Search and Rescue Mission Using Multicopters

Thor Audun Steen

Master of Science in Cybernetics and Robotics
Submission date: June 2014
Supervisor: Tor Arne Johansen, ITK
Co-supervisor: Kristian Klausen, 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:	Thor Audun Steen
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Søk- og redningsoperasjon med Multicopter
Thesis title (English):	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. Investigate the MAVLink protocol and how it can be used in this mission. Make the necessary changes to the ArduPilot code to enable low-level guidance control on the Pandaboard.
3. Guidance (and control) with waypoint tracking for the multicopter. The proposed guidance algorithm should be viable in a global frame. Investigate how ArduPilot can be utilized for different motion control scenarios and evaluate the need for designing other guidance algorithms for the mission.
4. Methods for performing autonomous operations. Design the methods modular such that they could easily be updated, maintained and extended for future development.
5. Study object dynamic estimation based on input from an external camera module. I.E., given an object location in the camera frame, how can this data be used to describe the motion of the object.
6. The results should be verified by simulations and experiments.
7. Conclude findings in a report. Include Matlab/C-code as digital appendices together with a user-guide.

Start date:	2014-01-18
Due date:	2014-06-14

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

Search and rescue operations can greatly benefit from the use of autonomous unmanned aerial systems to survey the environment and collect evidence about the positions of missing persons. This thesis considers the design of an autonomous multicopter system for use in a search and rescue mission. The ArduPilot Mega is used as the autopilot and is presented together with detailed information about the utilized hardware and software. The design of a low-level control interface is implemented as an extension to APM:Copter using the MAVLink protocol allowing attitude or velocity control of the multicopter.

Furthermore, promising methods for autonomous behavior are discussed and developed with the use of the low-level control interface. The integration of a camera is characterized as a vital part of the fully autonomous search and rescue muliticopter system and presented together with a method used to describe and estimate motion of a target object. The estimator used for the state estimation is the extended Kalman filter.

Finally, experiments of the system are conducted at a test field to demonstrate how it can be utilized and to prove the viability of the complete system. The experiments verify that the autonomous search and rescue multicoper can contribute in a search operation using an observer to spot for objects.

Sammendrag

(Norwegian translation of the abstract)

Søk- og redningsoperasjoner kan ha stor nytte av selvstyrte ubemannede luftfartøy til å undersøke områder for å lokalisere savnede personer. Denne avhandlingen omfatter utformingen av et selvstyrt multikopter-system til bruk i søk- og redningsaksjoner. Autopiloten brukt i systemet er ArduPilot Mega som er presentert sammen med detaljert informasjon om den benyttede maskin- og programvaren. Et grensesnitt for lavnivå styring er utformet og implementert som et tillegg til APM:Copter og bruker MAVLink-protokollen til kontroll av flygestilling eller hastighet av multikopteret.

Videre er lovende metoder for autonom oppførsel diskutert og utviklet ved bruk av grensesnittet for lavnivå styring. Integreringen av et kamera er karakterisert som en viktig del av et fullstendig autonomt multikopter-system for bruk i søk og redning, og er presentert sammen med en metode som brukes for å beskrive og beregne bevegelsen til et målobjekt. Estimatoren som benyttes for tilstandsestimeringen er et utvidet Kalman-filter.

Til slutt er det utført forsøk på et testfelt for å vise bruksområdet til systemet og for å bevise at det komplette systemet er levedyktig. Forsøkene bekrefter at det autonome søk og redning-multikoperet kan bidra i en søkeoperasjon der en observatør blir brukt for å oppdage savnede personer.

Preface

The present thesis is submitted in partial fulfillment of the requirements for the degree MSc at the Norwegian University of Science and Technology in the field of engineering cybernetics.

I would like to thank my supervisor Professor Tor Arne Johansen for guidance and overall support for this project. I would also like to thank PhD student Kristian Klausen for the great interest he has shown in this project. I am grateful for all his support, guidance and excellent comments when I have discussed matters with him.

Finally, I would like to thank my parents and Tina for their never-ending support.

Trondheim, 14. June, 2014.

