where a square search pattern is used to cover a 400m × 400m area. Here the pattern is twice created using the angles for the IR-camera described in Section 3.2.2, $\alpha_w = 35°$ and $\alpha_h = 27°$. The distance between each search line S is given by Equation 5.5. With a height of 50 meters and no overlap, S is 31.5 meters and 24 meters for for $\alpha = \alpha_w$ and $\alpha = \alpha_h$ respectively. As shown in the figures, the pattern with wrong camera placement (marked with blue) has less distance between each search line, and thus needs more line to cover the area. Its total length is 6938 meters, while the shorter one has a length of 5329 meters, that is roughly 1600 meters shorter. Therefore, it is important to place the camera such that the sideways coverage is as large as

possible.

$$S = 2 \cdot altitude \cdot \tan\left(\frac{\alpha}{2}\right) - overlap \tag{5.5}$$



(a) The pattern with correct camera placement



(b) The pattern with wrong camera placement

Figure 5.17: Area covered by each search pattern

Chapter 6

Finding the Optimal Search Height

This chapter deals with finding the optimal height for the multicopter to perform a search.

In Chapter 4, an algorithm for deciding the area covered by the camera was outlined. This chapter tries to find the height that gives the largest area covered but is still so small that a human will occupy enough pixels to be recognized by the detection and tracking algorithm. It is assumed a gimbal is used for roll and pitch stabilization, and that the gimbal is set to have an angle $\alpha = 45°$ with the y-axis.

6.1 Area Covered

The algorithm from Chapter 4, outputs the area covered in the form of four points. With the gimbal, these points will make a trapezoid, as can be seen in Figure 6.1. The area of a trapezoid is given by Equation 6.1.

$$A = \frac{1}{2} \cdot h \cdot (w1 + w2) \tag{6.1}$$

Figure 6.2 shows the area covered by a picture frame from different altitudes. Here, it can be seen that the area increases rapidly as the altitude increases. For instance, the area increases by roughly $1200m^2$ between 20 and 40m, while between 80 and 100m, the area is increasing by almost $2500m^2$.

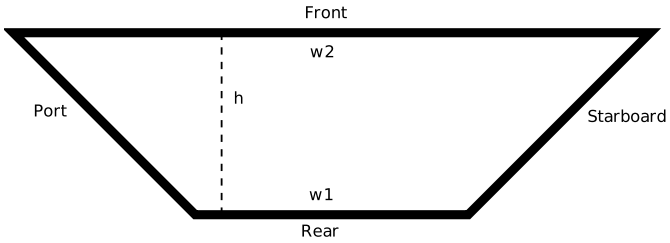


Figure 6.1: A picture shaped as a trapezoid

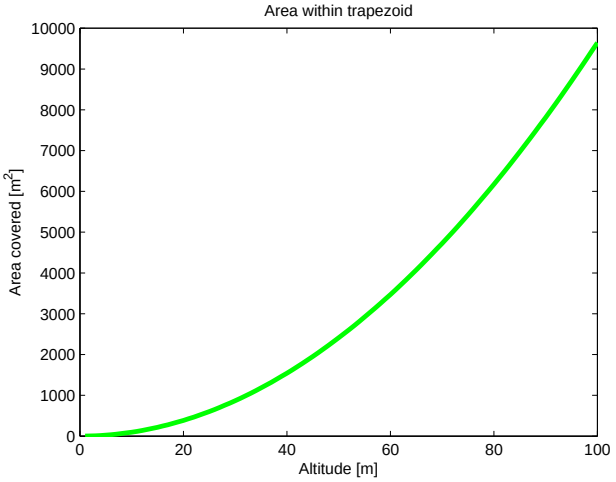


Figure 6.2: The area covered by a picture frame from different altitudes

Table 6.1: Area covered by camera from different altitudes.

Altitude [m]	**Area covered** $[m^2]$
10	96
20	386
30	868
40	1543
50	2411
60	3472
70	4725
80	6172
90	7811
100	9644

Table 6.2: Human size in the center of a frame.

Altitude [m]	**Height [pixels]**	**Width [pixels]**
10	105	39
20	50	19
30	32	13
40	24	9
50	19	7
60	16	7
70	13	5
80	12	5
90	10	5
100	9	3

6.2 Human in a Frame

To see how a human will appear in each picture, the model from Chapter 4 was expanded. In stead of finding only the corner points of the picture, a line is computed for each pixel in the frame. Then, it must be checked whether or not this line intersects with the human. The human is modelled as a plane 1.80 meters high and 0.50 meters wide facing the camera.

According to [17], to achieve good detection and tracking, the subject should occupy a piece of the photo that is minimum 16 pixels high and 8 pixels wide.

Table 6.2, shows how many pixels a human placed in the center of the frame occupies, when the frame is taken from different altitudes. From these results, altitudes above 60 meters can quickly be excluded, as the human in these pictures occupies too few pixels to achieve good detection and tracking. On the other end of the scale, altitudes such as 10, 20 and 30 meters are also excluded as these give a too small area covered in square meters,as seen in Table 6.1. Low altitudes are physically not practical due to obstacles such as trees.

6.3 Usable Area

In the previous section, the optimal height was found to be somewhere between 30 and 60 meters, when looking at how large a human in the center of the frame will appear. This section tries to find how much of the covered area that is suitable for object detection and tracking, i.e where a human occupies enough cells to be

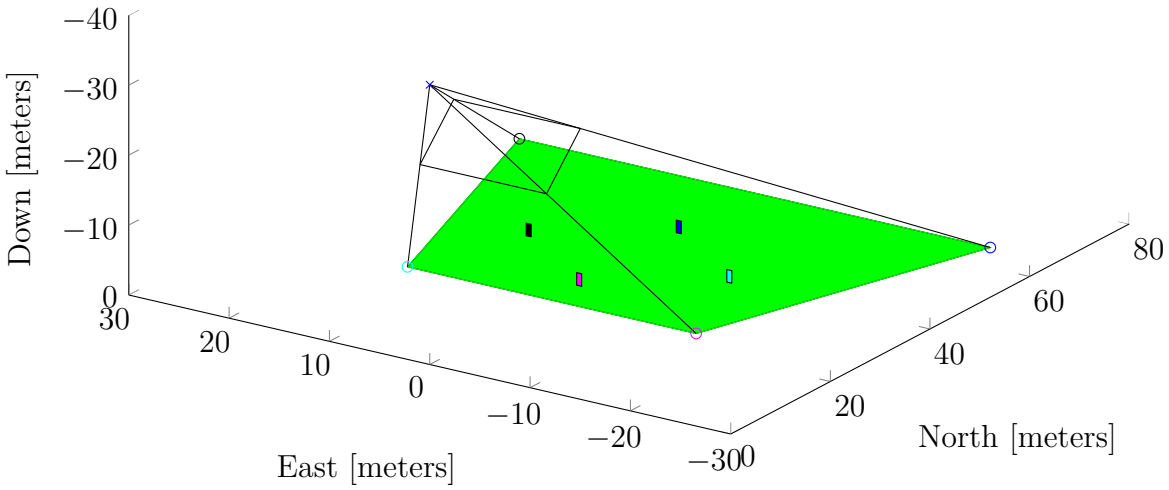


Figure 6.3: Four subjects in the camera frame. The multicopter is placed in $[0, 0, -40]^T$.

detected.

This is done by placing three new subjects in the model. Each subject is the same size. The subjects will then be moved one meter in each direction from the center, forming a circle as seen in Figure 6.3. This is done as long as the subject still are large enough to be covered by the camera. The optimal height will be the one where this circle has the largest radius.

Simulations of the model show that for moving the subjects sideways, the number of pixels it occupies remains constant, although the lines become more oblique as it approaches the side of the frame. As each camera frame will overlap each other when the multicopter moves forward, it is most important to have good coverage sideways.

Figure 6.4 shows how many pixels the subjects from Figure 6.3 occupy. The vertical number of pixels is constant when moving towards the camera, and a small increase in the sideways number of pixels is observed. Moving away from the camera has naturally the opposite effect, the sideways number of pixels decreases slowly, and the vertical number of pixels remains constant, until one barrier value of distance from the center is reached. After this, a more rapid decrease in both vertical and sideways pixels is observe.

This barrier value was found to be roughly the same as the distance from the center to the rear line of the trapezoid. That means the whole trapezoid, except from the part furthest from the camera can be used for object detection and tracking. This is shown in Figure 6.5 where the usable area is marked as green, while the

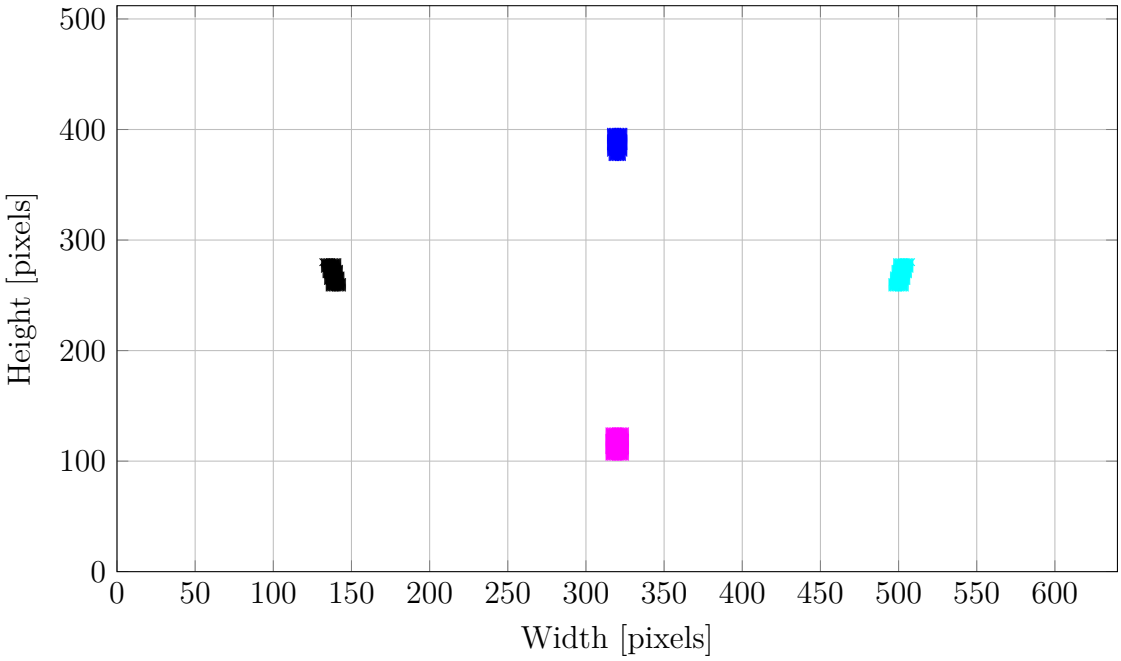


Figure 6.4: The pixels occupied by the subjects

uncertain area is marked as red.

6.4 Discussion

As the subjects get the same size when moving sideways inside the area covered (trapezoid), the optimal height will be the one that gives the maximum sideways coverage and still get enough pixels for object detection and tracking.

From Table 6.2, it can be found that the size of a human from both 50 and 60 meters are both on the border of the minimum given by [17] for what will achieve good tracking and detection. The optimal height is therefore thought to be between 40 and 60 meters. This will be also be strictly dependent on how large the subject to be searched for is, and as the subject used in these simulations is modelled as a relatively large human, a height between 40 and 50 meters would be a more appropriate decision.

In order to test this more thoroughly, pictures of a person should be taken from different altitudes with the infrared camera and tested with the object detection algorithm.

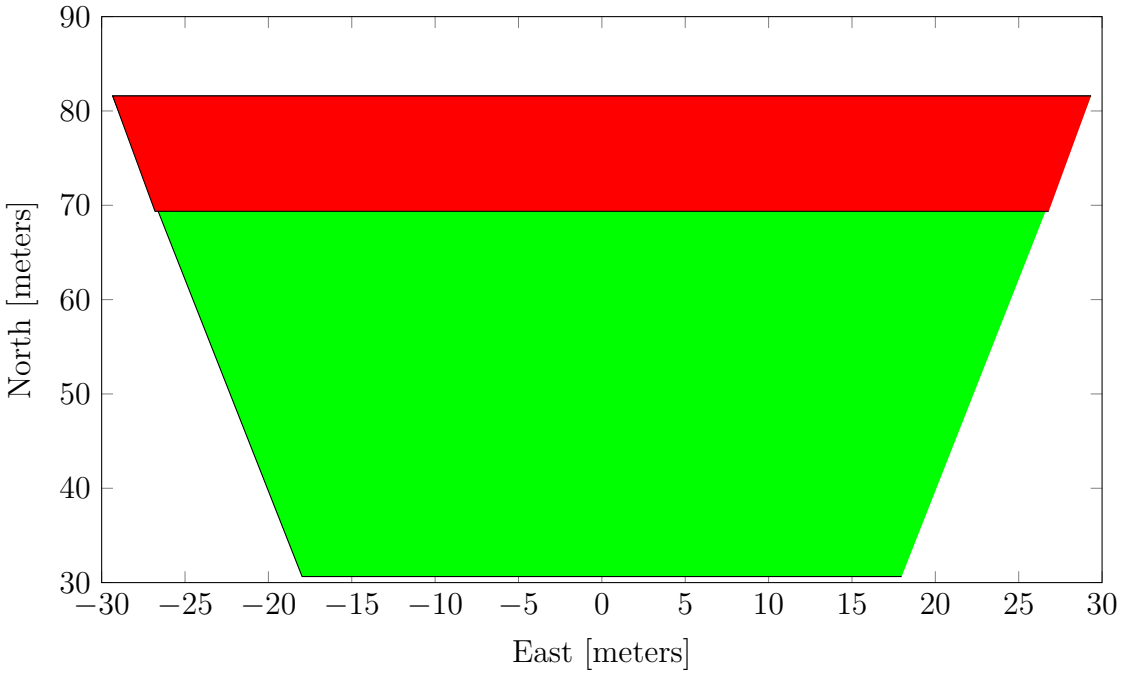


Figure 6.5: The usable area of the frame

Chapter 7

MATLAB Simulations

To test the various search patterns performance in different scenarios, a series of simulations were done in MATLAB. These simulations assume a multicopter that is capable of perfectly following the path. Section 7.1 details the mathematical model for the multicopter with camera. In Section 7.2, a scenario with search for 20 stationary subjects is simulated. An estimate of the area covered with the camera by each search pattern is found in Section 7.3, and in Section 7.4, a search for moving subjects is simulated.

7.1 The Setup

7.1.1 The Multicopter

In this simulations, a multicopter with unlimited acceleration and angular rate was used, which gives the following equations for the multicopter's position in the NED-frame:

$$x_{k+1} = x_k + u \cdot \cos\psi_d \tag{7.1}$$
$$y_{k+1} = y_k + u \cdot \sin\psi_d \tag{7.2}$$
$$z_{k+1} = z_d \tag{7.3}$$

The desired heading ψ_d where calculated by lookahead-based stearing. A height of $z = 50$m, was chosen, as found in Chapter 6.

7.1.2 Camera

The height of $z = 50$m, gave the camera footprint shown in Figure 7.1. To ease computations, this was estimated as a circle with radius $r = 20$m and its origin in the center of the frame, i.e $[x + 50\cos\psi, y + 50\sin\psi]^T$, where (x, y) is the multicopter's position in the NED-frame. The whole circle is placed within the covered area while the rest of the covered area is assumed to be covered by the next circle as pictures are taken continuously. A point $[x^*, y^*]$ on the map will then be within a frame if:

$$\sqrt{(x_c - x^*)^2 + (y_c - y^*)^2} \leq r \tag{7.4}$$

where $[x_c, y_c]$ is the origin of the circle.

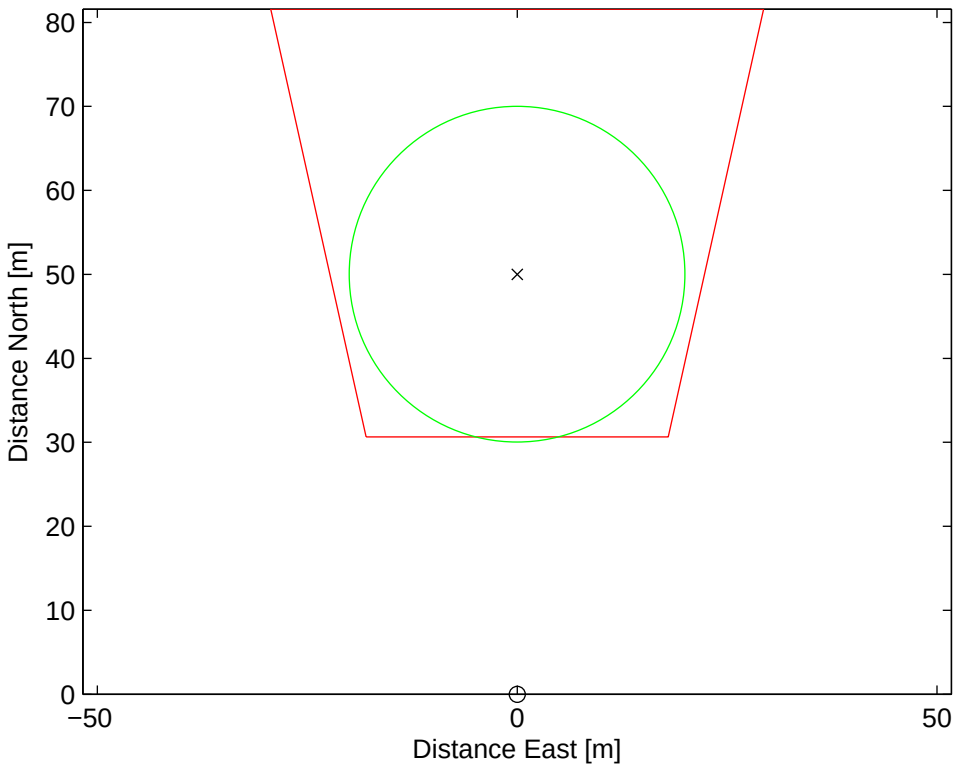


Figure 7.1: The picture frame from an height of $z = 50m$ and position $[0, 0]^T$ when the gimbal sets the camera to point 45° ahead. The green circle is the estimation of the frame used in calculations in this chapter.

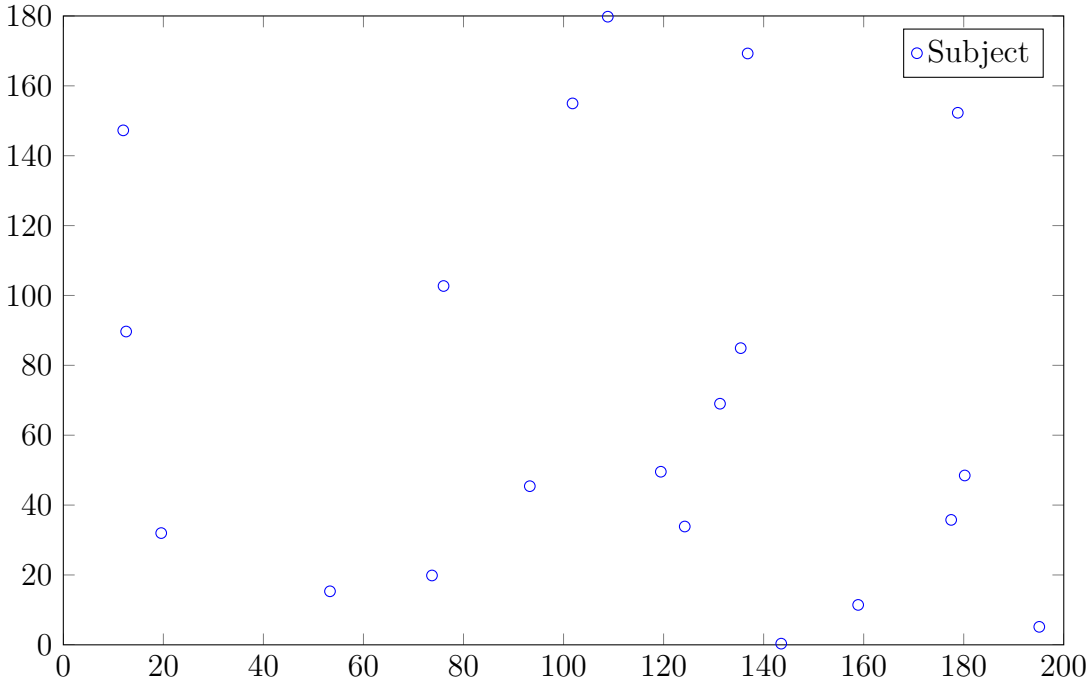


Figure 7.2: The stationary subjects

7.2 Searching for a Stationary Subject

When searching for a stationary subject, the goal is to cover as much area as possible, as fast as possible.

7.2.1 The Subjects

For this simulation, 20 stationary subjects were placed within the area. The search was conducted on a $400m \times 400m$ area, and the CSP was set to $[0, 0]^T$. The parallel and creeping line search patterns search the area to the North and East of the CSP (0 to 400m in each direction), and the square, sector and barrier patrol search the area around the CSP (-200 to 200 m in each direction). Therefore, the subjects were placed in the mutual part, between 0 and 200 m in North and East direction.

The subjects were generated using the *rand()* function in MATLAB, where MATLAB draws a pseudorandom value from the open interval $(0, 1)$ and their positions are shown in Figure 7.2.