Thor Audun Steen

Table of Contents

List of Tables

List of Listings

List of Figures

List of Abbreviations

AHRS	Attitude Heading Reference System
APM	ArduPilot Mega
AUV	Autonomous Underwater Vehicle
AV	Audio/Video
CPU	Central Processing Unit
DOF	Degrees of Freedom
DUNE	DUNE Unified Navigational Environment
EEPROM	Electrically Erasable Programmable Read-Only Memory
EKF	Extended Kalman Filter
FOV	Field of View
FPV	First-Person View
GCS	Ground Control Station
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
GPS	The NAVSTAR Global Positioning System
HIL	Hardware In Loop
HTTP	Hypertext Transfer Protocol
IMC	Inter-Module Communication
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
JRCC	The Joint Rescue Coordination Centers
LiPo	Lithium Polymer
LOS	Line-of-Sight

LSTS	*Laboratório de Sistemas e Tecnologias Subaquáticas*, The Underwater Systems and Technology Laboratory, located in Porto
LWIR	Long-Wavelength Infrared
MAVLink	Micro Air Vehicle Link
MEMS	Micro-Electromechanical System
MoJ	The Ministry of Justice and Public Security
PID	Proportional, Integral and Derivative
PWM	Pulse Width Modulation
RC	Radio Controller
RTL	Return-to-Launch
SAR	Search and Rescue
SBC	Single-Board Computer
SIL	Software In Loop
TCP	Transmission Control Protocol
TIL	Target In Loop
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol

Chapter 1

Introduction

Search and rescue (SAR) is a term used about the search for, and provision of aid to persons or property in potential or actual distress. To assist in search operations, an unmanned aerial vehicle (UAV) can provide critical support from the air. This thesis presents a system using multicopters to support search and rescue missions.

1.1 Background and Motivation

An unmanned aerial vehicle is an airborne vehicle without a human pilot aboard. The multicopter is a UAV consisting of multiple fixed-pitch rotors attached to a simple mechanical construction. The vehicle uses thrust force generated by rapidly spinning rotors to push air downwards keeping the vehicle aloft. As a side effect of the thrust, the propellers also generate torque making the multicopter free to move and rotate in three dimensions. The total generated thrust and torque from each individual motor controls the motion of the multicopter. However, since this is the only control input, the multicopter is highly unstable and the control problem has proven to be difficult and impossible without internal navigation to track the position and orientation.

Using measurements provided by accelerometers and gyroscopes an inertial navigation system (INS) is a device that computes the real-time position, velocity, and orientation of a moving vehicle using motion sensors. INSs are widely used in military and commercial projects, and have been under constant development and revision for several decades. Recent advances in the construction of microelectromechanical systems (MEMS) have made it possible to manufacture small and light INSs and made the technology accessible to hobbyists. This has resulted in an increasing popularity of the multicopter. Figure 1.1 shows a six arms configuration of the multicopter.

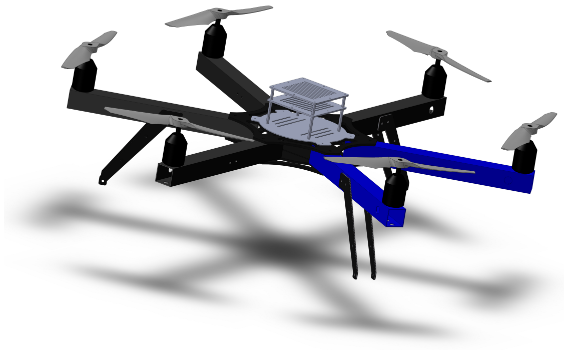


Figure 1.1: The 3DR ArduCopter Hexa is a multicopter designed and manufactured at the 3D Robotics headquarters in San Diego, California. The model is rendered by Kristian Klausen.

Historically, the term and primary use of UAVs have been in the military sector. The civilian applications have gained momentum caused by the continuous development and technology transfer, which has made the technology available outside the military domain. Today, most of the civilian applications use handheld radio transmitters to remotely control their UAVs. In addition, some users have equipped their UAVs with a camera allowing them to control their vehicles using first-person view (FPV) for various tasks. FPV, also known as video piloting, is a method used to control a radio controlled vehicle from the pilot's point of view and is especially great for inspections. The next big technological aspect is autonomous control.