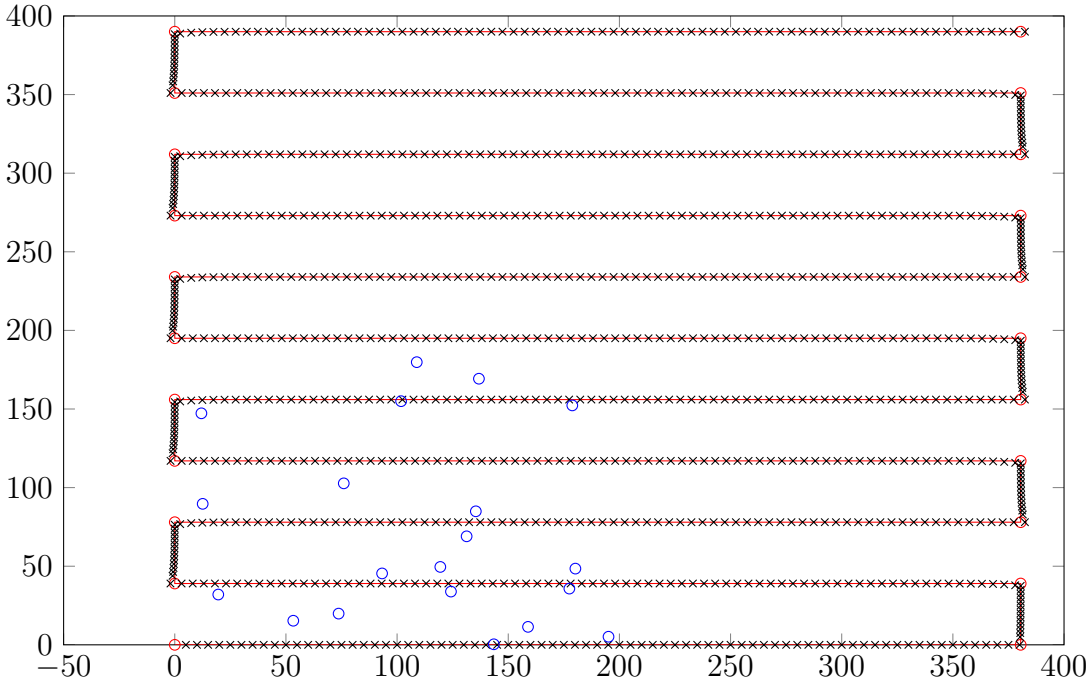


Figure 7.3: Simulation of the parallel search pattern

7.2.2 Simulations

Using the algorithms described in Chapter 5, a waypoint representation of each search pattern was made, and used for simulations with the multicopter model. Simulations were done in Simulink. The speed of the multicopter was set to 5m/s, and one camera frame per second was used for object detection.

Plots of the simulations are shown in figures 7.3 to 7.7. The waypoints are marked by red circles, defining a desired path in red lines between them.

7.2.3 Results

The results of these simulations are shown in Table 7.1. By the three first search patterns, 19 of the 20 subjects were found. Sector search found 17 of the subjects. The worst performance in this simulation was the barrier patrol search pattern, that only found 5 of the subjects. As seen in Figure 7.7, there are too wide space between the lines for the camera to cover the entire area of interest. The barrier patrol search pattern is however designed for finding moving subjects, and therefore a poor result in this simulation was expected.

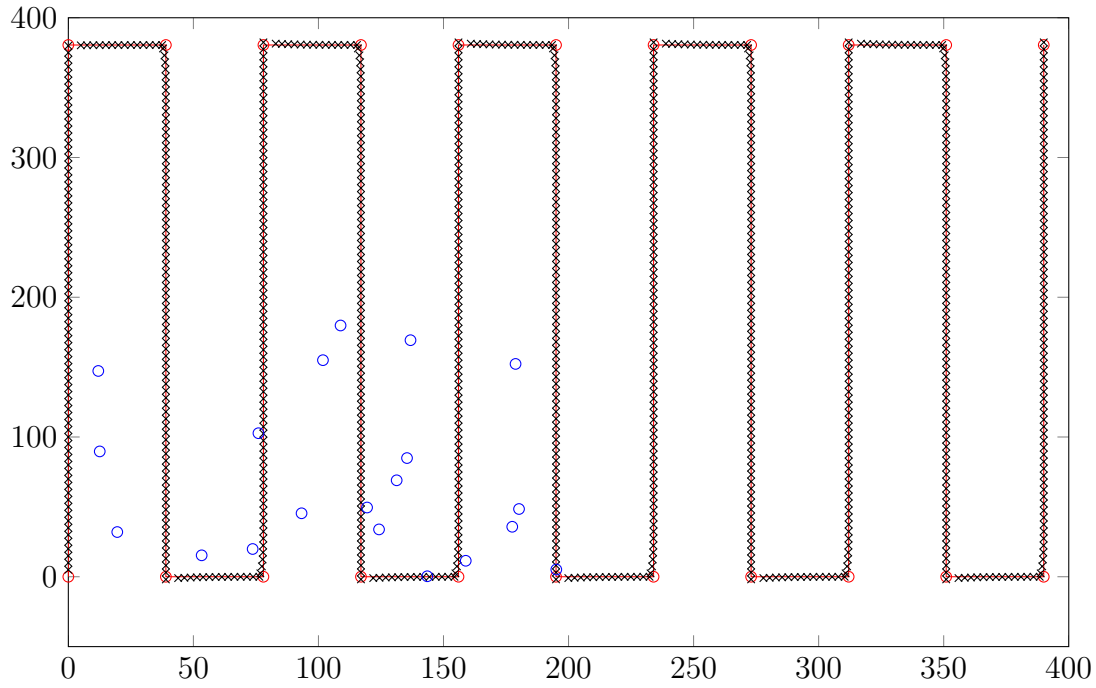


Figure 7.4: Simulation of the creeping line search pattern

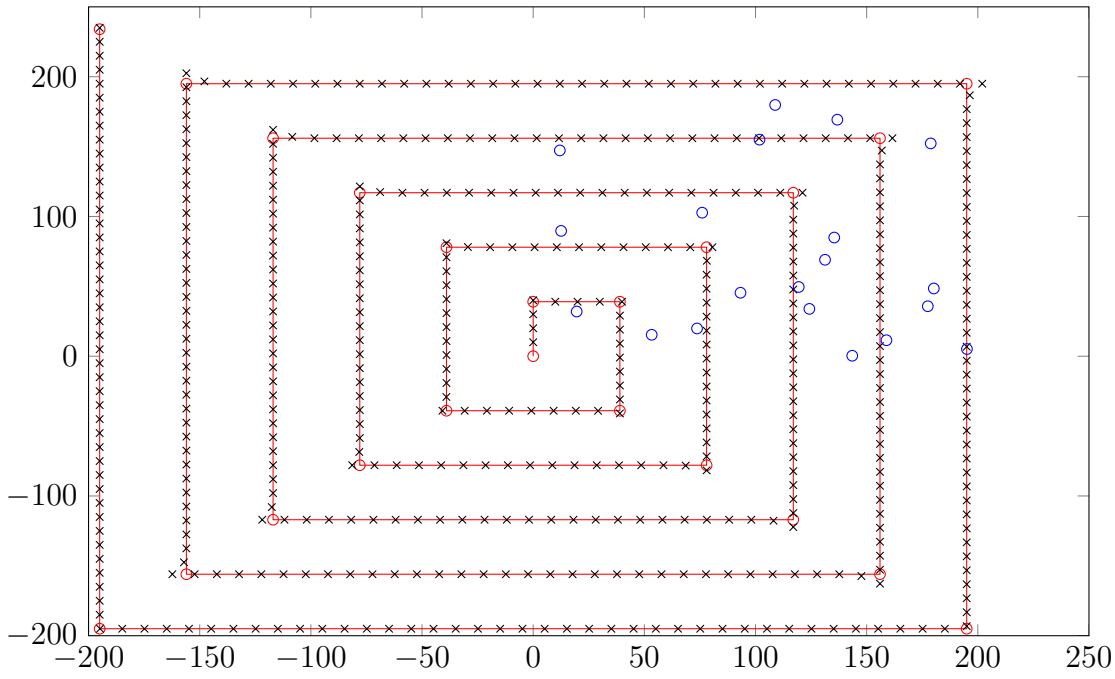


Figure 7.5: Simulation of the square search pattern

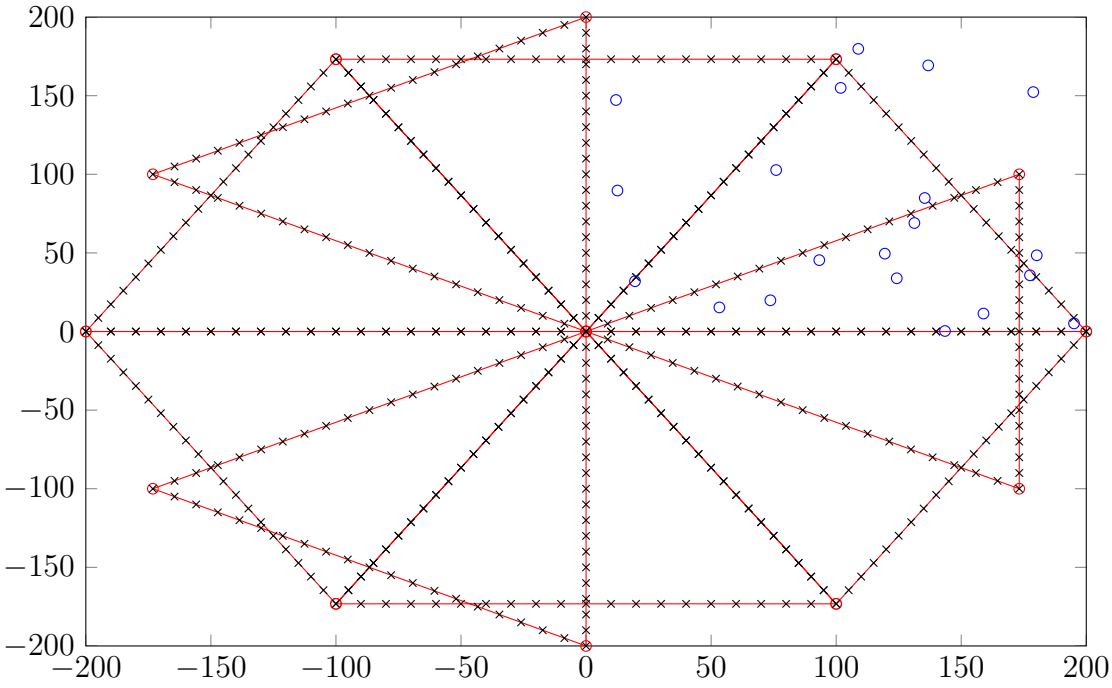


Figure 7.6: Simulation of the sector search pattern

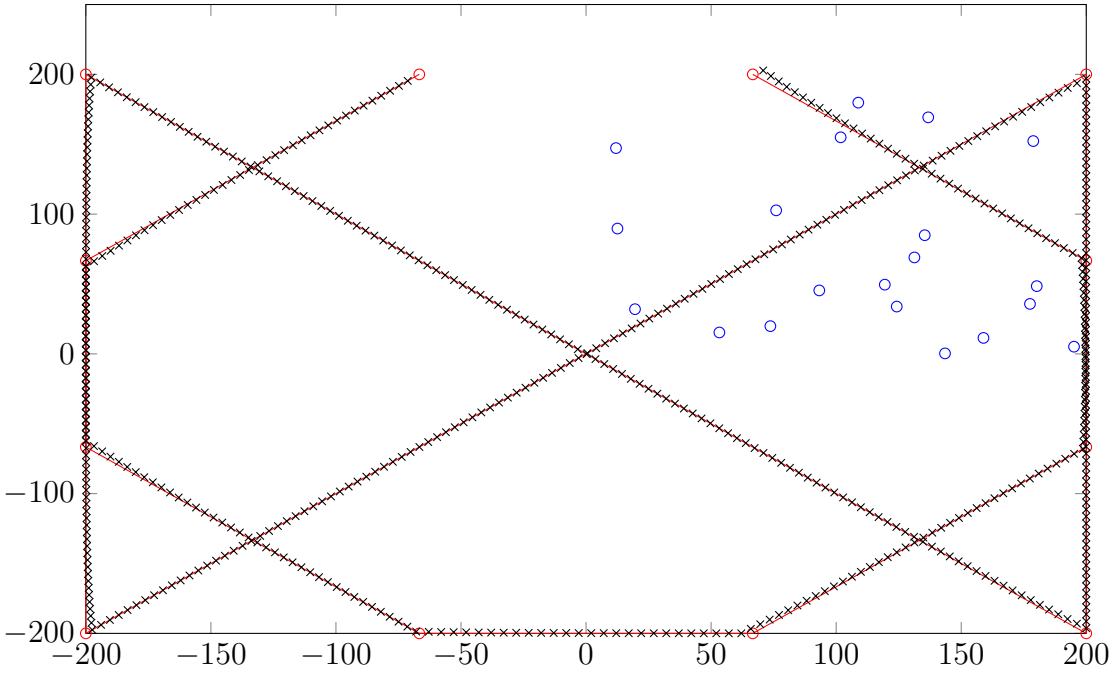


Figure 7.7: Simulation of the barrier patrol search pattern

Table 7.1: Results of search for stationary subject simulation

Search pattern	Parallel	Creeping	Square	Sector	Barrier patrol
Subjects found	19	19	19	17	5

Search length

The length of the path for each search pattern on this area is given by Table 7.2. When these are considered, it can be seen that the parallel and creeping line search patterns have the shortest distance of the three that finds the most targets. The sector search pattern has the longest search distance which is about 800 meters longer than the parallel and creeping line, and combined with not finding the same number of subjects, there are few good reasons for choosing this pattern for this scenario.

The barrier patrol search pattern has by far the shortest distance of the search; it does however not find many subjects.

Table 7.2: Search length for the patterns on a 400m × 400m area.

Search pattern	Parallel	Creeping	Square	Sector	Barrier patrol
Search length	4576	4576	4719	5400	3086

7.3 Estimated Area Coverage

The same simulation-setup can be used to estimate the area coverage of each search pattern. The search area can be considered as a matrix, where each element represents a point in the area. When the distance in each direction is equal and the resolution in each direction is the same, i.e the matrix is quadratic; the distance between each point is equal in x- and y-direction and is given by:

$$d = \frac{distance}{resolution} \tag{7.5}$$

Equation 7.4 is then used to check whether each point is covered. The matrix stores a 1 if the element is covered by a frame, and 0 otherwise. As the resolution has to be a large number, i.e. 100, this matrix becomes large (100 × 100) and therefore, this method is not suited for other implementations than simulation in MATLAB. The area coverage in percent is found by:

$$areacoverage = \frac{sum(matrix)}{resolution^2} \cdot 100\% \tag{7.6}$$

The results for each search pattern are shown in Table 7.3. The barrier patrol and sector search patterns performed worst in this simulation, as they do not provide full area coverage. However, they have other interesting properties.

The areas covered by the search patterns, with this camera setup, are shown in Figure 7.8. The parallel and creeping line search patterns lose some area coverage when turning. This is because the center of the camera frame is 50 meter ahead of the multicopter, and the multicopter turns fast as it has unlimited angular rate. This will improve a bit when using a physical multicopter. Another solution could be making the lines longer, and do the turning outside the search area.

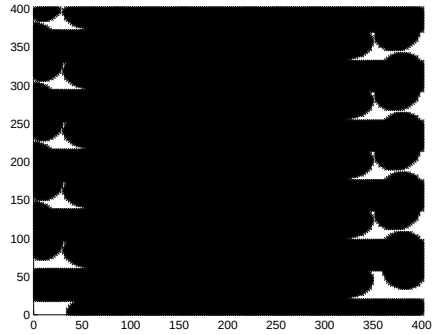
The square search has an uncovered part in the middle of the area, around its starting point and the CSP. This is because the lines here are much shorter than the 50 meters which the camera frame is ahead of the craft. However, in a real life scenario the multicopter has to get to the start of the search somehow. If it flies towards the starting point it will cover this area on the way. If the search starts in this point, the searchers responsible for the multicopter will have to put it in that position, and therefore this area will be covered regardless.

The sector search will not cover the whole area. As can be seen in Figure 7.8d, it offers very good coverage of the CSP and its surrounding area. Outside this, there are large portions of the area that ends between the search lines, which will not be covered. These areas grow when the length of the search area increases.

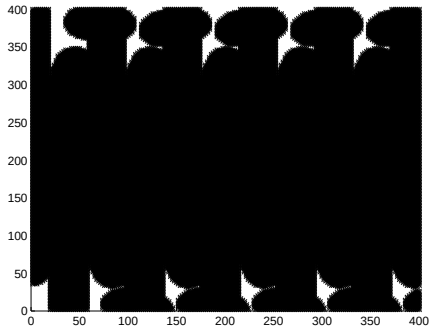
Nor the barrier patrol search pattern will cover the entire area. This search pattern focuses on the borders of the search area, with the exception of the two diagonal crossings.

Table 7.3: Estimated area coverage in percent for the search patterns

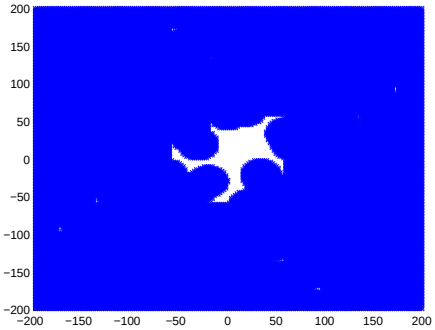
	resolution		
Search Pattern	50	100	200
Parallel	92.72	92.96	93.12
Creeping line	92.72	92.94	93.10
Square	96.12	96.61	96.53
Sector	73.92	73.39	73.54
Barrier Patrol	45.20	45.21	45.06



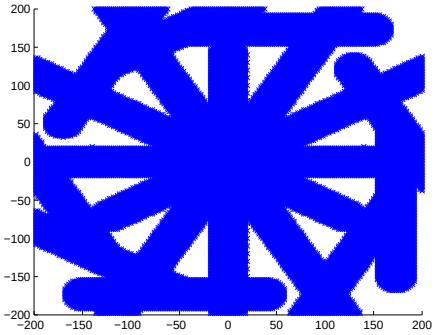
(a) Parallel



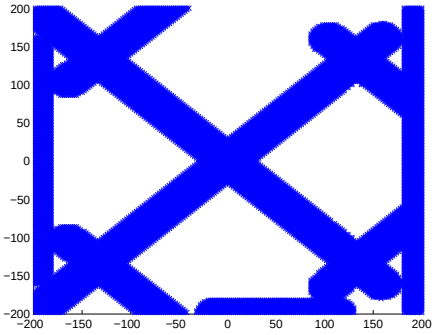
(b) Creeping line



(c) Square



(d) Sector



(e) Barrier

Figure 7.8: Area covered by each search pattern

7.4 Searching for a Moving Subject

When the subject is stationary, the goal is to cover as much area as possible. When the subject is moving, the goal is to cover the right area at the right time, to be able to detect the subject.

7.4.1 The Moving Subjects

For this simulation, four moving subjects were modelled in Simulink. The subjects had different starting positions, heading and walking speed. White noise was added to the speed and heading, to simulate terrain differences.

The path of each moving subject is shown in Figure 7.9. The CSP is still $[0, 0]^T$, and the subjects were given starting points close to this.

The red subject starts in $[50, 30]^T$ and moves with a speed of 1.0m/s in a random direction before it ends in the North-East corner of the area. The pink subject starts in $[40, 40]^T$. It moves slowly with a speed of 0.5m/s North-East. The blue subject starts far from the CSP, in $[150, 150]^T$, and moves westward with a speed of 1.1m/s. The yellow subject starts South in the area in $[0, 150]$. From there it walks North with a speed of 0.4m/s.

7.4.2 Simulations and Results

The same simulations as in Section 7.2 were executed to find these subjects. The area's dimensions were the same, and thus, some of the patterns will experience the subjects quickly leaving the search area. Plots of the simulations are shown in figures 7.10 to 7.14.

The results of these simulations are summarized in Table 7.4. The square search pattern did best, as it managed to detect all four subjects. The parallel and sector search patterns came second by detecting three of four subjects. Neither managed to find the blue subject. Parallel did not find the blue because it started too far from the CSP, and quickly disappeared from the search area. The sector search pattern crosses the blue subject's path six times, as can be seen in Figure 7.13, but it is always at the wrong time and thus it did not find the subject.

The worst performers in this test were the creeping line search pattern, and again the barrier patrol search pattern, which both only found one of the subjects. Creeping line discovered the red subject, which path crosses its lines several times. It

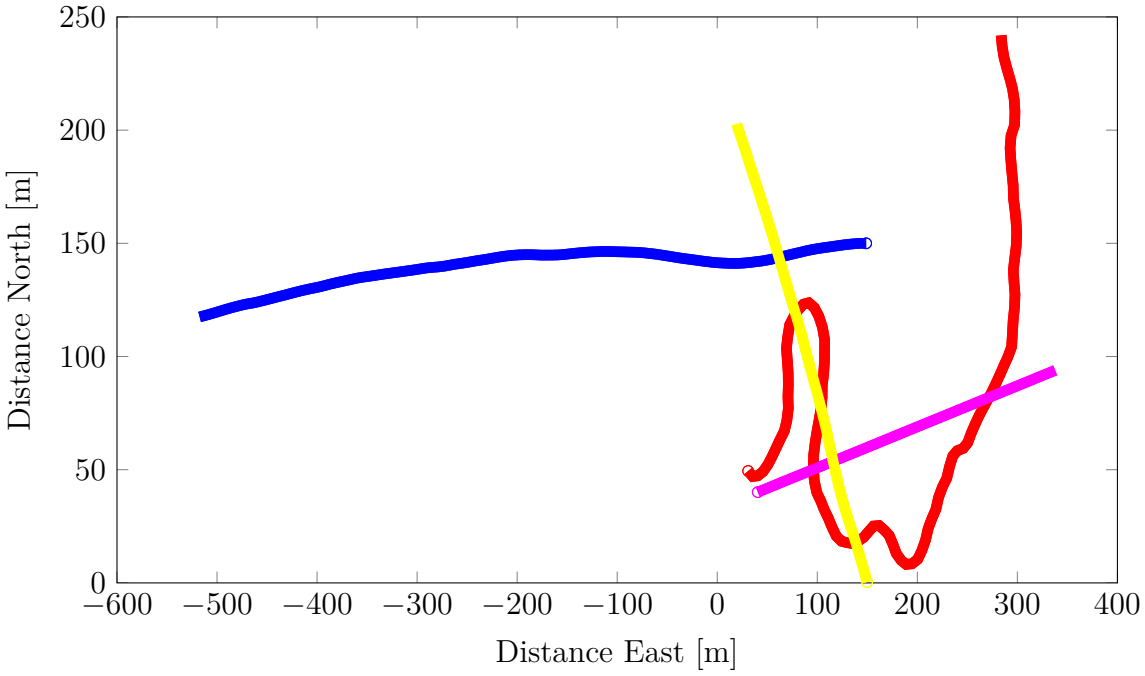


Figure 7.9: The stationary subjects

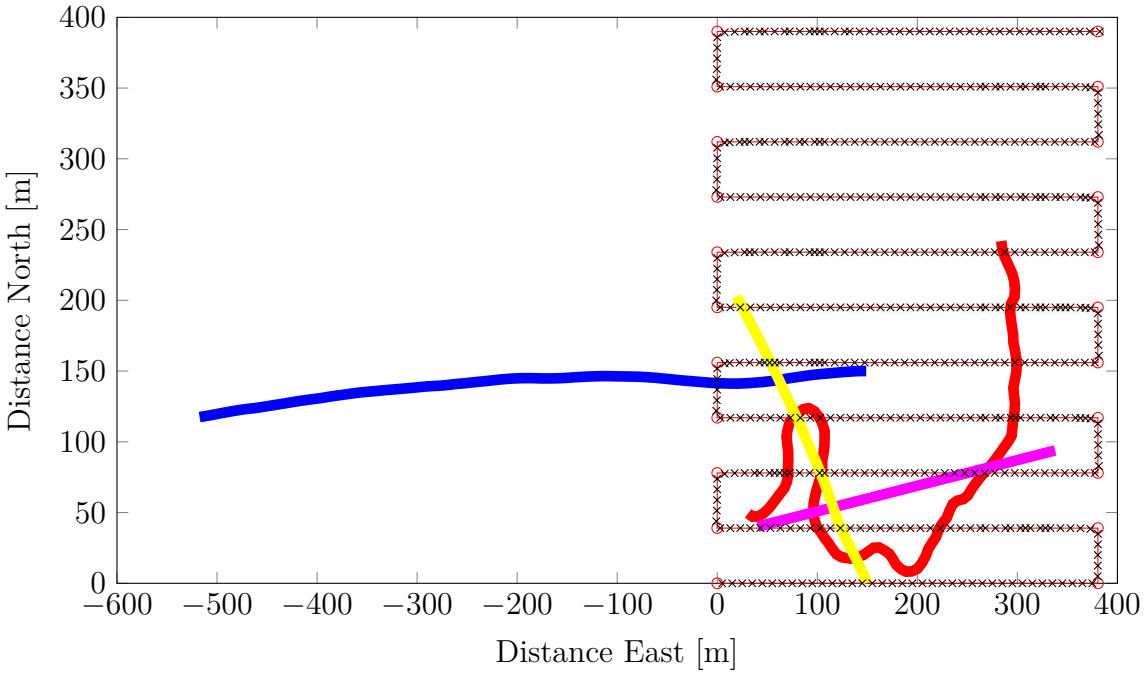


Figure 7.10: Simulation of the parallel search pattern

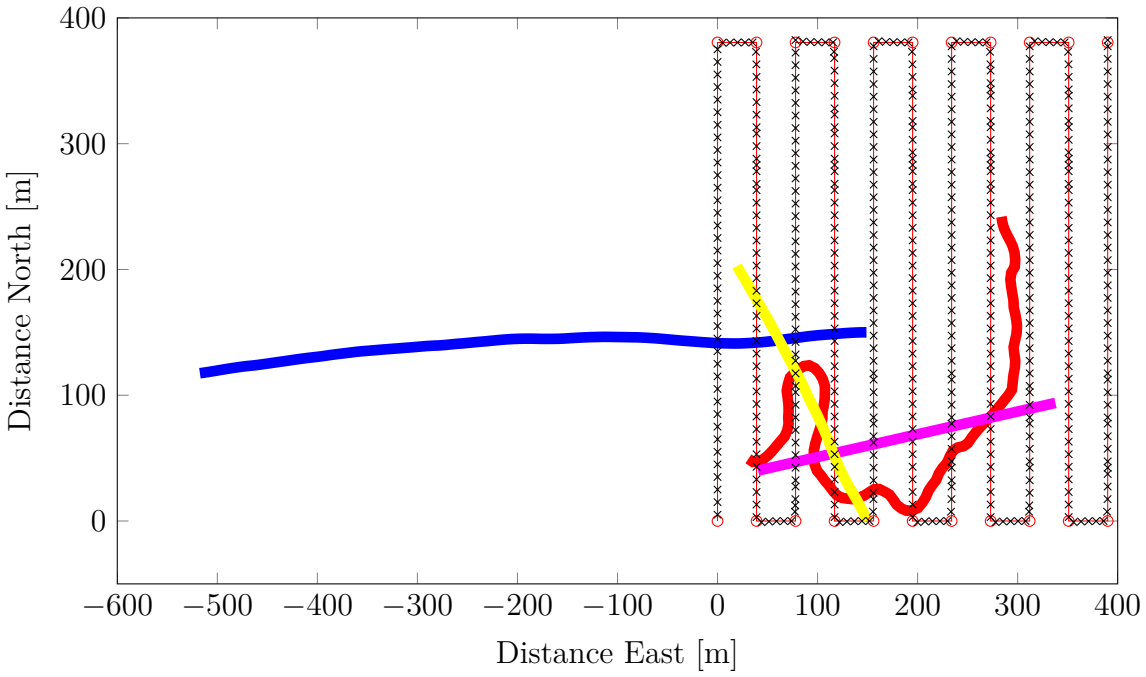


Figure 7.11: Simulation of the creeping line search pattern

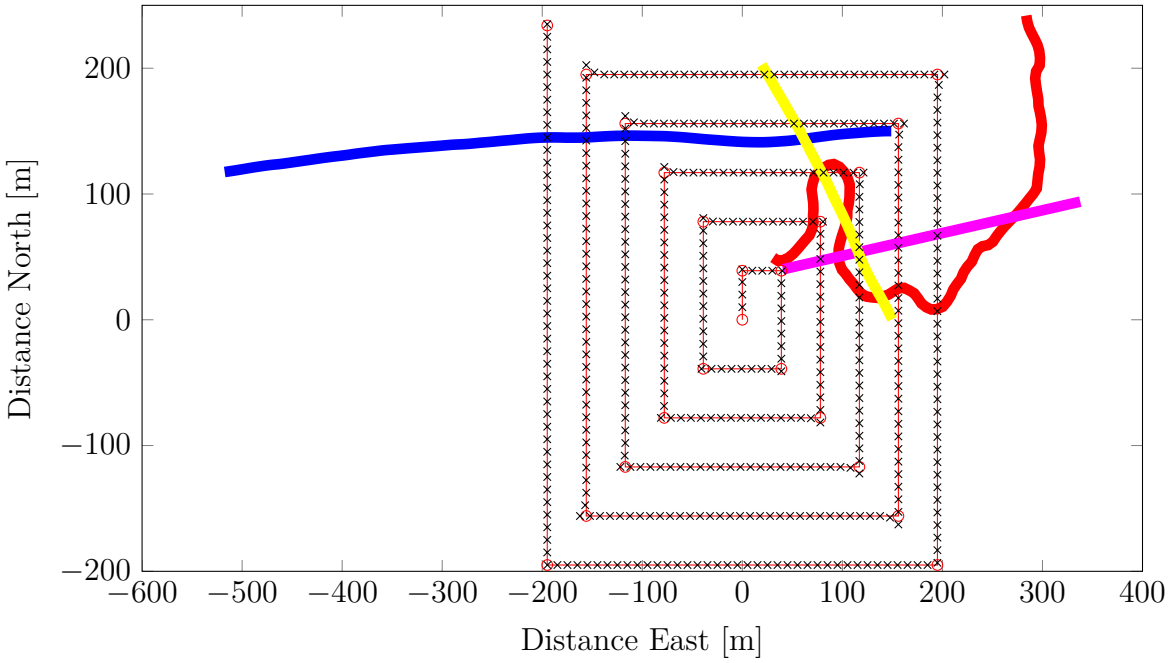


Figure 7.12: Simulation of the square search pattern

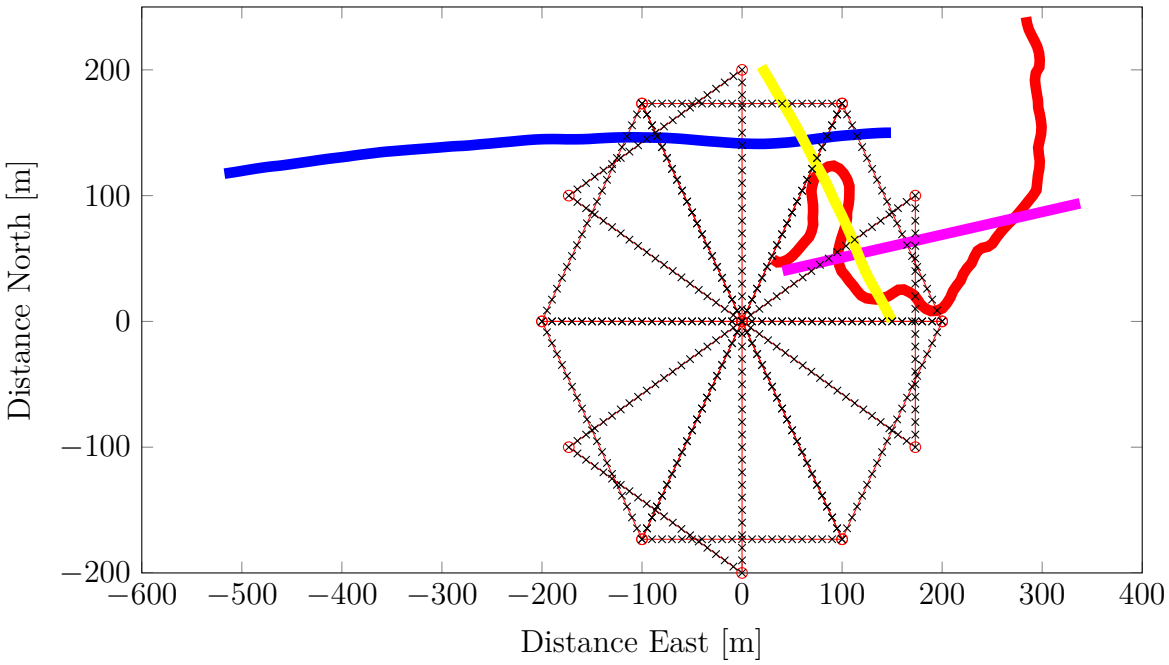


Figure 7.13: Simulation of the sector search pattern

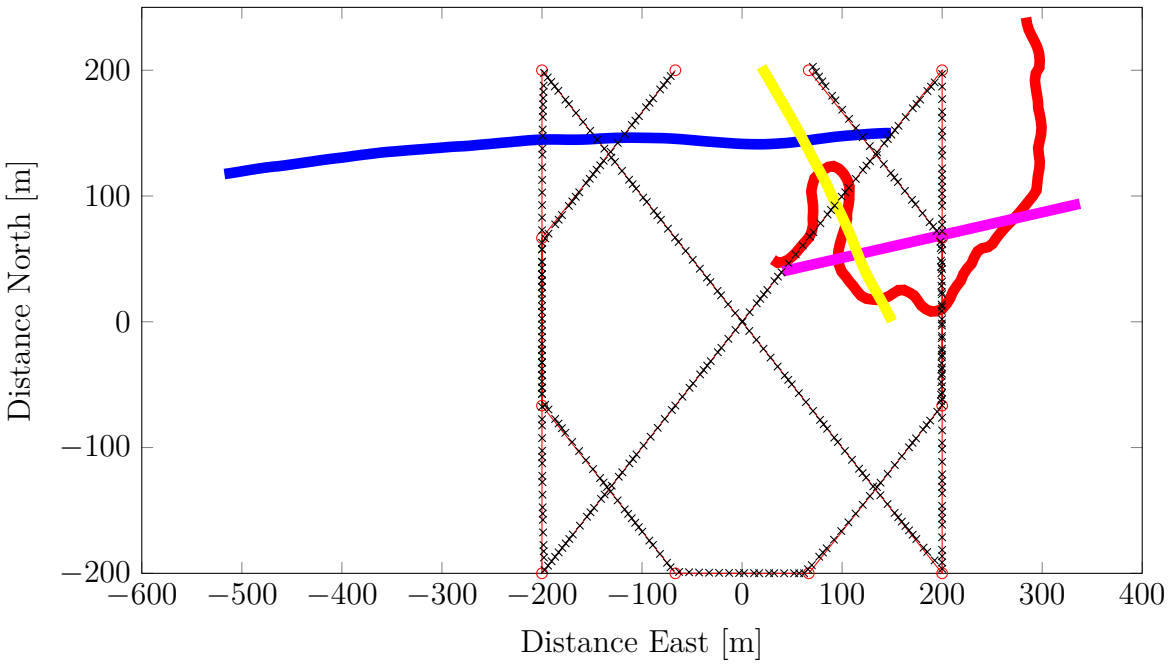


Figure 7.14: Simulation of the barrier patrol search pattern

Table 7.4: Results of search for moving subject simulation

Search pattern	**Red**	**Pink**	**Blue**	**Yellow**
Parallel	found	found	not found	found
Creeping	found	not found	not found	not found
Square	found	found	found	found
Sector	found	found	not found	found
Barrier	not found	not found	found	not found

did not find the pink or yellow subject, despite the fact that they stay inside the area covered by the pattern the entire time. The blue subject moving towards the start of the search was not found, even though it crosses four search lines.

The barrier patrol search pattern only finds the blue subject, even though the three other subjects all cross the barrier and leave the search area.

7.5 Discussion

In this chapter, the various search patterns performances were tested in two different scenarios. In these simulations, the best results came with the square search pattern, which found 19 of 20 of the stationary subjects in the first simulation, and 3 of the 4 moving subjects in the other simulation. When estimating the area coverage, the square search covered 96% of the area with the remaining 4% being around the start area in the middle. As noted, this is because the simulation starts with the multicopter in the first waypoint. The remaining area will probably be covered either by a ground team setting up the multicopter, or by the multicopter flying towards the first waypoint of the pattern. The square search pattern was also found to have the best performance in the simulations in [7].

The parallel and creeping line search patterns both found 19 of the 20 stationary subjects. From Figure 7.8, it can be seen that both do not provide full area coverage. The area coverage was estimated to 93 % for both patterns. The patterns lose some area coverage when turning, due to how the camera was chosen to be placed, with the camera pointing 45° forward. With a different camera placing, these two patterns would provide 100% area coverage. A way to get 100 % area coverage with this camera placing is to extend the search lines such that the turning is done outside the search area, this will however increase the total search length.

When using these two patterns for finding the moving subjects, the parallel search pattern found 3 of the 4 subjects, while creeping line only discovered 1 subject.

These patterns are weak for finding moving subjects if the subjects start to far from the start of the search and move fast, or if the search lines are too long, which is illustrated by Figure 7.15. Here a moving subject, drawn as a blue circle, is moving across the search lines of a creeping line search pattern. The multicopter is illustrated by a red rectangle, and the area covered by the multicopter's camera is illustrated by a yellow circle. If the subject can move from point A to point B, using shorter time than the multicopter uses on the corresponding distance, it will never be discovered by the multicopter. Said in another way, if the time used by the subject to complete two short lines are shorter than the time used by the multicopter to complete two short and two long lines, the multicopter will not find the subject, if the subject starts inside the search area and moves in the same direction as the search pattern.

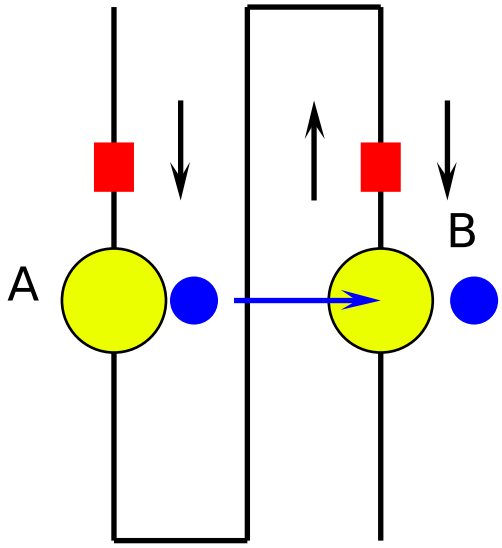


Figure 7.15: How the subject avoids the search. If the subject illustrated by the blue square moves faster from A to B than the multicopter illustrated by the red rectangle with a yellow circle indicating its camera coverage, the multicopter will not catch up with the subject, and will not detect it.

To avoid such a problem the three following factors can be considered; the multicopter's speed can be increased, the search lines decreased or the sideways coverage (and thus the short lines) increased.

The sector search pattern performed well on the stationary subjects, finding 17 of 20, even though it does not provide full area coverage. The area coverage was estimated to 73%. On search length, it scored worst, having a total length that was almost 900 meters longer than parallel and creeping line search patterns. In

the moving subject simulation, it found 3 of the 4 subjects, and then performed as good as the parallel search pattern, only beaten by square search that found all the subjects. When searching for moving subjects, the sector search has the advantage that it quickly moves from one part of the area to another, and is thus able to intersect with the path of the different moving subjects faster. However, it pays the price for this by not gaining complete area coverage.

As seen in Figure 7.8d, the sector search provides good coverage in the area in the middle, around the CSP. This is the main advantage of the sector search. In a real scenario, this area may be covered, for instance by dense vegetation, and thus getting good pictures could prove difficult. It would then be advantageous to cover this area multiple times from different directions to get the best camera pictures.

The barrier patrol search pattern performed worst in all the tests. It only managed to find 1 of the 20 stationary subjects and 1 of the 4 moving subjects. A poor performance for finding the stationary subjects was expected as it is designed to find moving subjects, which is also seen by its estimated area coverage of only 45%.

When searching for moving subjects however, a better performance was expected. Figure 7.14 shows that all of the subjects' paths cross one of the search lines a couple of times. Still, only one of them is discovered.

Several reasons can be behind this poor result such as bad implementation of the pattern, too large area and/or too low flying speed of the multicopter.

Chapter 8

Implementation and Simulations

In Neptus, a *mission plan* is specified as a set of maneuvers (each with a specific type and parameters) and transitions between those maneuvers, forming a graph. A *maneuver* is thus a unit of work that can be accomplished by a specific vehicle or a class of vehicles by instantiating a controller that potentially changes the physical state of the vehicle. A *transition condition* is a boolean expression that can be evaluated against the vehicle state or triggered by asynchronous events [22]. A typical transition is switching between waypoints, where the transition condition may be the *circle of acceptance* or the *along-track distance* found in Section 2.2.1. Existing maneuvers in Neptus include among others *Goto*, *Loiter*, *Rows* and *Hover*. To use the current mission planning scheme in Neptus, each search pattern was implemented as a maneuver.

Neptus and Dune are large complex systems, and takes lots of time and effort to learn. This chapter details the implementation of the search patterns from Section 5.2.The sector search pattern is used as an example of an implemented maneuver.

In the GNC-system explained in Section 2.2, Neptus where the mission planning is done, is in the *Mission Planner*-block. The Dune-tasks, Maneuver.SectorMan and Control.UAV.ArduCopter, will be in the *Guidance System*-block. Different from the figure, Neptus sends the waypoints indirectly in the form of an IMC-maneuver-message, IMC::SectorMan, that specifies the parameters of the maneuver, and tells the Guidance System to compute the waypoints.

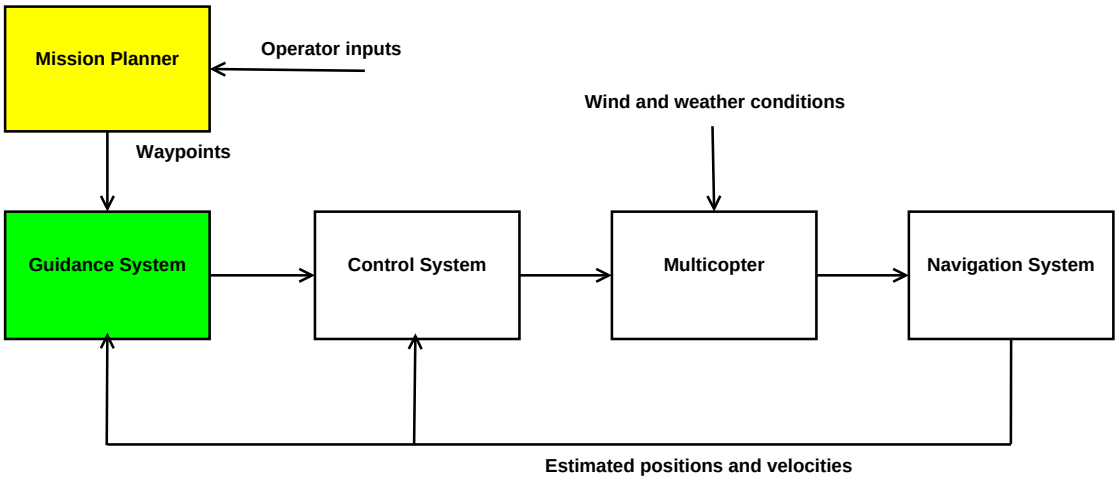


Figure 8.1: The GNC signal flow. Neptus is in the Mission Planner-block, coloured with yellow, and the implemented Dune-tasks are in the Guidance System-block, coloured with green.