The degree of autonomy in a UAV varies greatly, ranging from manual control to fully autonomous operation. Remotely operated vehicles, such as radio controlled hobby airplanes or helicopters, belong to a category where the degree of autonomy is minimal because continuous input from an operator is required. The main reason for introducing autonomous control is to avoid using pilots for manual control and with that simplify the process seen from the perspective of the operator. Although the system itself will be more technologically complex, less training is required for the operators and human error factors are reduced. Fully autonomous UAVs are characterized by the ability to maintain flight and operate based on self-governed decisions in real-time. The autonomous UAV is able to carry out sequences of actions prepared by an operator to achieve *high-level* goals. The natural high-level goal in a search and rescue mission is to find objects and report their positions.

Now, with UAVs becoming cheaper and more available to the public, search and rescue is an area that can benefit greatly from this technological development. According to statistics from the Joint Rescue Coordination Centers (JRCC), *Hovedredningssentralene*, the number of persons reported missing on land in Norway is increasing. In 2008, the numbers exceeded for the first time thousand persons annually, and the numbers are trending upwards (see Figure 1.2).

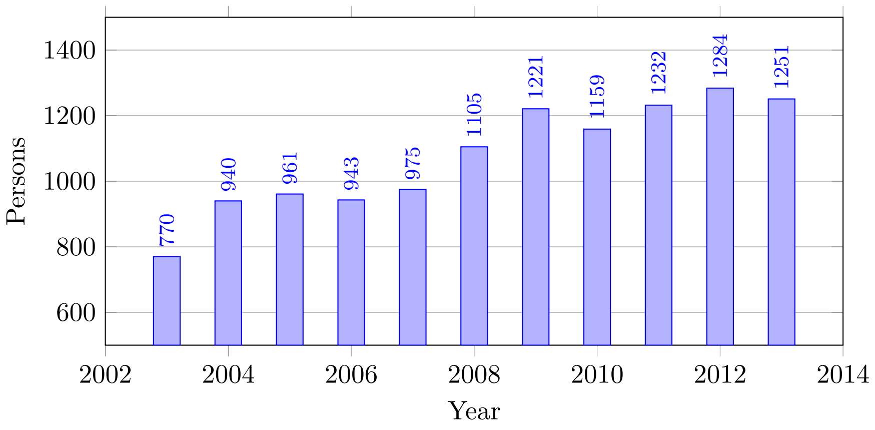


Figure 1.2: Persons reported missing on land in Norway over the last ten years according to the Joint Rescue Coordination Centers [2012].

The JRCC have the overall operational responsibility during SAR operations in Norway. They are carried out by cooperation between government agencies, voluntary organizations and private companies who have resources appropriate for rescue services. The Rescue and Emergency Planning Department in the Ministry of Justice and Public Security (MoJ) coordinates the administration of the Rescue Service in Norway. They stated in 2002 that one of the central elements in the Norwegian SAR Service is the large number of voluntary organizations that take part in the operations [MoJ, 2002]. Further, these organizations are particularly characterized as a valuable asset in search operations in forested and mountainous areas, with the ability to field, on short notice, large numbers of personnel who are both trained for the situation and familiar with the terrain.

The considered major lifesaving resource of the SAR services is the 12 Westland Sea King helicopters of the 330 Squadron. The Sea King is specially designed for SAR operations at sea, though it also performs quite well over land. One of the powerful tools that the helicopters have, beside the natural aerial overview, are the thermal imaging camera. This sensor is controlled by an operator and can spot heat signatures on ground. Although helicopters are extremely valuable assets, they are expensive and may not always be available as they operate and cover large areas.

Imagine that every local SAR unit has multicopters equipped with thermal imaging cameras. They can then provide their own valuable air support as early as possible in a search operation, before a Sea King or other helicopters arrive at the scene. To assist in a typical SAR scenario, one or several multicopters can be deployed in an area of interest, perform sensor operations to collect evidence of the presence of a missing person, and report their collected information to a remote ground station or rescue team. As time often is an important factor for success in SAR missions, multicopters can be of vital importance to minimize the time it takes to find persons of interest. A final important aspect of using UAVs in search missions is that they can minimize the risks rescue teams are exposed to in dangerous areas.