8.1 IMC

The first step of the implementation is to find whether or not an existing IMC-message can be used for the communication between Neptus and Dune. For the parallel and creeping line search pattern, the existing IMC-message for the Rows maneuver was used, while for sector search, square search and barrier patrol, new IMC-messages were created.

The IMC-messages are defined in an XML-file, and can be translated into different language bindings using XSLT. This is an important property as a Java-version for Neptus is needed, as well as a C++ version for Dune. Parts of the XML-file containing the created IMC-messages are shown in Appendix C. The list of IMC-Maneuver-messages is shown in Listing 8.1, where the created maneuvers area added in the maneuver message-group.

Listing 8.1: The IMC-Maneuver-Messages

```
1 <message-groups>
2     <message-group name="Maneuver" abbrev="Maneuver">
3       <message-type abbrev="Goto"/>
4       <message-type abbrev="PopUp"/>
5       <message-type abbrev="Teleoperation"/>
6       <message-type abbrev="Loiter"/>
7       <message-type abbrev="IdleManeuver"/>
8       <message-type abbrev="LowLevelControl"/>
9       <message-type abbrev="Rows"/>
```

```
10        <message-type abbrev="SquareMan"/>
11        <message-type abbrev="SectorMan"/>
12        <message-type abbrev="BarrierMan"/>
13        <message-type abbrev="FollowPath"/>
14        <message-type abbrev="YoYo"/>
15        <message-type abbrev="StationKeeping"/>
16        <message-type abbrev="Elevator"/>
17        <message-type abbrev="FollowTrajectory"/>
18        <message-type abbrev="CustomManeuver"/>
19        <message-type abbrev="VehicleFormation"/>
20        <message-type abbrev="CompassCalibration"/>
21        <message-type abbrev="CoverArea"/>
22        <message-type abbrev="FollowReference"/>
23        <message-type abbrev="CommsRelay"/>
24        <message-type abbrev="FormationParameters"/>
25        <message-type abbrev="FormationPlanExecution"/>
26      </message-group>
27 </message-groups>
```

In an IMC-message, each variable is defined by a field. The field must contain a variable name, used mostly for displaying, an abbreviation, which is the variable's name in the code and a data type, i.e signed or unsigned integer, floating point, text etc. and size 8, 16, 32 or 64 bit. It is also possible to add minimum, maximum and initial values of each field and the unit. Listing 8.2 shows an example of a field in an IMC-message. This field is the pattern bearing angle, which is the pattern's rotation around the z-axis.

Listing 8.2: An example of a field from a XML-file

```
1 <field name="Bearing" abbrev="bearing" type="fp64_t" unit="rad"
      min="0" max="6.283185307179586">
2   <description>
3     Pattern bearing angle.
4   </description>
5 </field>
```

8.2 Neptus

Neptus is where the operator creates the mission plan, and chooses the properties of each maneuver. The mission planner in Neptus is called *Plan Edition*, and is found to the left on the screen shown in Figure 8.2, marked as a red P.

When accessing the mission planner, the operator is asked to specify the vehicle for which he or she will create a plan. Each vehicle is defined by a vehicle definitions-

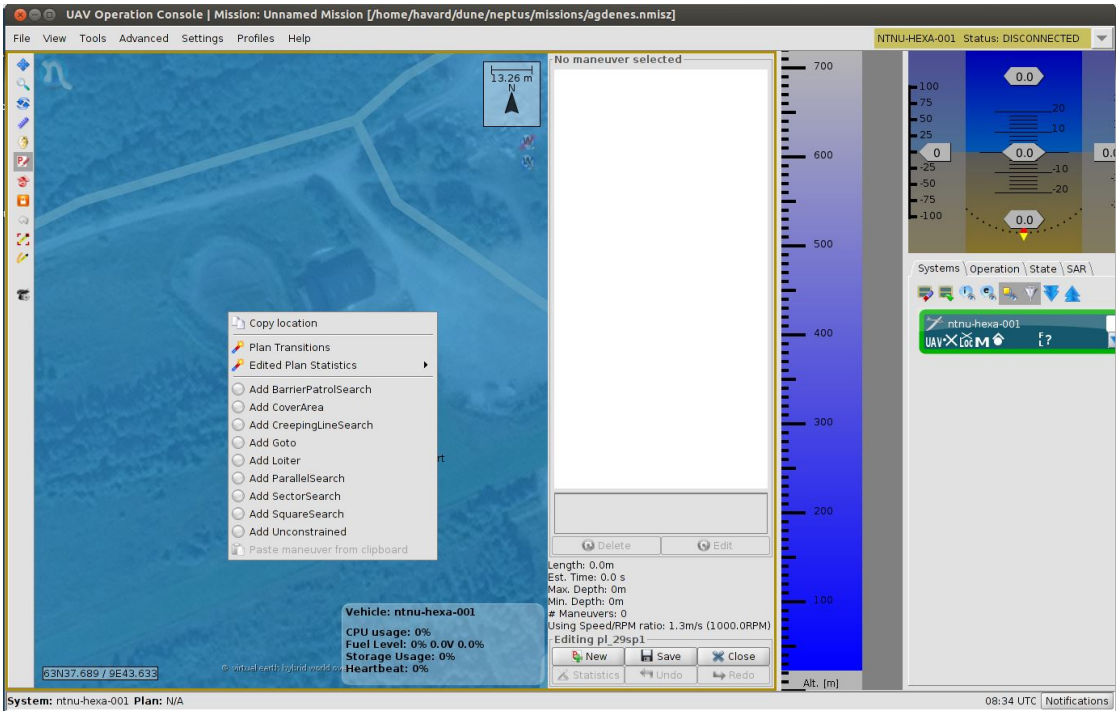


Figure 8.2: Neptus after the *Plan Edition* is opened

file in Neptus, where each applicable maneuver for this vehicle is defined. An example of this is shown in Listing 8.3. In this file, the different properties can also be initialized. A right click on the map in the screen lets the operator add one of these maneuvers; the applicable maneuvers for the vehicle *ntnu-hexa-001* are shown in the popup window in Figure 8.2

Listing 8.3: An example of a maneuver defined in a vehicle definitions-file

```
<feasibleManeuvers>
    <!-- Other feasible maneuvers -->
    <maneuver>
        <SectorSearch kind="automatic">
            <basePoint type="pointType">
                <point>
                    <id/>
                    <coordinate>
                <latitude>0N</latitude>
                <longitude>0E</longitude>
                <depth>0.0</depth>
                    </coordinate>
                </point>
            <radiusTolerance>0.0</radiusTolerance>
            </basePoint>
```

```
16          <length>200.0</length>
17          <height>0.0</height>
18          <bearing>0.0</bearing>
19          <speed unit="METERS_PS">10.0</speed>
20          <annotation>
21              <documentation>No documentation available</
                    documentation>
22              <implementation-class>pt.lsts.neptus.mp.maneuvers.
                    SectorSearch</implementation-class>
23          </annotation>
24      </SectorSearch>
25    </maneuver>
26    <!-- Other feasible maneuvers -->
27 </feasibleManeuvers>
```

Each maneuver is defined in its own Java-class. When a maneuver is chosen, the waypoints are calculated and presented to the operator using functions from the Java-class. A properties-window is then created, which can be seen on the right side of the screen in Figure 8.3. From there, the operator can change the parameters of the maneuver, such as size of the search area, bearing angle etc. The waypoints are then recalculated and shown on the screen.

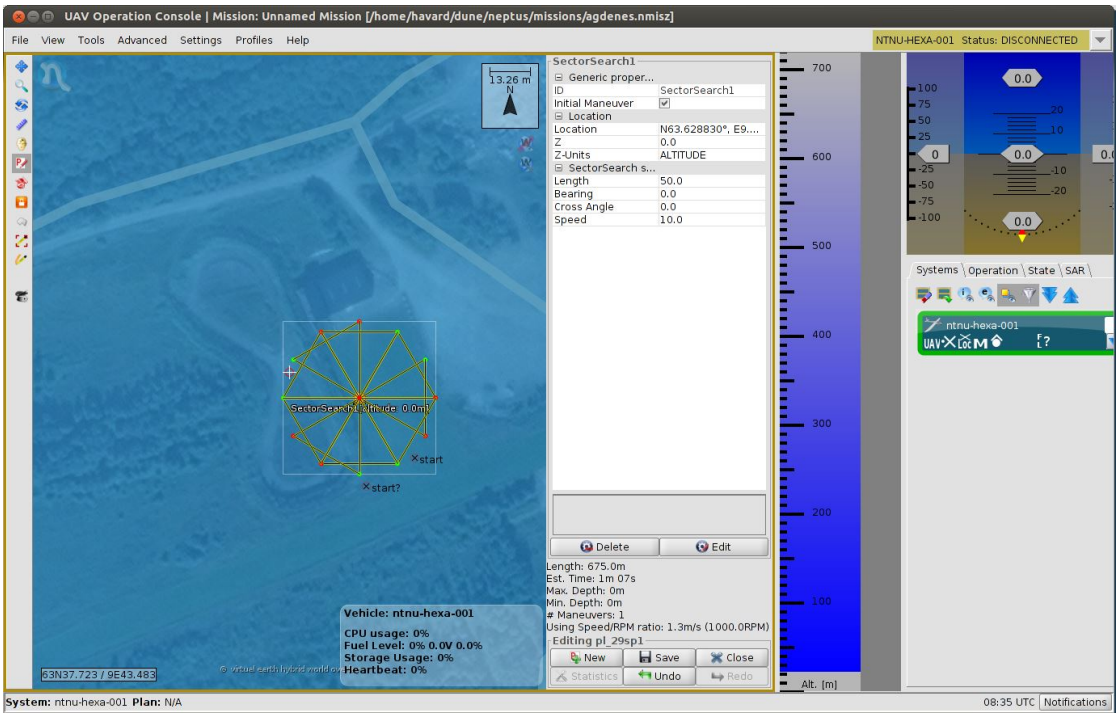


Figure 8.3: Neptus after a maneuver is selected

After the operator has made a satisfactory mission plan, it must be sent as IMC-messages, and thus the Java-class must include functions for writing the properties to the correct IMC-message.

8.3 Dune

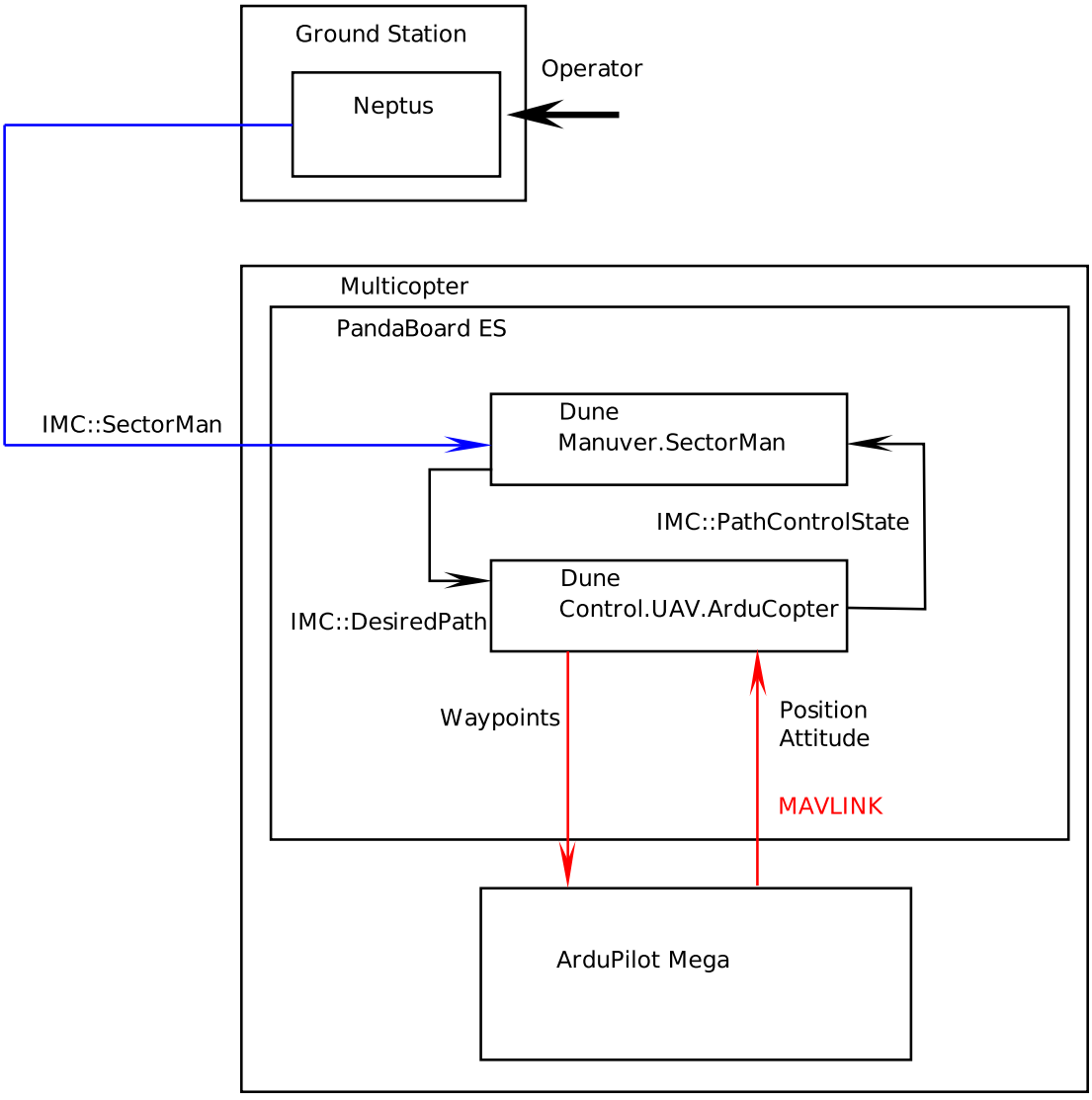


Figure 8.4: The Dune and Neptus signal flow for the Sector Search maneuver.

The signal flow for the sector search maneuver is shown in Figure 8.4, similarly

dune tasks for the square search pattern and the barrier patrol search pattern were made. The figure shows that two Dune-tasks are used when sending a waypoint from the mission created in Neptus to the autopilot (ArduPilot). These are the maneuvers specific task (SectorMan in the drawing) and Control.UAV.ArduCopter. Their function is described in the following sections.

8.3.1 Maneuver.SectorMan

Each maneuver defined in an IMC-message has a corresponding Dune-task that is responsible for handling or consuming that message. For the sector search maneuver, this Dune task is called SectorMan.

This task has two consume-functions; one for the maneuver-message (IMC::SectorMan) and one for the *IMC::PathControlState*-message.

IMC::SectorMan

When receiving the IMC::SectorMan-message, the Dune-task creates the search pattern in the NED-frame based on that message. The pattern is stored as a set of *Stages*, rather than as a set of points. A Stage is defined as the distance between two waypoints, Figure 8.5a.

A simple approach to finding these Stages is to think of the Stage as a vector. Then the Stage is given by

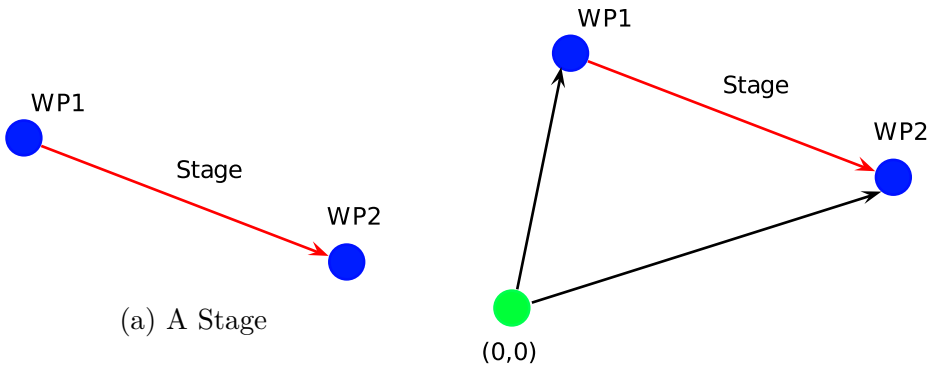
$$Stage = -OWP1 + OWP2 \tag{8.1}$$

where $\vec{OWP}i$ is the vector from the origin to the ith waypoint.

After computation is done, the first waypoint is rotated by the bearing angle, and computed in WGS-84 latitude and longitude. Then an *IMC::DesiredPath*-message is filled and sent.

IMC::PathControlState

The PathControlState-message tells between which waypoints the craft currently is and its along- and cross-track positions. A flag is set when the craft is close to the targeted waypoint. If this flag is active when the message arrives, the Dune-task finds the next Stage, rotates it by the bearing angle, and fills a new DesiredPath-message with the next waypoint in WGS-84 latitude and longitude coordinates.



(a) A Stage

(b) The Stage as a vector

Figure 8.5: A Stage

8.3.2 Control.UAV.ArduCopter

This Dune-task is a modified version of the existing Control.UAV.ArduPlane-task, created by [28]. In this context, it is responsible for consuming the IMC::DesiredPath-message sent by the Maneuver.SectorMan task and to send the waypoint set here via MAVLink to the ArduPilot Mega.

It also gets the multicopter's position and attitude from the ArduPilot and checks if the multicopter is close enough to a waypoint so that a waypoint switch can occur. If close enough, a flag *FL_NEAR* is set in the IMC::DesiredPath-message that is consumed by the Maneuver.SectorMan-task.

8.4 Running Dune on a Vehicle

Each vehicle running Dune has their own configurations file (vehicle.ini). This file specifies which Dune-tasks that will be running for that vehicle, and initializes some fields which values, during operations, can be changed from Neptus. For instance, a multicopter must run the Control.UAV.ArduCopter-task, while other vehicles like an AUV does not run that task.

A part of the configurations file for the hexacopter described in Chapter 3 is found in Listing 8.4. This part shows the part where the Control.UAV.ArduCopter task is initialized. As seen in the listing, the task is defined for three profiles; *Hardware SimulationRC* and *Simulation*, and the task is initialized differently for different profiles. Simulation or SimulationRC is used when simulating with or without an external radio controller. Hardware is used for tests flights.

Listing 8.4: A part of the hexacopter's configurations file

```
1  [Control.UAV.ArduCopter/Hardware]
2  Enabled                                  = Hardware
3  Entity Label                             = Autopilot
4  Ardupilot Tracker                        = True
5  TCP - Address                            = 127.0.0.1
6  TCP - Port                               = 9999
7  uBlox GPS                                = False
8  Telemetry Rate                           = 10
9  Default altitude                         = 30
10 Default speed                            = 20
11 Ardupilot Velocity BODY                  = True
12
13 [Control.UAV.ArduCopter/Simulation]
14 Enabled                                  = Simulation,
       SimulationRC
15 Entity Label                             = Autopilot
16 Ardupilot Tracker                        = True
17 TCP - Address                            = 127.0.0.1
18 TCP - Port                               = 5760
19 Debug Level                              = Debug
20 Default altitude                         = 30
21 Default speed                            = 20
22 Auto Custom Maneuver                     = True
23 Auto Custom Maneuver - Name              = My custom maneuver
24 Auto Custom Maneuver - Plan Name         = Quick plan
25 Auto Custom Maneuver - Parameter        = Some=parameter
26 Ardupilot Velocity BODY                  = True
```

For space saving, the vehicle.ini-file can include other .ini-files, such as the maneuvers.ini-file which enables all the maneuvers. The part of the maneuvers.ini file enabling the five search patterns implemented in this chapter is shown in Listing 8.5.

Listing 8.5: A part of maneuvers.ini

```
1  [Maneuver.Rows]
2  Enabled                                  = Always
3  Entity Label                             = Rows Maneuver
4
5  [Maneuver.SquareMan]
6  Enabled                                  = Always
7  Entity Label                             = Square Search
8
9  [Maneuver.SectorMan]
10 Enabled                                  = Always
11 Entity Label                             = Sector Search
12
13 [Maneuver.BarrierMan]
14 Enabled                                  = Always
```

```
15 Entity Label                    = Barrier Patrol Search
```

8.5 Simulations

The implementation was verified by simulations, using an Arduino SITL (software in the loop) simulator. This simulator is based on a build of the autopilot code using an ordinary C++ compiler, that gives a native executable so that the behaviour of the code without hardware can be tested. A step by step guide for setting up a SITL simulator on Linux is found in [1].

This simulator can be set to interact with Dune and Neptus, so that Neptus can be used to send commands and mission plans to the simulated craft. Figure 8.6 shows Neptus during a simulation of a square search. To the bottom right, a list of the available mission plans can be seen. The plan is marked in either white, green, red or purple, where green indicates in synch in both the vehicle and Neptus. The plan is sent from Neptus to Dune by pressing the blue arrow above the list. The green circle is pressed for executing the plan and the red circle is pressed for aborting a plan in progress. To the top right, the vehicle's attitude and altitude are presented. The craft's position is shown on the map by a green shape. The position of the simulated craft is sent by the IMC::EstimatedState-message from Dune.