1.2 Previous Work and Related Literature

The challenges with autonomous control have been recognized for a long time [Pachter & Chandler, 1998]. Practical aspects of visual-based aerial search through field tests of wilderness search and rescue are presented by Goodrich et al. [2008] in a field report. The report characterizes small (human-packable) unmanned aerial vehicles as the ideal application to provide aerial imagery of the search region that can be large and have potentially limited mobility for ground searchers. The field tests give good insight in some of the practical challenges related to how video information should be presented to a search unit. They specifically describes how UAVs should be integrated into a search team and best can support the search and rescue process.

Further, there are done research that are slightly more specific into a given part related to the use of multicopters in search and rescue missions. An introduction to the basics of control structure for a quadcopter for search and rescue applications is done by Naidoo et al. [2011]. It provides an overview of the dynamics of the multicopter. Waharte & Trigoni [2010] present different search algorithms with the objective of minimizing the time finding the target object. In particular, the benefit of sharing data between UAVs with different search techniques are compared. A survey of autonomous control for UAVs is presented by Hai et al. [2009]. Fossen [2011] has written a handbook on motion control where guidance, navigation and control is extensively presented. Breivik and Fossen have published several papers presenting different guidance laws for marine crafts. One of them is presenting guidance laws for target tracking and path scenarios [Breivik & Fossen, 2008]. The guidance laws in the perspective of UAVs include motion in three dimension, but marine guidance laws (in two dimensions) are still highly relevant since altitude can be decoupled from the three dimensional guidance law to a separate control objective. Waharte et al. [2010] have done observations proving that altitude is important for optimizing the search process due to speed and detection.

There are done a lot of research on how different computer algorithms can be used to track humans in different scenarios. Rudol & Doherty [2008] have done experiments detecting humans using thermal and color imagery specialized for the search and rescue missions. Their technique presented two video cameras. One thermal imaging camera to detect human-temperature silhouettes and further a standard color camera to classify human bodies. The system has a high processing rate of 25 Hz. The experimental results of this system showed that both human bodies and dummies were detected despite the lower temperature of the latter.

Motion estimation from the UAV perspective is done by Prevost et al. [2007] where an extended Kalman filter is used to estimate the states of a moving object. Further in the same work is motion prediction done, which is closely connected to state estimation. In a different approach by Zhiyuan et al. [2010], an adaptive law is designed to estimate the target position with the focus on designing a simple but effective guidance law. Wu et al. [2009] is also addressing the moving target matching tracking, locating calculation and motion estimation. The benefits of the proposed model have been proven in traffic monitoring site experiments and engineer applications.

Open source projects have pushed the field of UAV projects forward. ArduPilot Mega (APM) is an open source multiplatform autopilot, which is able to control autonomous multicopters. It was created in 2007 by the DIY Drones community and has now evolved to the ArduCopter, ArduPlane and ArduRover software projects. The free software approach of Ardupilot is important to keep costs low and availability for the public community. The ArduCopter project is designed to be easily approachable for the novice, while remaining open-ended for custom applications, education, and research use. Although it is one of the most popular autopilot on the market, the source code is not written in a way that is particular easy-to-understand, making it somewhat hard to customize.

Laboratório de Sistemas e Tecnologias Subaquáticas (LSTS), the Underwater Systems and Technology Laboratory in Porto, is specialized on the design, construction and operation of unmanned underwater, surface and air vehicles. Their focus is to develop tools and technologies for the deployment of networked vehicle systems. One of their strength is that all vehicle systems utilize the LSTS control architecture with the help of the LSTS Neptus-IMC-Dune software toolchain (see Figure 1.3) [Pinto et al., 2013]. The toolchain is designed to be modular and can be customized to different operations and mission setups. The MAVLink communication protocol can be used to extend the LSTS toolchain to include Ardupilot and enable low-level control of UAVs. The LSTS has decided to make their research open source stating that international cooperation is the key to success. This makes their software attractive and makes them ideal as a collaborating partner.