8.5.1 Results and Discussion

The results of the simulations are shown in figures 8.7 to 8.11. These plots are shown in the NED-frame, as this frame will make more sense than GPS-measurements when reading the plots. When simulating, the waypoint switch criteria used was: switch waypoint if the distance to target reported from the autopilot is less than 10 meters.

The plots from the simulations of parallel search, Figure 8.7, and creeping line search, Figure 8.8, show that the simulated craft follows the path well. Some cross-track error is experienced along the short lines when turning, and some constant cross-track error along some of the long lines. Even though Dune is set to signal a waypoints switch 10 meters before reaching the waypoints, it can be seen that the simulated craft travels nearly all the way to the waypoint before switching for the next. This might be because the desired speed of the simulated craft is set relatively high ($u_d = 10$ m/s), and the estimated speed u varies from 2 to 5 m/s,

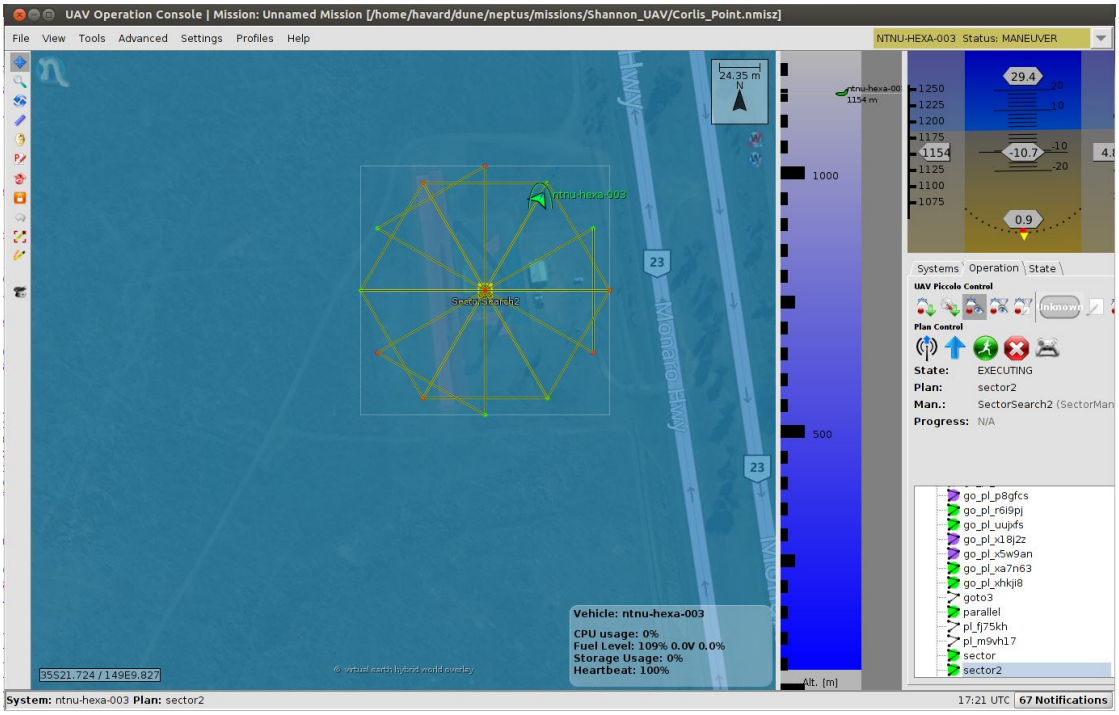


Figure 8.6: Neptus during the flight simulation of the sector search pattern

which makes the craft move some meters before the next waypoint has reached the autopilot.

The simulation of the square search is shown in Figure 8.9. The simulated craft experiences some cross-track error in the beginning of the simulation, due to short distance between waypoints and thus faster waypoint switching. After this, it sticks closer to the path, with the exception of a the small sections after each turn.

Figure 8.10, shows the results of the simulation of the sector search, which is also the pattern being simulated in the screenshot in Figure 8.6. This simulation looks a bit worse than the others, as there are some minor cross-track error during the whole simulation. It also experiences two bad turns, one at the top of the pattern and one down to the right. The scale of the axes in this plot are smaller than the others, and thus the cross-track error might not be as large, relative to the others, as it appears to be.

The simulation of the barrier patrol search is shown in Figure 8.11. Here, the path following is very good, and little cross track error is experienced along the path.

These simulations proved the functionality of the scheme illustrated in Figure 8.4, with the implemented maneuvers and their Dune-tasks. Both the communication

between Neptus and Dune by IMC-messages, and the communication between Dune and the autopilot by MAVLink was found to be working successfully.

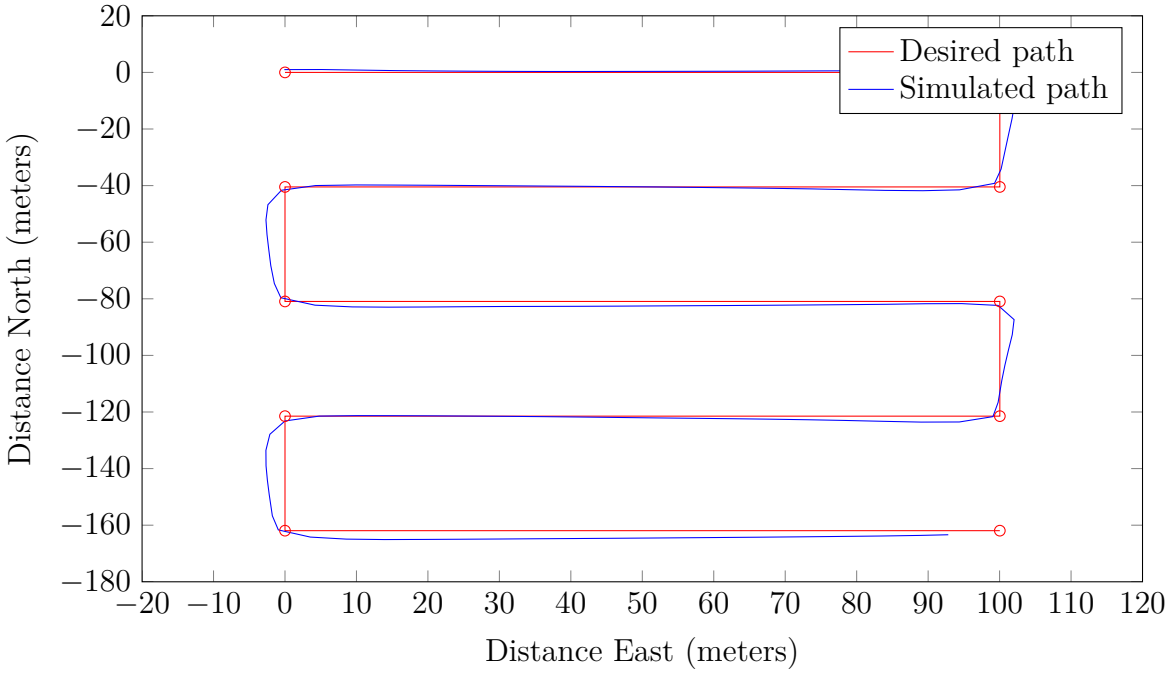


Figure 8.7: Simulation of the parallel search

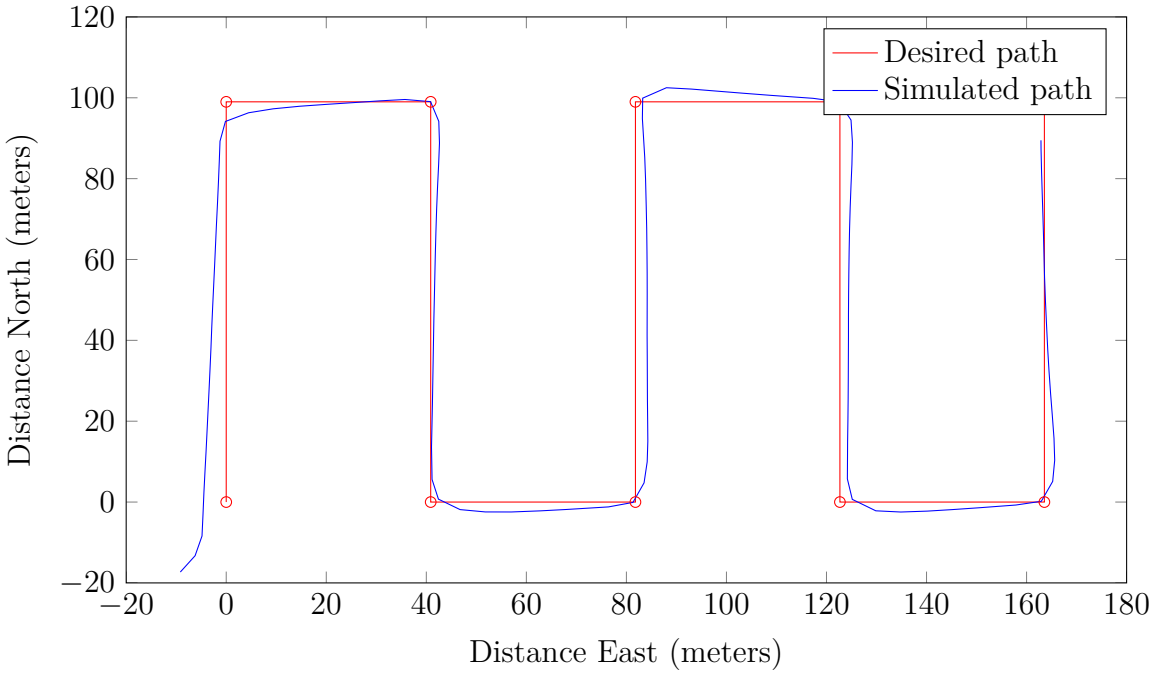


Figure 8.8: Simulation of the creeping line search

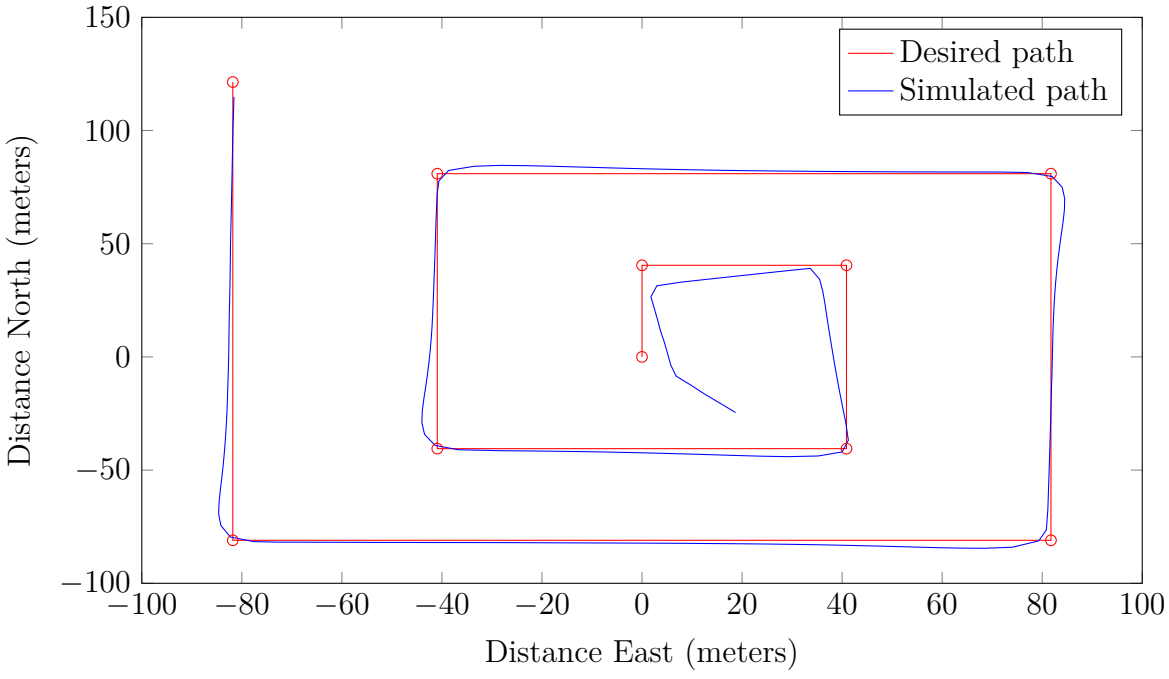


Figure 8.9: Simulation of the square search

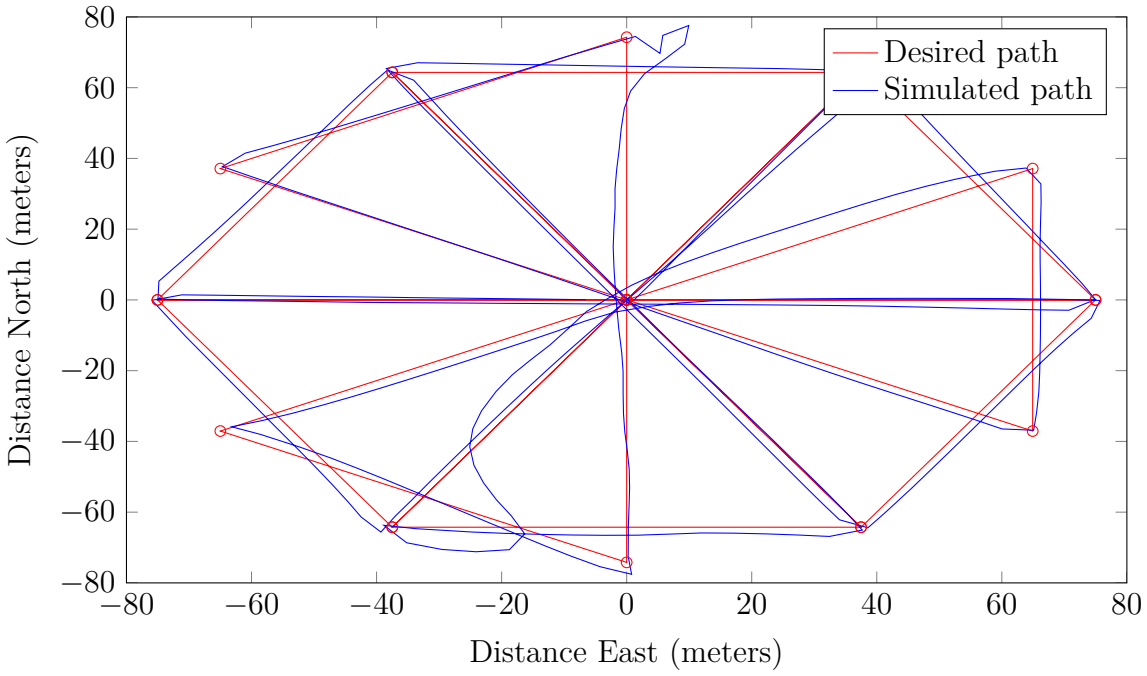


Figure 8.10: Simulation of the sector search

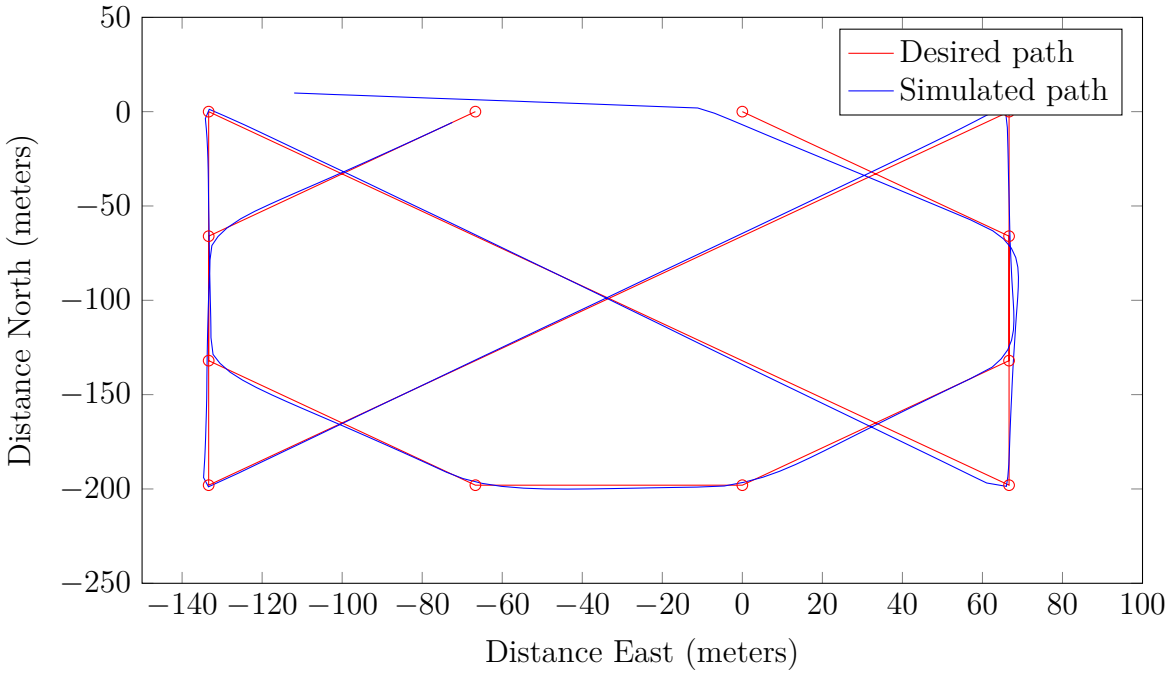


Figure 8.11: Simulation of the barrier patrol search

Chapter 9

Flight Tests

This chapter describes the flight tests conducted in this thesis, and the results gotten. The tests were conducted at Agdenes airport in Breivika May 20th 2014. Tests conducted earlier gave no results due to hardware problems.

To increase flight time, these tests were conducted using only a single-board computer as payload. The onboard computer used during the flights was a BeagleBone, due to previous problems with the PandaBoard.

To be easier to understand for a reader, the plots in this chapter have been translated from GPS-measurements to NED-coordinates using the method described in Apppendix A.2. The first waypoint is chosen as the reference point, and it therefore becomes the origin of the NED-frame.

9.1 Test of existing Goto-maneuver

In the first experiment, the existing Goto-maneuver was tested. The Goto-maneuver is simply setting a simple waypoint that the multicopter should move to. The mission consisted of three Goto's forming an L. The mission in Neptus is shown in Figure 9.1.

The results from this test is shown in Figure 9.2. Here, the hexacopter's path is shown in blue, and the path defined by the three waypoints is shown in red. The hexacopter stears towards the waypoint, when approaching the condition set in the Control.UAV.ArduCopter-task kicks in and the FL_NEAR flag is set, and sent in an IMC::PathControlState-message. Then a new waypoint is received in an IMC::DesiredPath-message, and sent to the ArduPilot Mega by MAVLink.

Figure 9.1: The Goto-plan from Neptus on a map of Agdenes.

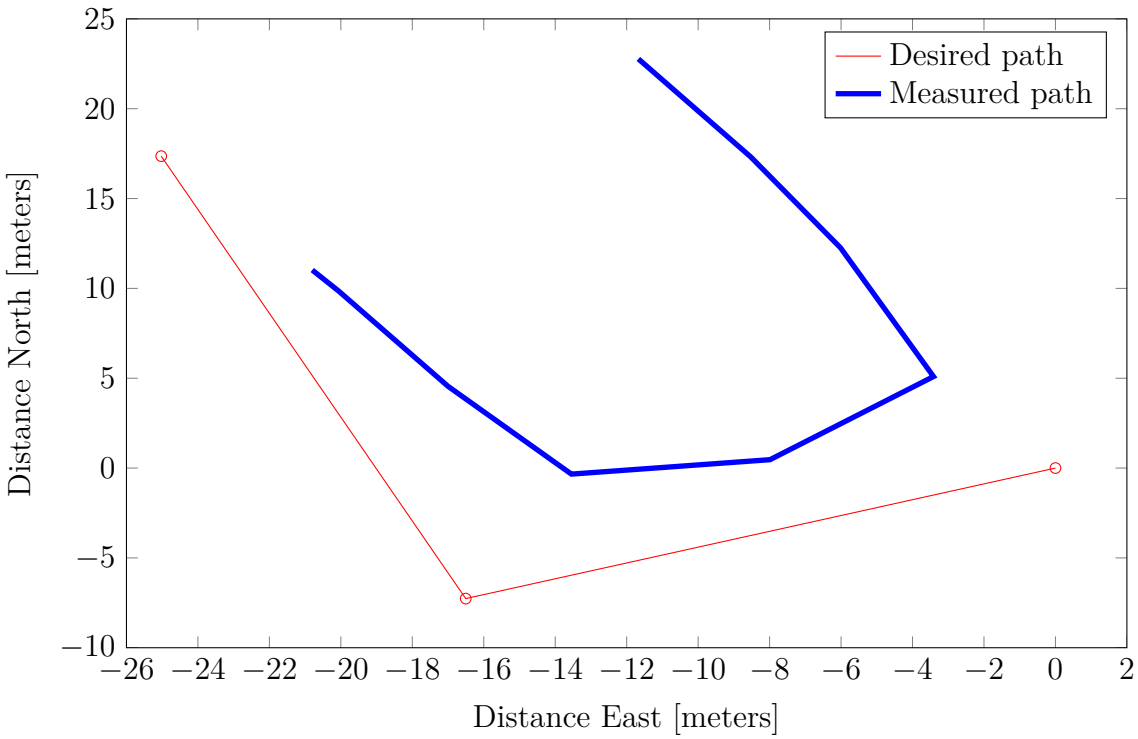


Figure 9.2: The result from the test of the Goto-maneuver.