The design of a payload that can perform real-time object detection and tracking by UAVs using thermal imaging camera is presented by Leira [2013]. Due to the limited payload capacity of UAVs, the payload system was designed to be small, lightweight and power efficient. Leira implemented two different detection algorithms where both turned out to have their strengths and weaknesses. The success of object detection is determined by the feature representation and the learning algorithm. The feature representation is responsible for how an object is represented, while the learning algorithm decides whether the object is found or not. The learning algorithm is referred to as a classifier and needs to be trained for

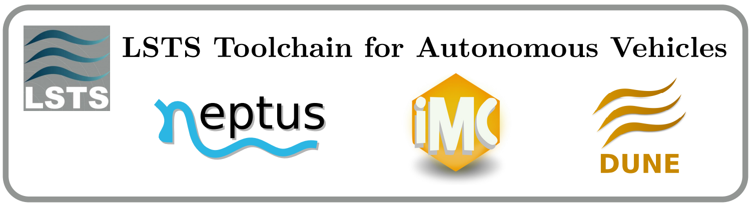


Figure 1.3: The LSTS Neptus-IMC-Dune software toolchain. Neptus is a distributed command, control, communications and intelligence framework for operations with networked vehicles, systems, and human operators. IMC is a communications protocol that defines a common control message set understood by all types of LSTS nodes. DUNE is the system for vehicle on-board software used for e.g. control, navigation, or to access sensors and actuators. Based on `http://lsts.fe.up.pt`

a specific object shape. For example, humans have different shapes when they are observed from various angles. Thus, the camera angle that the classifier is trained for is of great importance for the performance of the recognition. Leira trained the classifiers for a horizontal pointing infrared camera. The tracking algorithm implemented was based on a linear motion model making it sensitive for fast, abrupt and relative large displacement rates in the image position. These findings concluded that slow motion in the image frame was important to achieve good results. To produce good estimates of the real world coordinates of the object Leira proposes to extend the object tracking algorithm, having one estimator for the position in the image plane, and a second estimator for the real world coordinate.

The APM, the LSTS software toolchain and the work done by Leira [2013] are directly related to some of the work in this thesis.

1.3 Contribution and Scope of this Thesis

The goal of this thesis is to design a system using multicopters to give air support for a search and rescue mission. The main focus is to develop a payload for the multicopter that can autonomously perform real-time search for objects. Although the multicopter should be able to communicate and receive updated commands from a ground station, it should primarily be able to conduct a search mission based on some predefined parameters. These parameters can differ for each mission regarding for example search area, search pattern, flight time, speed, altitude and the autonomous reaction upon detection of an object.

The objectives of this thesis can be divided into the following parts:

Hardware and software As the manual controlled multicopter using FPV already have potential to be used in search operations, this thesis strives to take the system one step further and add autonomous functionality. One challenge is to combine and exploit the potential of hardware and software that already exist. Although the system should be designed and optimized for the SAR mission, it should be made modular so it easily could be adjusted to other applications where autonomous control is needed.

Low-level control To enable autonomous control of the multicopter an on-board payload should be designed. The payload will consist primarily of a computational unit that communicates with the on-board autopilot. The computer should be able to control the multicopter attitude and altitude, in addition to give velocity or waypoint-based commands.

Visual feedback The utilization of a camera module should be investigated. The infrared camera setup discussed by Leira [2013] should be considered in addition to a regular color camera. Methods for stabilization of the camera module should be tested with the purpose of increasing the camera performance.

Communication The ground control station should be able to monitor status data and track motion of the multicopter. The data should include vital information about the multicopter together with a video stream from the camera. The ground control station should have the possibility to change predefined parameters for the mission, abort the mission or make other necessary interactions.

Autonoumous behavior Promising methods for performing autonomous operations based on real-time object detection from a thermal imaging camera should be implemented and evaluated. The methods should be made modular such that they could easily be updated, maintained and extended for future development.

For the implementation of the system to be considered successful, it should be tested and verified to be able to accomplish a real search for objects. The following points are a set of partial goals that together define the complete search and rescue mission.

- Define hardware and software that are suitable for the search operations.
- Be able to control roll, pitch, yaw or the velocity together with the altitude of the multicopter externally from ground and on-board the vehicle.
- Be able to control the multicopter by giving waypoints that include speed, heading and altitude control.
- Be able to show status data and track motion of the multicopter on a ground control station.
- Send video from the multicopter to a control station located on the ground.
- Implement a module that from ground can give input to the autonomous unit on-board the multicopter due to mission planning and manual detection of possible targets.

- Autonomously execute a set of maneuvers based on camera detection using low-level control.
- Autonomously resume to the primary search after the object is inspected.
- Estimate the motion of a target object from noisy measurements.

The ultimate goal is to be able to conduct a field test showing the concept of the search and rescue multicopter. The multicopter should be able to perform a search given by waypoints, and given that a detection of an object occur it should be able to do strategic maneuvers addressed to give optimal information to a ground control station.

1.4 Organization of this Thesis

The hardware and software utilized in the SAR multicopter system is described in Chapter 2. The presented system is designed to achieve low-level control of the multicopter. The overall system description is provided at the end to summarize and give an understanding of how the different hardware and software components communicate and cooperate.

In Chapter 3, an introduction to APM:Copter and DUNE is given. The introduction gives an overview of the structure and existing features of the two most important software modules in this thesis. Further, Chapter 4 gives an introduction to the principles of guidance, navigation and control. The definitions are given and followed by a correspondingly presentation of the realization of each subsystem in APM:Copter.

The implementation of a low-level control interface using the MAVLink protocol is presented in Chapter 5 as an extension to APM:Copter. In addition, a customized flight mode is implemented to deal with possible security threats due to the use of the MAVLink protocol.

The autonomous behavior of the multicopter is discussed in Chapter 6. The utilization of a pre-programmed waypoint guidance scheme is presented before the fully autonomous search and rescue multicopter is introduced. A framework for fully autonomous guidance control is designed and a draft to a search and rescue guidance controller is implemented. Further, the integration of a camera module in the the fully autonomous search and rescue multicopter is discussed and related to Chapter 7 where an approach to estimate the position of a detected object is presented using the extended Kalman filter. The estimation algorithm is simulated in MATLAB.

In Chapter 8, the development process is presented through different simulators and concluded with final experiments in the field. The experimental setup of a search operation is given and the results of the field test are presented. The complete checklist preparing for a flight is given in Appendix A.

Chapter 9 briefly summarizes the results and generally concludes the thesis, proving the viability of the search and rescue multicopter system that has been presented.

The digital appendix included with this thesis contains the MATLAB file for the extended Kalman filter implementation used in the simulation in Chapter 7. It also includes a video of the experiments edited by Kristian Klausen. The source-code of the extensions to APM:Copter and the presented tasks in DUNE are found in the digital appendix but also on the uavlab git repositories:

`git@uavlab.itk.ntnu.no:uavlab/ardupilot.git`
Under the branch *feature/ArduCopter-ctrl*

`git@uavlab.itk.ntnu.no:uavlab/dune.git`
Under the branch *feature/stable-object-detect*

Chapter 2

The Search and Rescue Multicopter System

Weight and functionality are two of the most important aspect when the search and rescue (SAR) multicopter system is designed. The combined weight of the multicopter and payload is limited due to influence of the maneuverability, but until the total weight threshold is approached it mainly affect the flight time. In this thesis, designing a mulitcopter with possibility for low-level control, the functionality has been the major focus at the expense of weight. To make the realization of a lighter system possible to be conducted at any time, hardware and software independence has been important. In this chapter, the chosen hardware and software modules that form the SAR multicopter system is presented together with an overall system description.

2.1 Hardware

2.1.1 Multicopter

There are a wide variety of multicopter shapes, sizes and configurations on the market. The multicopter or multirotor is a simple mechanical construction consisting of multiple arms attached to the center with a motor at each end. The number of arms and motors varies and affects the desired lift capacity. As a rule of thumb, the lift capacity of all motors must be twice as much as the total weight of the multicopter with payload. This gives the multicopter enough thrust to maintain its maneuverability and specifies the hexacopter lift capacity.

The maximum lift capacity of the hexacopter is directly determining the maximum weight of the payload. Indirectly it also determines the flight time. There is usually an objective to maximize flight time in every system. The logical way to do this is to increase the battery capacity. However, since extra batteries correlate to an increase in weight, this may not be a possible due to the weight threshold. As a general rule, a doubling of battery capacity only give 50 % of increased flight time. Thus, the optimized multicopter setup regarding flight time will differ depending on individual configuration.

The hexacopter, a multicopter with six arms and rotors, is considered to be a good compromise between production costs, size and weight, for search and rescue missions. The experiments in this thesis is conducted with the hexacopter shown in Figure 2.1. The specification of *3DR ArduCopter Hexa* is summarized in Table 2.1.