This test proved the functionality of the mission planning scheme, the Goto-maneuver and the waypoint switching. However Figure 9.2, shows that the condition for waypoint switching can be tightened, so that the hexacopter flies nearer the waypoint before making a switch, which will give a path closer to the red lines.

9.2 Parallel Search

The first maneuver tested was the parallel search pattern. As the parallel search pattern and the creeping line search pattern use the same IMC-message and Dune-task, only one of them were tested. The mission plan for this test is shown in Figure 9.3. The flying altitude was set to 10m, and the dimensions of the search area were 30 m $\times$ 30 m. The criteria for switching waypoints was that the reported distance to target from the ArduPilot Mega was less than 5 meters.

Figure 9.3: The parallel search pattern tested at Agdenes.

Figure 9.4, shows the results from this test. Here, the hexacopter follows the desired path for some time, and everything appears to work fine. However, when approaching the 5th waypoint, it stops in the waypoint, and enters Loiter mode.

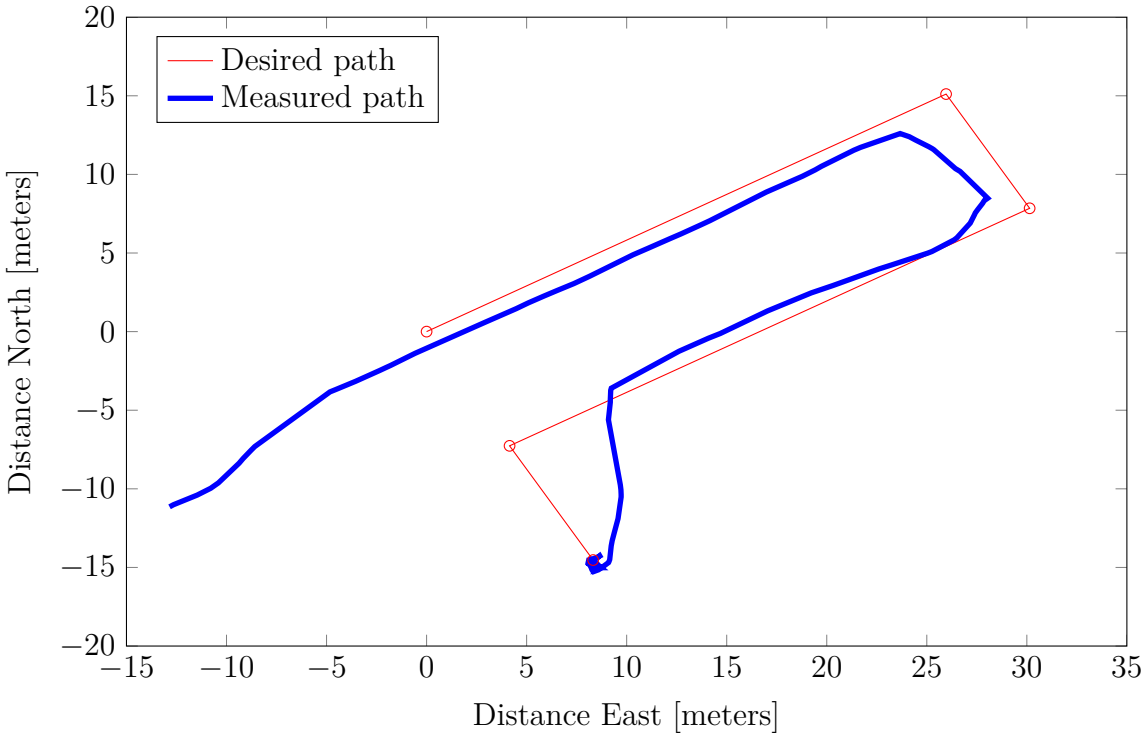


Figure 9.4: The result from flying a parallel search pattern.

9.3 Square Search

Test 1

The next maneuver tested was the square search pattern. The pattern was set to cover a 20m × 20m area, which resulted in 6 waypoints. The pattern is shown in Figure 9.5.

Figure 9.5: The square search pattern tested at Agdenes.

How the hexacopter follwed the path can be seen in Figure 9.6. The hexacopter starts to the right of the pattern; from there it travels the shortest route to the middle of the expanding square, where the first waypoint is. After this, it sticks close to the red line, for the rest of the pattern. It is also worth noting that for this test, the waypoint switch criteria was tightened, so that the hexacopter flew closer to the waypoints before heading for the next waypoint.

Test 2

As the first test was successful, a new test was executed with a larger square search maneuver. This time, the pattern was set to cover a 30m × 30m area. This pattern now gave 10 waypoints, as can be seen from Figure 9.7.

The hexacopter's path is shown in Figure 9.8. The hexacopter follows the path relatively well, with some exceptions like the final line (to the left in the plot).

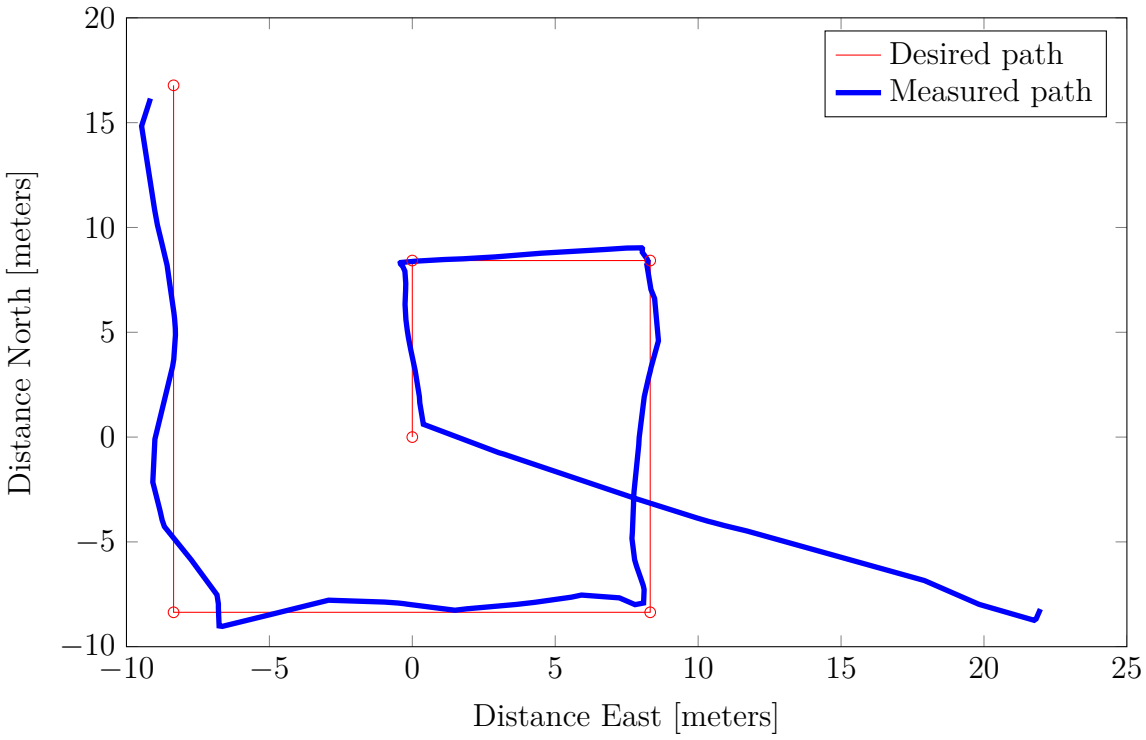


Figure 9.6: The result from flying the first square search pattern.

Figure 9.7: The second square search pattern tested at Agdenes.

This might be due to some wind during this test. The first line in the middle is also not good, because of the hexacopter's starting position. The starting position was almost in waypoint 2, causing the hexacopter to be ordered a 180° turn when reaching waypoint 1.

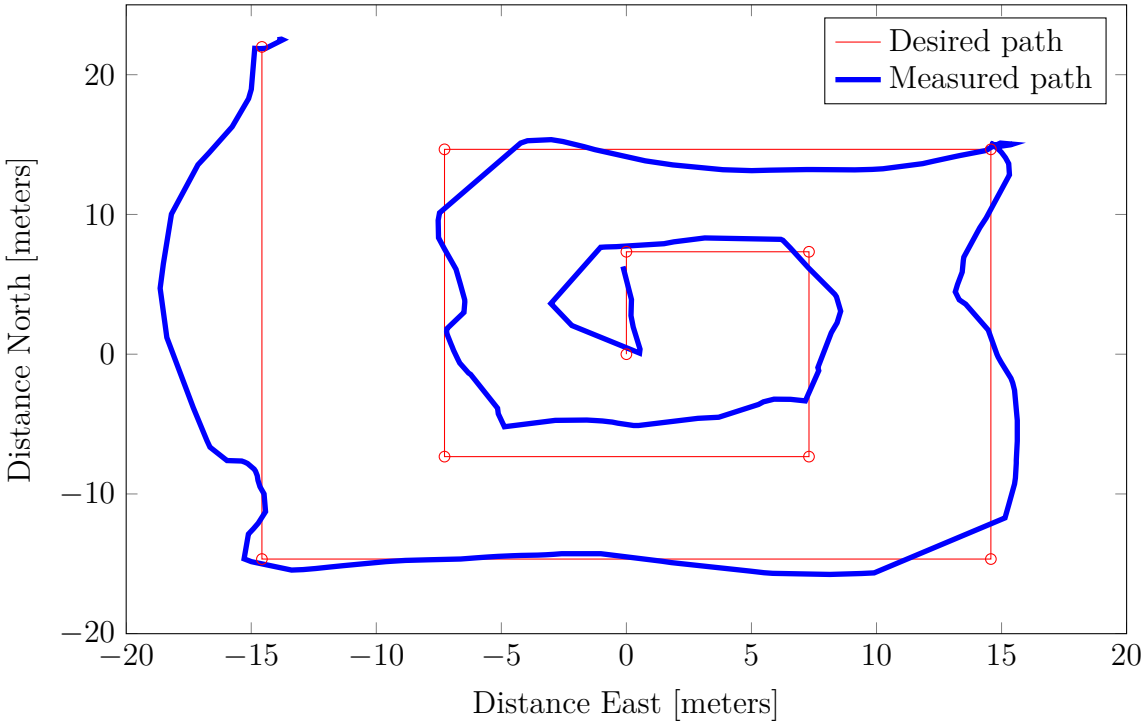


Figure 9.8: The result from flying the second square search pattern.

9.4 Sector Search

The plan for the test of the sector search is shown in Figure 9.9. The circle the sector search pattern was set to search within, had a radius of 20m. How the hexacopter followed this path is seen in Figure 9.10. The plot shows that the hexacopter follows the path well, however some cross-track error is experienced. A plot of the cross-track error, computed by Equation 2.10, is found in Figure 9.11, where the red circles indicate a waypoint switch. Here, it is shown that the hexacopter get some crosstrack error all over, and the cross track error is smallest around the time when a waypoint switch occurs. If the hexacopter followed the path perfectly, the cross-track error would have been zero along the lines, and showed an increase when the waypoint switch took place.

Figure 9.9: The sector search pattern tested at Agdenes.

A plot of the hexacopter's heading versus desired heading is shown in Figure 9.14. Here, it can be seen that the hexacopter's heading controller works fine. There are some minor deviations when trying to keep a constant heading, and a small overshoot. On the plot, two large overshoots appear to be present when moving from a large angle like 270° to a small angle of 30°. These are however not overshoots, but rather a sign that the hexacopter has turned left in stead of right. Due to low battery on the hexacopter, the search had to be aborted after returning to middle when it had completed 8 waypoints, or 3 triangles.

9.5 Barrier Patrol Search

The last search pattern tested was the barrier patrol search pattern, as can be seen in Figure 9.15. This search pattern is mainly used to detect when a subject within an area leaves the area, and therefore it sticks to the sides of the area. Due to space limitations on Agdenes, the area had to be small, and the dimensions used for this test were 35m × 35m.

When flying the barrier patrol search pattern, the same problem as described in Section 9.2 occurred. The best result from many tries is shown in Figure 9.16. This time, the hexacopter flew 7 waypoints, before it stopped and went into Loiter-mode.

In this test, the Dune-task was set to give a new waypoint, when the reported distance to target (from the autopilot) were less than 3 meters. Since the path is

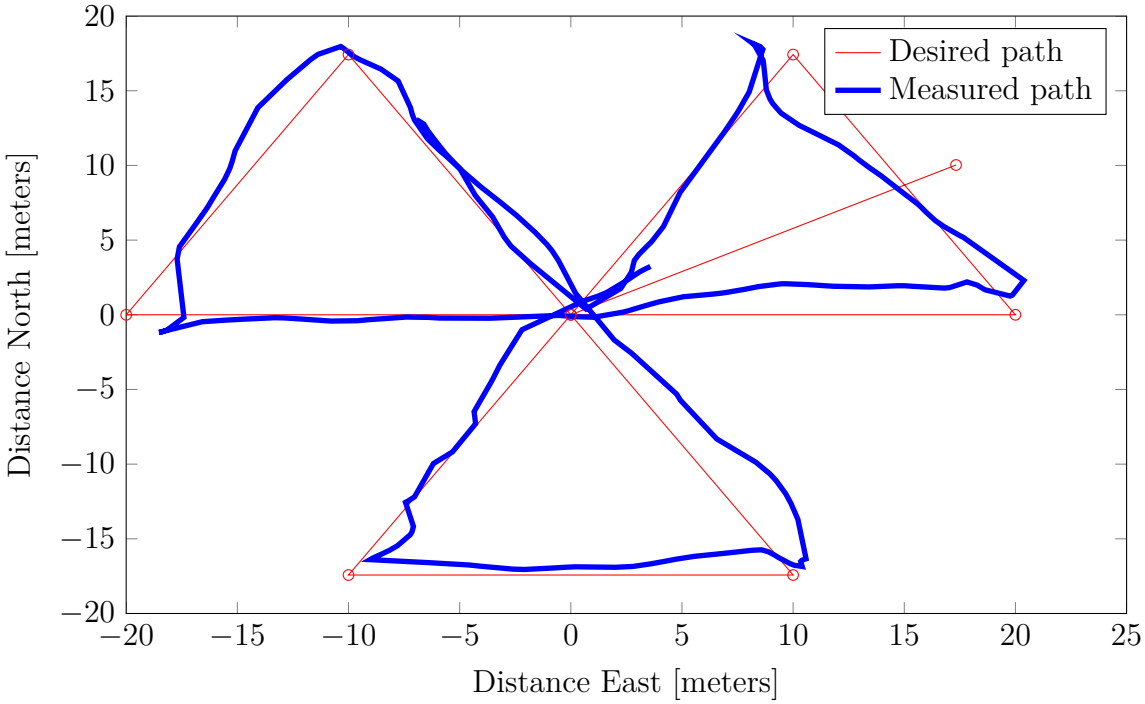


Figure 9.10: The result from flying the sector search pattern.

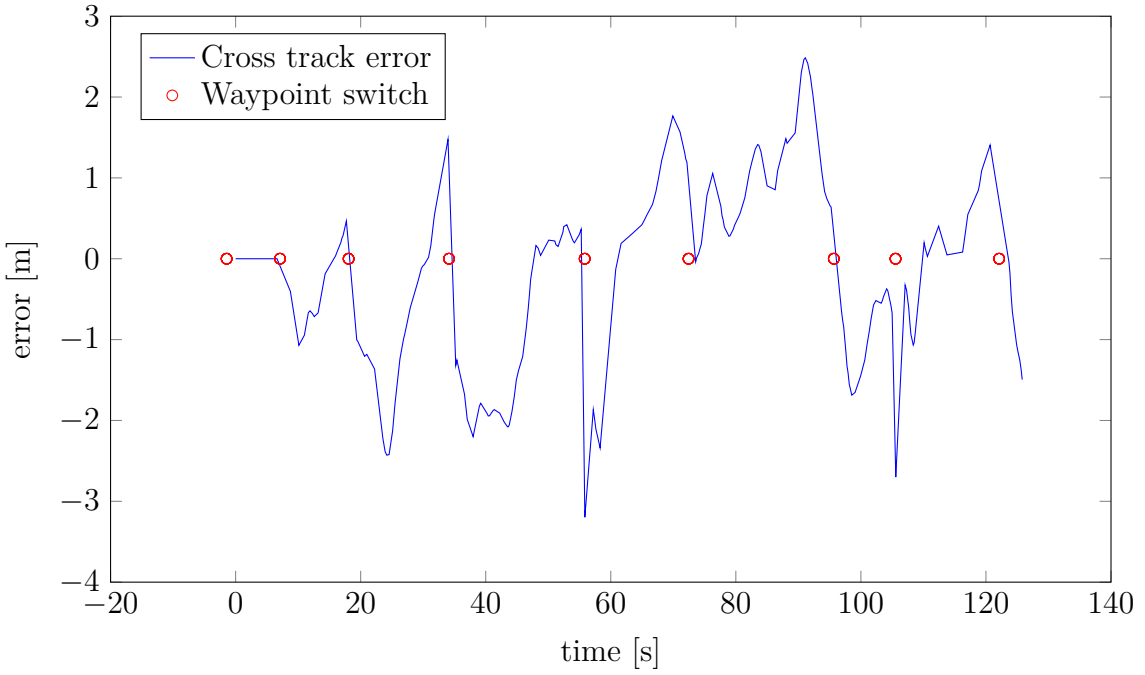


Figure 9.11: The cross-track error when flying the sector search pattern.

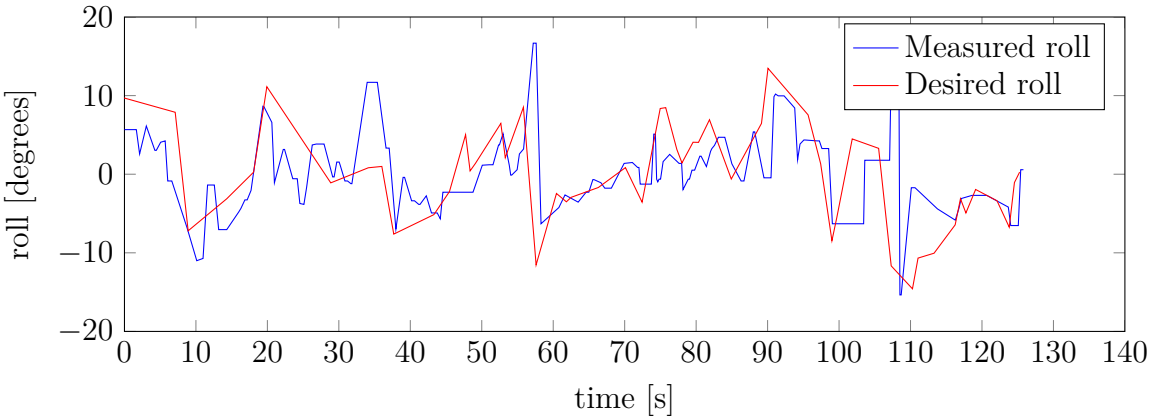


Figure 9.12: Desired roll versus measured roll.

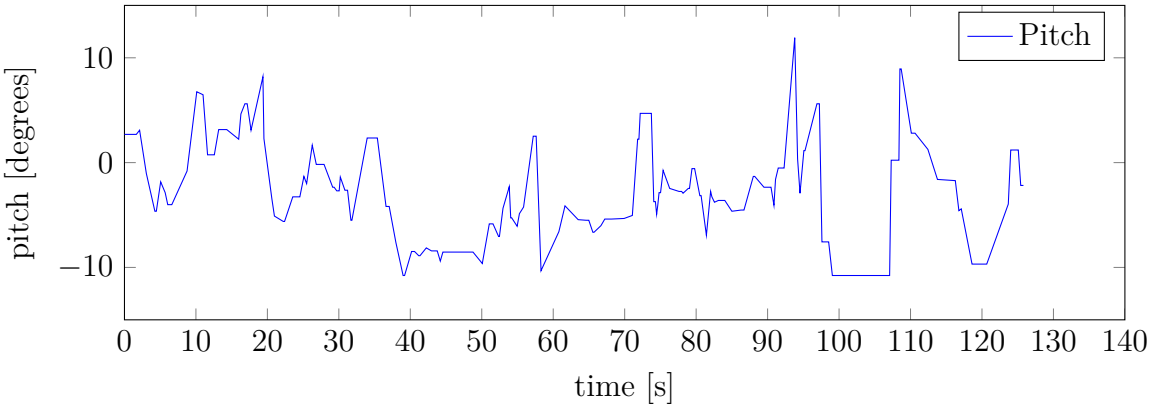


Figure 9.13: Measured pitch.

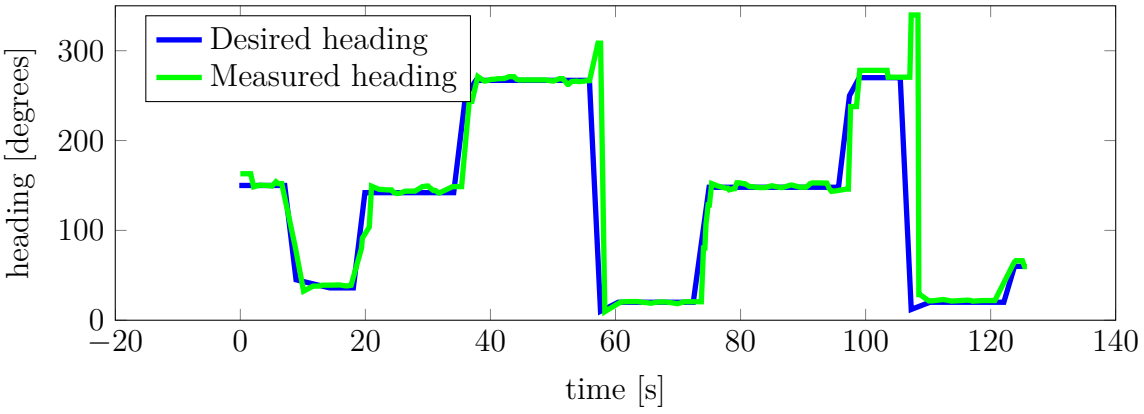


Figure 9.14: Desired heading versus measured heading.

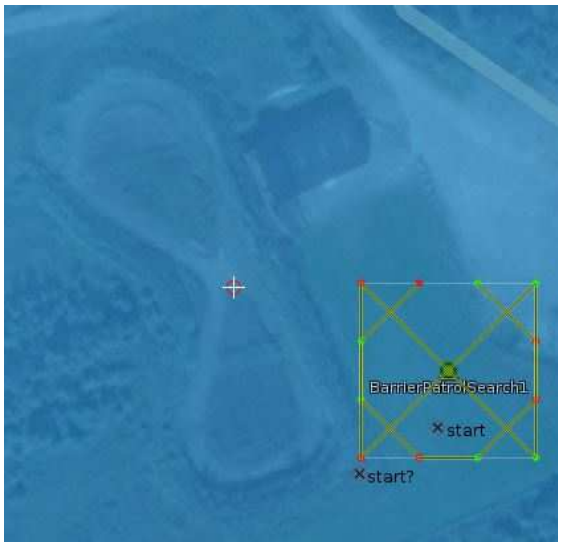


Figure 9.15: The barrier patrol search pattern tested at Agdenes.

plotted in the NED-frame, a circle with radius of 3 meters can be drawn around each waypoint, Figure 9.16. Here, it is clear that the hexacopter is well within this limit on the last waypoint it encounters, and then a new waypoint should have been sent to the autopilot.

9.6 Mission Review

After the flights, Neptus' Mission Review and Analysis-tool can be used to review and analyse the flights. A screenshot from the review of the parallel search in Section 9.2 is shown in Figure 9.17. This screenshot shows the planned path consisting of waypoints and lines between them. The performed path is shown in white dots. In this screen, the performed flight can be replayed by pressing the play-button in the top of the screen.

When Dune is running during a mission, IMC-messages such as *EstimatedState*, *DesiredPath* and *PathControlState* are logged and stored for later use. Choosing the Messages-tab to the left in Figure 9.17 lets the operator access these messages.

During the tests, a problem that frequently occurred was the hexacopter stopping in one of the waypoints in the middle of the plan, and entering Loiter-mode. Examples of tests where this took place are found in sections 9.2 and 9.5.

When flying, the movement of the hexacopter is controlled by the autopilot which

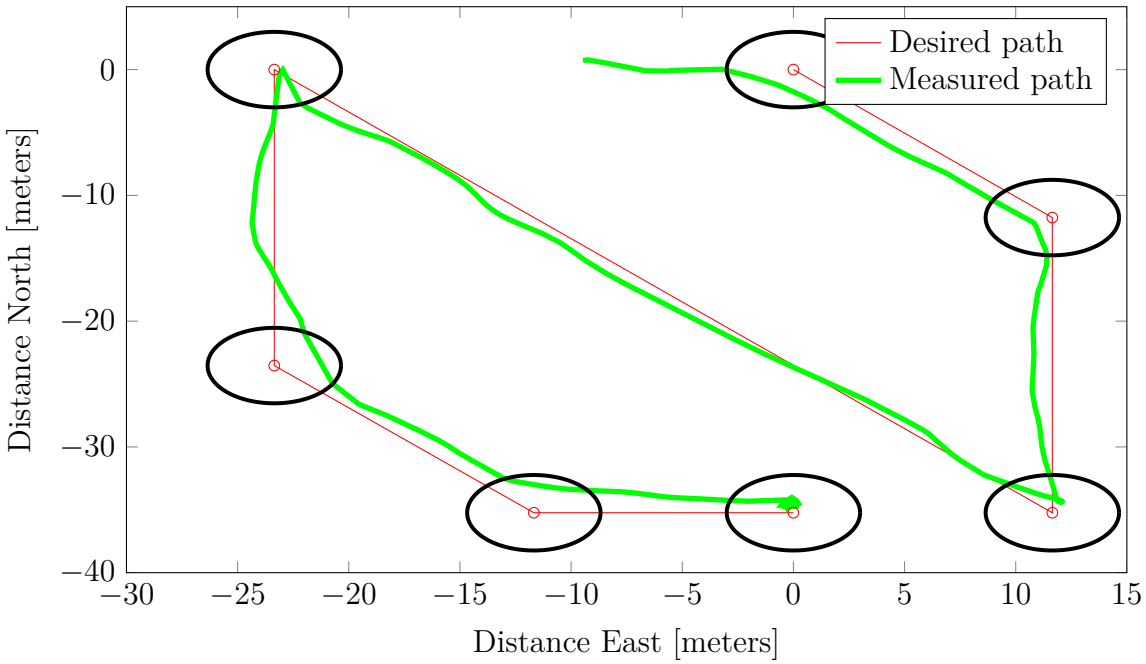


Figure 9.16: The result from flying the barrier patrol search pattern.

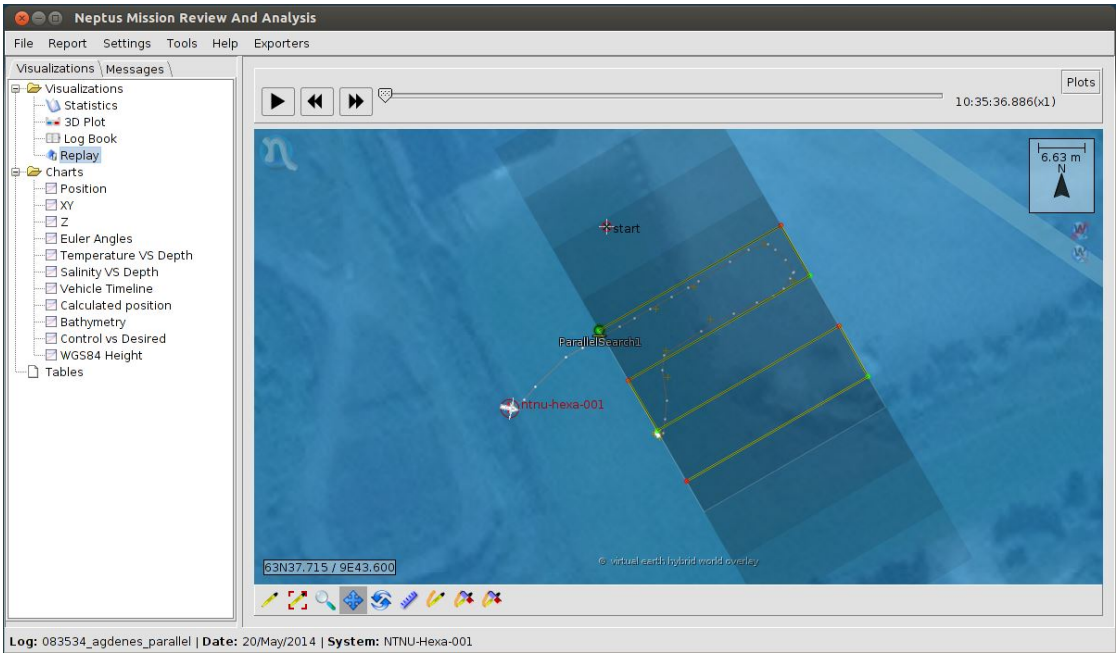


Figure 9.17: Neptus' Mission Review and Analysis-tool

receives the waypoints from the Dune-tasks running, as shown in Figure 8.4. If the autopilot get position measurements close enough to its current targeted waypoint, it enters Loiter-mode.

Outside the autopilot, the Dune-task Control.UAV.ArduCopter is responsible for detecting that the hexacopter is close to a waypoint and set the FL_NEAR flag in the IMC::PathControlState-message. When this flag is set, the maneuver task of the current maneuver will send the next waypoint in return with an IMC::DesiredPath-message.

As both these IMC-messages are logged by Dune, their logs could be examined. Examining the PathControlState-log from these plans shows that when these stops occurred, the FL_NEAR flag in the PathControlState-message was not set by the Control.UAV.ArduCopter-task. Then, no new waypoint is sent by the maneuver-task, and thus the autopilot stays in Loter-mode until some other command is received.

As this flag was not set, the problem appears to be the condition in the Control. UAV.ArduCopter-task detecting closeness to a waypoint, and setting this flag. This was considered an uncertainty before the testing, due to previous flights done by [28]. Therefore, two different conditions were implemented and tested during the flights. These are shown in Listing 9.1. The value of the boolean variable *m_args.wptmethod*, deciding which condition to use, can be set from Neptus and sent by IMC during flight. The same goes for the variables *m_args.wptolerance* and *m_args.secs*.

Listing 9.1: Conditions for waypoint switch

```
1  if (m_args.wptmethod)
2  {
3      if (!m_changing_wp
4          && (nav_out.wp_dist <= m_args.wptolerance)
5          && (m_mode == CP_MODE_GUIDED))
6      {
7          if (!(m_cloops & IMC::CL_GUIDANCE)) {
8              m_pcs.flags |= PathControlState::FL_NEAR;
9              trace("FL_NEAR");
10         }
11     }
12 }
13 else
14 {
15     if (!m_changing_wp
16         && (nav_out.wp_dist <= m_desired_radius + m_args.secs *
               m_gnd_speed)
17         && (nav_out.wp_dist >= m_desired_radius - m_args.secs *
               m_gnd_speed)
```

```
18          && (m_mode == CP_MODE_GUIDED))
19      {
20          if (!(m_cloops & IMC::CL_GUIDANCE)) {
21              m_pcs.flags |= PathControlState::FL_NEAR;
22              trace("FL_NEAR");
23          }
24      }
25  }
26
27  dispatch(m_pcs);
```

As seen in Listing 9.1, both conditions depend on the field *nav_out.wp_dist* which is the autopilot's reported distance to the target waypoint. Its value is not logged by Dune, and therefore it is not possible to verify that it gives the correct values.

When using the Arduino software in the loop simulator in Chapter 8, using the same criteria for waypoint switching, this problem did not exist. Therefore, a theory could be that when running on less powerful hardware, as the ArduPilot Mega is less powerful than the computer used for simulations, the autopilot must prioritize its efforts, and this field is not updated as frequently as it should.

To fix this problem, an implementation independent of this field should be tested. This is further described in Section 10.1.1.

9.7 Discussion

The six flight tests described in this chapter show that the implemented mission planning scheme from Figure 8.4 works, and that Neptus and Dune can be used to control the movement of a real flying hexacopter. With the exception of the problem with switching waypoints discussed in the previous section, the flight tests were considered a success.

The first test conducted was a test of the existing Goto-maneuver. The hexacopter was commanded to visit three waypoints placed from Neptus. This test proved the functionality of the existing mission planning in Neptus, and its communication with the Dune-tasks.

In Figure 9.2, the path of the hexacopter when following the Goto-plan is shown. It can be seen that it does not follow the path closely; in fact its path does not intersect with the desired path at all. This is because the distances between the waypoints are short, and it therefore does not have the time to reach the line before a waypoint switch is commanded.

The tests of the parallel search pattern did not provide any noteworthy results; the best result is found in Figure 9.4. In this test, the hexacopter followed the path well for the first four waypoints, before it stopped in the fifth waypoint, and never received the next waypoint. From the plot, it can be seen that there was, in this test too, some difference between the path of the hexacopter and the desired path.

The square search pattern was the subject for two tests, as the first test became a success. The first test is shown in Figure 9.6. In this plot, the hexacopter follows the desired path closer than in the two previous mentioned tests for the first lines, and then the course becomes more unstable on the last two lines.

The second test, a square search pattern with more waypoints was subject to some windy conditions and therefore some cross-track error is experienced. The result from this test is seen in Figure 9.8. This is especially clear on the last line, to the left on the plot. The choice of starting point was also not optimal, as this resulted in a sudden 180°-turn in the beginning of the flight.

The test of the sector search pattern gave the plot shown in Figure 9.10. Due to low battery time left on the hexacopter, this test had to be aborted after completing one third of the path. The cross-track error appears to be large in the plot. However, when this is plotted in Figure 9.11 it can be seen that it varies between -3 and 3 meters, which are acceptable values.

The attitude, roll pitch and yaw, of the hexacopter is shown in figures 9.12, 9.13 and 9.14. The first plot shows that the roll of the hexacopter changes frequently, and varies from around $-15°$ to $15°$. The pitch angle from the second plot shows that the pitch angle varies from $-10°$ to $10°$. Ideally, the value of the roll angle should be zero when the hexacopter is following the straight lines, and the pitch angle should have a small value less than zero, to keep a constant speed along the straight lines. However, as the tests were done in an outdoor environment subject to wind and disturbances, some variations in these values are expected.

The plot of the desired versus measured heading shows that the heading controller works fine. There are some minor variations in heading when the desired heading is constant, and the measured heading is sometimes a bit behind the desired. This is expected as the controller can not produce unlimited angular rate.

Figure 9.16 shows the measured path when the barrier patrol search pattern was tested. This test was also subject to the waypoint-switching problem, and therefore it stopped before completing the pattern. Before stopping, the path looks good, and it sticks close to the desired path, especially along the long diagonal line.

This plot includes circles with radius of 3 meters around the waypoints, as the

criteria used for this test was switch waypoint when the distance to the target waypoint is less than 3 meters. The plot shows that position measurements well inside this circle are received, also for the last waypoint where the test stopped.

Chapter 10

Conclusion

In the present thesis, the main objective was to study different paths or trajectories that can be used when executing a Search and Rescue mission with a multicopter and discuss how the onboard camera should be placed to ensure the entire area of interest is covered.

Two different methods for placing the onboard camera were considered. The first method was using a gimbal for roll and pitch stabilization. This gimbal would be set so that the camera was pointing forward with an angle downwards. The other method was attaching the camera on the bottom of the multicopter without a gimbal, thus making the camera point straight down and sideways and backwards as the multicopter experienced roll and pitch. The method using the gimbal was ultimately chosen, as this method was the one that would give the best images and with its angle it would give images that were more suitable for the object detection and tracking algorithm.

In Chapter 4, a method for computing which area of the map the onboard camera covers with a single frame is outlined. This method uses the ideal pinhole camera model and creates a projected image plane between the ground and the camera. The corner points of the image plane are then found in the camera frame, and transformed into NED-coordinates. Using the origin of the camera frame, and each corner point, a parametric equation is created. Each corner point's coordinate on ground level is then found by setting the z-component of the parametric equation to zero. This method supports both the methods for placing the camera, as the only change will be the transformation matrix between the camera-frame and the NED-frame.

The method was implemented into the ground-station software Netups, to give the operator a live graphical overview of which parts of the search area the multciopter

has covered.

Five different search patterns were chosen, and algorithms for creating waypoints for these were developed, in Chapter 5. The properties of the search patterns were tested through simulations in Chapter 7 where the search patterns were tested in scenarios with search for stationary subjects and moving subjects. An estimate of the search patterns' area coverage with the decided camera placement was also found through these simulations.

The best performer in the simulations was the square search pattern. It scored the highest on both the search for stationary and moving subjects, and its area coverage was estimated to 96% of the search area. Some coverage were lost in the middle of the search area, mainly due to short lines with rapid turns and that the multicopter started in the first waypoint with the simulation not saying anything about how the multicopter reached that point.

The parallel and creeping line search patterns also scored good on finding stationary subjects. Both had an estimated area coverage of 93%, loosing some coverage when turning, due to the camera pointing so far ahead of the craft. When searching for the moving subjects, parallel scored better than creeping line with parallel finding 3 of 4 and creeping line finding only 1 of 4. This difference is because the paths of the subjects intersects more frequently with the parallel search at an early stage of the search. If the search area had smaller dimensions, the patterns would have shorter search lines and then performed better when searching for moving subjects.

In the stationary subjects-simulation, the sector search finished slightly worse than the three mentioned search patterns finding 17 of 20 subjects. In the moving subjects-simulation however, it scored among the best with finding 3 of the 4 subjects. When searching for moving subjects, the sector search pattern has the advantage that it moves quickly from one side of the search area to the other, as opposed to the parallel and creeping line search patterns that use long time time covering the search area thoroughly. This leads to the sector search pattern having larger possibility of intersecting with the path of a moving subject, but reduces area coverage. The area coverage was estimated to around 73%.

The barrier patrol search pattern had the worst performance in all of the simulations. It only detected 5 of the 20 stationary subjects, and the area coverage was estimated to 45%. A bad performance in this simulation was expected from that pattern, as it is designed to find moving subjects that leaves an area, like the moving subjects-simulation. However, the performance in this simulation was not good, and it only detected 1 of the 4 subjects.

The search patterns were implemented as maneuvers in Neptus and Dune, so that the existing mission planning scheme could be used, and plans including both the new maneuvers and existing maneuvers such as the Goto-maneuver could be make. The parallel and creeping line search patterns are variants of an existing maneuver, Rows Maneuver, and could then use this maneuver's Dune-task and IMC-message. The other search patterns needed their own IMC-message and Dune-task. The implementation was verified by simulations using an Arduino software in the loop-simulator where Neptus and the Dune-tasks were used to command the simulated craft to move.

Using the hexacopter described in Chapter 3, flight tests were conducted on Agdenes airport. The results from these tests are shown in Chapter 9. During the tests, a problem with Dune not detecting that the craft was close to the waypoint, and therefore not sending the next waypoint, occurred frequently.

Except from the waypoint switching problem, the flight tests were successful and proved that the mission planning scheme with Dune and Neptus could be used to control a hexacopter, and that the implemented maneuvers worked.

The tests were conducted without most of the payload like the infrared camera. Then, the effective flight time was around five minutes, and thus a limited search area could be covered. For this to be used in a real-life Search and Rescue mission, a multicopter with longer flight time and capability of transporting heavier payload is needed.

10.1 Future Work

10.1.1 Waypoint-switching in Dune

An important task that must be dealt with is to find a better way to detect a waypoint-switch in Dune, as the existing ones gave problems during the flight tests. From Listing 9.1, it can be seen that both waypoint-switching methods tested used were dependent on the variable *nav_ out.wp_ dist* that is sent from the autopilot. This variable is not logged anywhere by Dune, and it is therefore some uncertainty regarding the value of this variable.

The implemented methods for detecting a waypoint-switch worked without problems on the simulator. A theory is that when running on less powerful hardware, as the autopilot is less powerful than a computer, the autopilot struggles to complete all its tasks. Updating the *nav_ out.wp_ dist*-field might be down-prioritized, and

thus not happening as frequently as required. A new implementation independent of this field should therefore be tested.

In Figure 9.16, where the position measurements from flight tests of the barrier patrol search pattern is plotted, it can be seen that position measurements from well within a circle with radius 3 meters around the last waypoint, is received from the autopilot.

Therefore, a waypoint-switching criteria based on the *circle of acceptance* from Equation 2.14 could prove to be a better implemented condition. Another possible criteria is the along-track distance from Equation 2.16. Both of these are found in Section 2.2.1 and [13].

10.1.2 Implementation of Cover Area

In Section 5.3, a method for covering a search area that is not rectangular is shown. This method has not yet been implemented. There exists an IMC-message that can be used to send the vertexes from Neptus to Dune. A Neptus maneuver called CoverArea also exists. This is however work in progress by the developers at LSTS, and is therefore not working at this moment.

If this is finished in a future release, one could write a separate Dune-task to read the IMC-message, and use the existing mission planner to send a CoverArea-maneuver as a part of a mission plan.

Another project [19] has implemented the cover area maneuver by creating a separate Neptus plugin to send the IMC::CoverArea-message from Neptus to Dune without the use of the mission planner. If a separate Dune-task is created this implementation can also be used for this implementation. The disadvantage with using that implementation is that the maneuver can not be combined with other maneuvers such as Goto, or the ones implemented by this thesis, in a mission plan.

Appendix A

Coordinate Transformations

In the path planning schemes developed in this project, the following methods are used for transformation from ellipsoidal coordinates (latitude, longitude and altitude) to coordinates in the navigation frame (p_x,p_y,p_z) and vice versa. These algorithms correspond to the MATLAB function blocks *Flat Earth to LLA* and *LLA to Flat Earth*, found in the Aerospace Blockset [5], [6].

The navigation frame is rotated an angle ψ around the z-axis of a NED-frame with its origin in a reference point, given in ellipsoidal coordinates by $p_{ref} = \begin{bmatrix} \mu_0 & l_0 & h_{ref} \end{bmatrix}^T$.

The transformation from navigational to ellipsoidal coordinates are described in Section A.1, and in Section A.2; the transformation from ellipsoidal to navigational coordinates can be found. Both transformations require two constants: the equatorial radius, $R_{eq} = 6378137$, and the flattening, $f = 1/298.257223563$, of the Earth.

A.1 Navigational to Ellipsoidal Coordinates

As inputs, this algorithm needs the point to be transformed $p = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$, the reference point p_{ref}, and the angle ψ which is the angle in degrees clockwise between the x-axis and North. Note that the reference point is also the origin of the navigation frame.

First, the North-East coordinates are calculated from the navigational coordinates.

$$\begin{bmatrix} N \\ E \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\ psi & \cos\psi \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{A.1}$$

Then, the radius curvature in the prime vertical R_N and the radius of curvature in the meridian R_M are calculated.

$$R_N = \frac{R_{eq}}{\sqrt{1-(2f-f^2)\sin^2\mu_0}} \tag{A.2}$$

$$R_M = R_N \frac{1-(2f-f^2)}{1-(2f-f^2)\sin\mu_0} \tag{A.3}$$

The small changes in latitude and longitude are then approximated from small changes in North and East positions:

$$d\mu = \arctan\left(\frac{1}{R_M}\right) dN \tag{A.4}$$

$$dl = \arctan\left(\frac{1}{R_N \cos\mu}\right) dE \tag{A.5}$$

Since the reference point has coordinates $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ in flat Earth coordinates, $N = dN$ and $E = dE$.
The output latitude and longitude is then given by

$$\mu = \mu_0 + d\mu \tag{A.6}$$

$$l = l_0 + dl \tag{A.7}$$

and the output altitude is given by:

$$h = -p_z + h_{ref} \tag{A.8}$$

A.2 Ellipsoidal to Navigational Coordinates

This algorithm takes as inputs the point to be transformed in ellipsoidal coordinates $p = \begin{bmatrix} \mu & l & h \end{bmatrix}^T$, and as above, the reference point p_{ref}, and the angle ψ.

After the calculation of R_N and R_M, the next step is to find the small changes in latitude and longitude:

$$d\mu = \mu - \mu_0 \tag{A.9}$$

$$dl = l - l_0 \tag{A.10}$$

The small changes in the North and East positions are then approximated by:

$$dN = \frac{d\mu}{\arctan\left(\frac{1}{R_M}\right)} \tag{A.11}$$

$$dE = \frac{dl}{\arctan\left(\frac{1}{R_N \cos\mu_0}\right)} \tag{A.12}$$

The output point is then transformed from North East coordinates by:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} N \\ E \end{bmatrix} \tag{A.13}$$

and the return height:

$$p_z = -h + h_{ref} \tag{A.14}$$

Appendix B

Cover Area Examples

This Appendix shows some examples of the shortest lawnmower pattern created by the cover area method outlined in Section 5.3, on different areas.

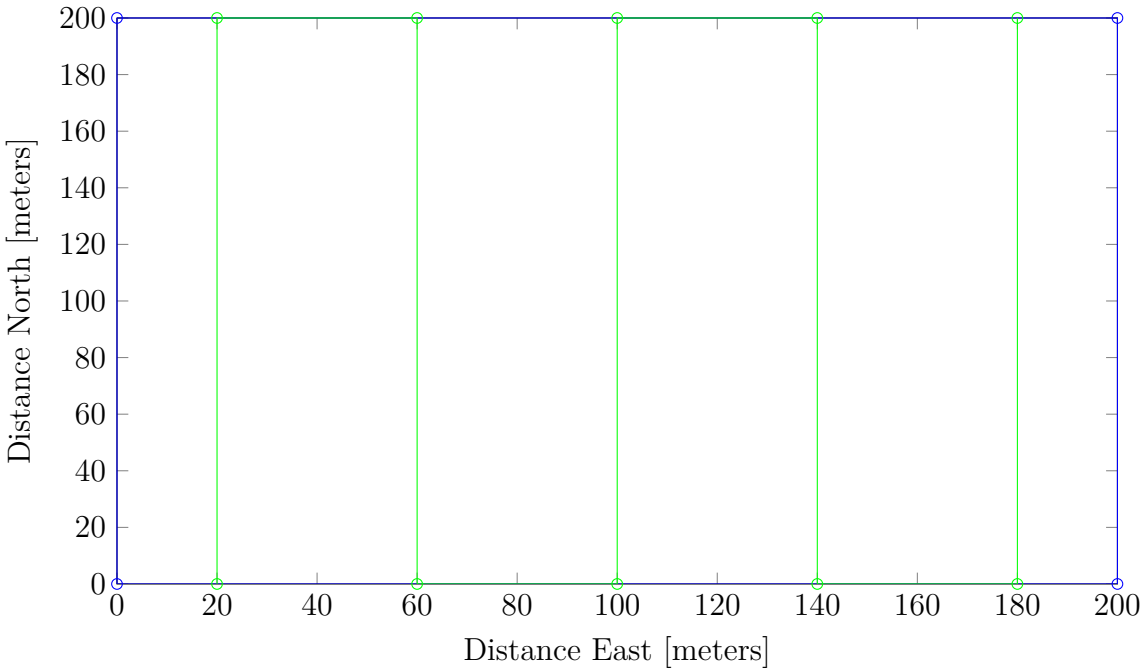


Figure B.1: A quadratic search area

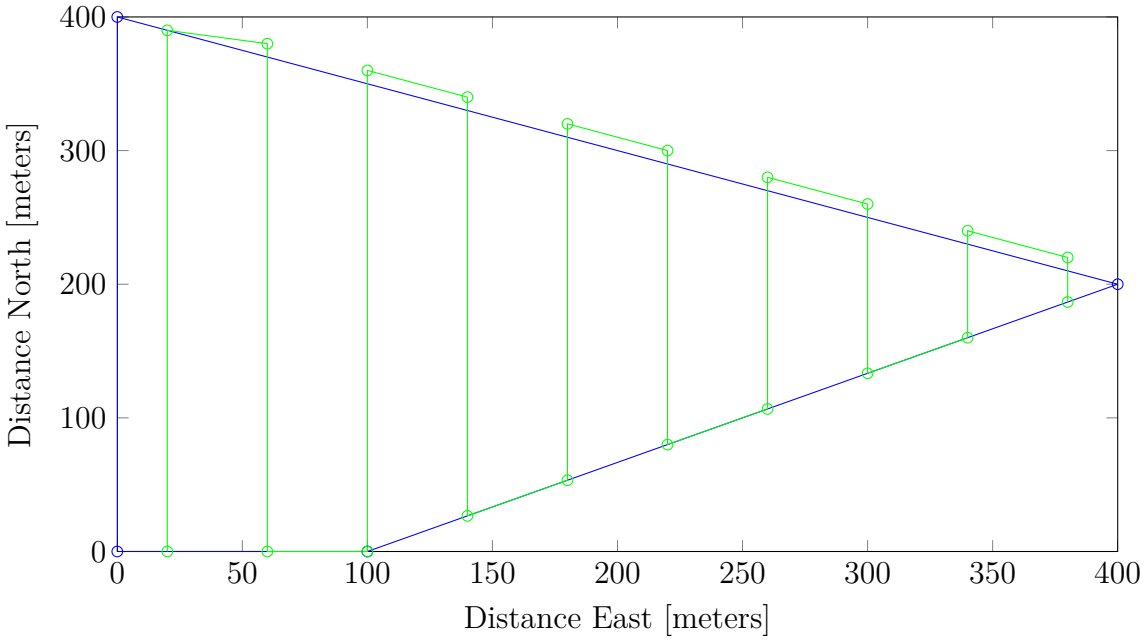


Figure B.2: A triangular search area

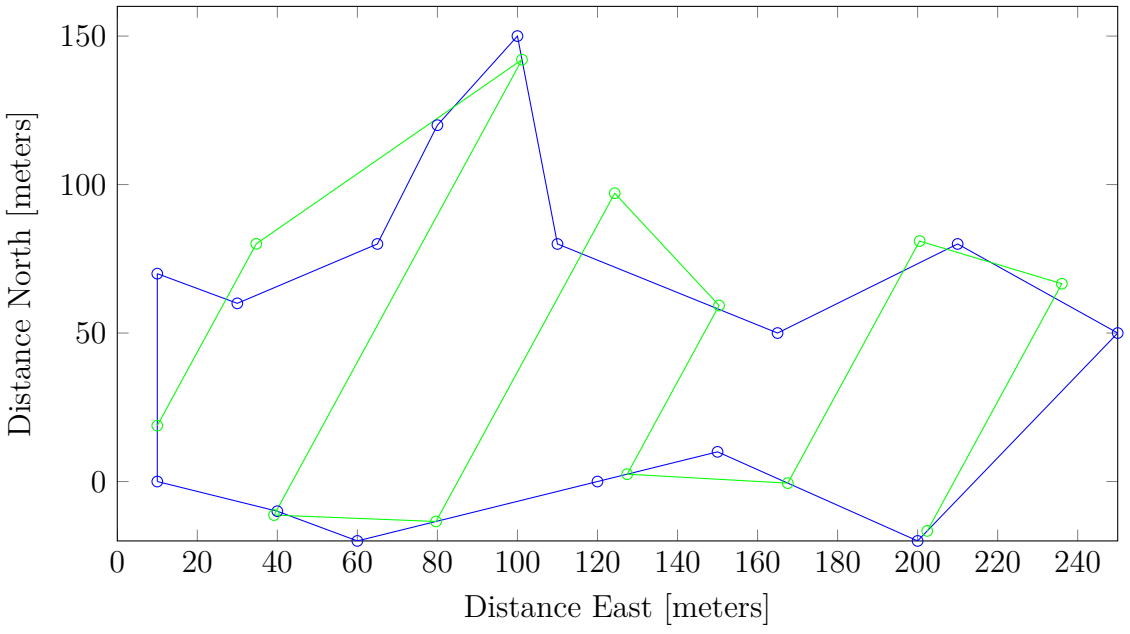


Figure B.3: A strangely-shaped search area

Appendix C

IMC-Messages

This chapter shows the IMC-messages created in this project.

Listing C.1: IMC-message for the Square Search Maneuver

```
<message id="1100" name="Square Search Manuever" abbrev="SquareMan
    " source="ccu">
  <description>Square Search Pattern</description>
  <field name="Timeout" abbrev="timeout" type="uint16_t" unit="s
      ">
    <description>
      The amount of time the maneuver is allowed to run.
    </description>
  </field>
  <field name="Latitude WGS-84" abbrev="lat" type="fp64_t" unit=
      "rad" min="-1.5707963267948966" max="1.5707963267948966">
    <description>
      WGS-84 Latitude of target waypoint.
    </description>
  </field>
  <field name="Longitude WGS-84" abbrev="lon" type="fp64_t" unit
      ="rad" min="-3.141592653589793" max="3.141592653589793">
    <description>
      WGS-84 Longitude of target waypoint.
    </description>
  </field>
  <field name="Z Reference" abbrev="z" type="fp32_t" unit="m">
    <description>
      Maneuver reference in the z axis. Use z_units to specify
      whether z represents depth, altitude or other.
    </description>
  </field>
  <field name="Z Units" abbrev="z_units" type="uint8_t" value="0
      " unit="Enumerated" enum-def="ZUnits">
```

```
25        <description>
26          Units of the z reference.
27        </description>
28      </field>
29      <field name="Speed" abbrev="speed" type="fp32_t">
30        <description>
31          Maneuver speed reference.
32        </description>
33      </field>
34      <field name="Speed Units" abbrev="speed_units" type="uint8_t"
            value="0" unit="Enumerated" enum-def="SpeedUnits">
35        <description>
36          Speed units.
37        </description>
38      </field>
39      <field name="Bearing" abbrev="bearing" type="fp64_t" unit="rad
            " min="0" max="6.283185307179586">
40        <description>
41          Pattern bearing angle.
42        </description>
43      </field>
44      <field name="Cross Angle" abbrev="cross_angle" type="fp64_t"
            unit="rad" min="-1.047197551197" max="1.047197551197">
45        <description>
46          Rows cross angle reference.
47        </description>
48      </field>
49      <field name="Length" abbrev="length" min="0" type="fp32_t"
            unit="m">
50        <description>
51          Length of the maneuver.
52        </description>
53      </field>
54      <field name="Step" abbrev="step" type="fp32_t" unit="m" min="0
            " value="30">
55        <description>
56          The step between each line.
57        </description>
58      </field>
59      <field name="Custom settings for maneuver" abbrev="custom"
            unit="TupleList" type="plaintext">
60        <description>
61          Custom settings for maneuver.
62        </description>
63      </field>
64    </message>
```

Listing C.2: IMC-message for the Sector Search Maneuver

```
<message id="1101" name="Sector Search Manuever" abbrev="SectorMan
    " source="ccu">
  <description>Sector Search Pattern</description>
  <field name="Timeout" abbrev="timeout" type="uint16_t" unit="s
      ">
    <description>
      The amount of time the maneuver is allowed to run.
    </description>
  </field>
  <field name="Latitude WGS-84" abbrev="lat" type="fp64_t" unit=
      "rad" min="-1.5707963267948966" max="1.5707963267948966">
    <description>
      WGS-84 Latitude of target waypoint.
    </description>
  </field>
  <field name="Longitude WGS-84" abbrev="lon" type="fp64_t" unit
      ="rad" min="-3.141592653589793" max="3.141592653589793">
    <description>
      WGS-84 Longitude of target waypoint.
    </description>
  </field>
  <field name="Z Reference" abbrev="z" type="fp32_t" unit="m">
    <description>
      Maneuver reference in the z axis. Use z_units to specify
      whether z represents depth, altitude or other.
    </description>
  </field>
  <field name="Z Units" abbrev="z_units" type="uint8_t" value="0
      " unit="Enumerated" enum-def="ZUnits">
    <description>
      Units of the z reference.
    </description>
  </field>
  <field name="Speed" abbrev="speed" type="fp32_t">
    <description>
      Maneuver speed reference.
    </description>
  </field>
  <field name="Speed Units" abbrev="speed_units" type="uint8_t"
      value="0" unit="Enumerated" enum-def="SpeedUnits">
    <description>
      Speed units.
    </description>
  </field>
  <field name="Bearing" abbrev="bearing" type="fp64_t" unit="rad
      " min="0" max="6.283185307179586">
    <description>
```

```
41          Pattern bearing angle.
42        </description>
43      </field>
44      <field name="Cross Angle" abbrev="cross_angle" type="fp64_t"
            unit="rad" min="-1.047197551197" max="1.047197551197">
45        <description>
46          Rows cross angle reference.
47        </description>
48      </field>
49      <field name="Length" abbrev="length" min="0" type="fp32_t"
            unit="m">
50        <description>
51          Length of the maneuver.
52        </description>
53      </field>
54      <field name="Custom settings for maneuver" abbrev="custom"
            unit="TupleList" type="plaintext">
55        <description>
56          Custom settings for maneuver.
57        </description>
58      </field>
59    </message>
```

Listing C.3: IMC-message for the Barrier Patrol Search Maneuver

```
1 <message id="1102" name="Barrier Patrol Manuever" abbrev="
      BarrierMan" source="ccu">
2     <description>Barrier Patrol Search pattern</description>
3     <field name="Timeout" abbrev="timeout" type="uint16_t" unit="s
          ">
4       <description>
5         The amount of time the maneuver is allowed to run.
6       </description>
7     </field>
8     <field name="Latitude WGS-84" abbrev="lat" type="fp64_t" unit=
          "rad" min="-1.5707963267948966" max="1.5707963267948966">
9       <description>
10        WGS-84 Latitude of target waypoint.
11      </description>
12    </field>
13    <field name="Longitude WGS-84" abbrev="lon" type="fp64_t" unit
          ="rad" min="-3.141592653589793" max="3.141592653589793">
14      <description>
15        WGS-84 Longitude of target waypoint.
16      </description>
17    </field>
18    <field name="Z Reference" abbrev="z" type="fp32_t" unit="m">
19      <description>
20        Maneuver reference in the z axis. Use z_units to specify
```

```
21          whether z represents depth, altitude or other.
22        </description>
23      </field>
24      <field name="Z Units" abbrev="z_units" type="uint8_t" value="0
            " unit="Enumerated" enum-def="ZUnits">
25        <description>
26          Units of the z reference.
27        </description>
28      </field>
29      <field name="Speed" abbrev="speed" type="fp32_t">
30        <description>
31          Maneuver speed reference.
32        </description>
33      </field>
34      <field name="Speed Units" abbrev="speed_units" type="uint8_t"
            value="0" unit="Enumerated" enum-def="SpeedUnits">
35        <description>
36          Speed units.
37        </description>
38      </field>
39      <field name="Bearing" abbrev="bearing" type="fp64_t" unit="rad
            " min="0" max="6.283185307179586">
40        <description>
41          Pattern bearing angle.
42        </description>
43      </field>
44      <field name="Cross Angle" abbrev="cross_angle" type="fp64_t"
            unit="rad" min="-1.047197551197" max="1.047197551197">
45        <description>
46          Rows cross angle reference.
47        </description>
48      </field>
49      <field name="Length" abbrev="length" min="0" type="fp32_t"
            unit="m">
50        <description>
51           Side of the box to be patrolled.
52        </description>
53      </field>
54      <field name="Number of Rounds" abbrev="numRounds" type="int8_t
            " min="1">
55        <description>
56          Number of rounds the patrol should make.
57        </description>
58      </field>
59      <field name="Custom settings for maneuver" abbrev="custom"
            unit="TupleList" type="plaintext">
60        <description>
61          Custom settings for maneuver.
62        </description>
```

```
63      </field>
64 </message>
```

Bibliography

[1] APM. Setting up sitl on linux. `http://www.dev.ardupilot.com/wiki/setting-up-sitl-on-linux/`. Accessed: 2014-06-06.

[2] Jean Berger, Nassirou Lo, and Martin Noel. Exact solution for search-and-rescue path planning. *International Journal of Computer and Communication Engineering*, 2(3), 2013.

[3] Scott A Bortoff. Path planning for uavs. In *American Control Conference, 2000. Proceedings of the 2000*, volume 1(6), pages 364–368. IEEE, 2000.

[4] Øyvind Breivik and Arthur A Allen. An operational search and rescue model for the norwegian sea and the north sea. *Journal of Marine Systems*, 69(1):99–113, 2008.

[5] Mathworks Documentation Center. Flat earth to lla. `http://www.mathworks.se/help/aeroblks/flatearthtolla.html`. Accessed: 2013-12-09.

[6] Mathworks Documentation Center. Lla to flat earth. `http://www.mathworks.se/help/aeroblks/llatoflatearth.html`. Accessed: 2013-12-09.

[7] Lance Champagne, R Greg Carl, and Raymond Hill. Agent models ii: search theory, agent-based simulation, and u-boats in the bay of biscay. In *Proceedings of the 35th conference on Winter simulation: driving innovation*, pages 991–998. Winter Simulation Conference, 2003.

[8] Phillip R Chandler, Meir Pachter, and Steven Rasmussen. Uav cooperative control. In *American Control Conference, 2001. Proceedings of the 2001*, volume 1, pages 50–55. IEEE, 2001.

[9] Atif Chaudhry, Kathy Misovec, and Raffaello D'Andrea. Low observability path planning for an unmanned air vehicle using mixed integer linear pro-

gramming. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 3823–3829. IEEE, 2004.

[10] Patrick Doherty and Piotr Rudol. A uav search and rescue scenario with human body detection and geolocalization. In *AI 2007: Advances in Artificial Intelligence*, pages 1–13. Springer, 2007.

[11] Olav Egeland and Jan Tommy Gravdahl. *Modeling and simulation for automatic control*. Marine Cybernetics Trondheim, Norway, 2002.

[12] Don Ferguson. Gis for wilderness search and rescue. In *ESRI federal user conference*, 2008.

[13] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley and Sons Ltd, 2011.

[14] US Coast Guard. Search and rescue. `http://www.uscg.mil/international/affairs/publications/mmscode/english/Chap9.htm`. Accessed: 2013-12-09.

[15] Bernard O Koopman. Search and its optimization. *The American Mathematical Monthly*, 86(7):527–540, 1979.

[16] BO Koopman. Search and screening. oeg report, no. 56. *Center for Naval Analysis, Rosslyn, Va., USA*, 1946.

[17] Frederik Stendahl Leira. Infrared object detection & tracking in uavs. *Norwegian University of Science and Technology*, 2013.

[18] Lanny Lin and Michael A Goodrich. Uav intelligent path planning for wilderness search and rescue. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 709–714. IEEE, 2009.

[19] Carl Magnus Mathisen. Search and rescue opertaions using a fixed-wing uav equipped with an automatically controlled gimbal. *Norwegian University of Science and Technology*, 2014.

[20] Gustav Öst. Search path generation with uav applications using approximate convex decomposition. *Linköping University*, 2012.

[21] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle. *Sensors*, 200:66Hz, 2004.

[22] José Pinto, Pedro Calado, José Braga, Paulo Dias, Ricardo Martins, Eduardo Marques, and JB Sousa. Implementation of a control architecture for net-

worked vehicle systems. In *Navigation, Guidance and Control of Underwater Vehicles*, volume 3(1), pages 100–105, 2012.

[23] QGroundControl. Mavlink. `http://www.qgroundcontrol.org/mavlink/start`. Accessed: 2013-05-19.

[24] Morgan Quigley, Blake Barber, Steve Griffiths, and Michael A Goodrich. Towards real-world searching with fixed-wing mini-uavs. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3028–3033. IEEE, 2005.

[25] Allison Ryan and J Karl Hedrick. A mode-switching path planner for uav-assisted search and rescue. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 1471–1476. IEEE, 2005.

[26] National Search and Rescue Committee (NSRC). United states national search and rescue supplement to the international aeronautical and maritime search and rescue manual. `http://www.uscg.mil/hq/g-o/g-opr/manuals.htm`. Accessed: 2014-06-09.

[27] Espen Skjong and Stian Aas Nundal. Tracking objects with fixed-wing uav using model predictive control and machine vision. Norwegian University of Science and Technology, 2013.

[28] Thor Audun Steen. Search and rescue using multicopters. *Norwegian University of Science and Technology*, 2014.

[29] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *Emerging Security Technologies (EST), 2010 International Conference on*, pages 142–147. IEEE, 2010.

[30] Helen Wollan. Incorporating heuristically generated search patterns in search and rescue. *University of Edinburgh*, 2004.