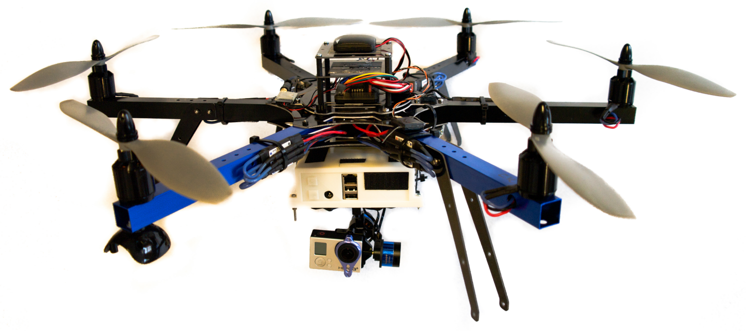


Figure 2.1: The hexacopter 3DR ArduCopter Hexa mounted with the SAR payload. The white box contains the computational unit and act as a platform for the gimbal. The payload and gimbal are discussed later in the section.

Table 2.1: Technical specifications of the hexacopter with payload.

Number of motors and propellers	6
Propeller length and pitch	11×4.7
Hexacopter weight	1550 g
Battery capacity	4000 mAh
Battery type	3 cell lithium polymer (LiPo)
Hexacopter lift capacity	1.0 kg
Gimbal and GoPro 3+	265 g
Box with PandaBoard	210 g
Batteries (2 cell and 3 cell)	430 g
Total payload weight	0.9 kg
Approximately flight time	5 min

2.1.2 Radio Controller

A radio controller (RC) is an essential tool in almost every UAV system (see Figure 2.2). Although the fully autonomous multicopter is able to perform take-off, flights and landing on its own, it must be possible to reclaim manual control if unexpected behaviors happen. This is especially important during testing of new features and in the development process. A RC is the typically tool used for manual control. The RC is sending a PWM (pulse width modulation) signal, which consists of series of repeating pulses of variable width. A 7-channels RC transmitter can send 3 auxiliary PWM signals in addition to roll, pitch, yaw and throttle.

Figure 2.2: The 7-channels RC transmitter Spectrum DX7s with receiver. Retrieved from `http://hobbyfly.com`.

In the context of multicopters, the expression manual control makes little sense since it is impossible to do stabilizing of a muliticopter with manual control for each motor. A more correct expression is *automatic control*, where a change at the RC is given as a meaningful input to the multicopter, e.g. lean angles or thrust. Automatic control is typically achieved by electronic assistance from an autopilot that translates the RC input to individual outputs to each motor stabilizing the multicopter at a given attitude. Automatic control is referred to as *stabilize mode* in some autopilots.

2.1.3 The ArduPilot Mega Autopilot

ArduPilot Mega is an open source multipurpose platform able to control autonomous multicopters, fixed-wing aircrafts, traditional helicopters and ground rovers [APM, 2014a]. The project was created by the DIY Drones community and was based on the Arduino open-source electronics prototyping platform, leading to the "Ardu" prefix. Today, the project support more than just Arduino-compatible hardware and the software projects are renamed with the prefix "APM" to APM:Copter, APM:Plane and APM:Rover. The hardware in APM consists of the core autopilot board, various sensors and accessories to add to its functionality. The APM is build as an inexpensive and "simple to use" autopilot and the free software approach is to keep the focus on availability and low costs. The APM 2.5 is shown in Figure 2.3.

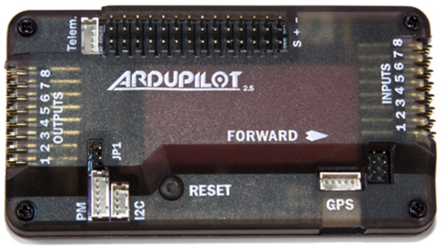


Figure 2.3: ArduPilot Mega 2.5 with side entry case. Retrieved from `http://store.3drobotics.com`

The ArduPilot Mega 2.5 includes the following features [APM, 2014b]:

- 3-axis gyroscope.
- 3-axis accelerometer.
- 3-axis magnetometer.
- Barometric pressure sensor for altitude.
- 5 Hz external NAVSTAR global positioning system (GPS).
- Voltage and current sensors for battery status.
- 4 MB of onboard datalogging memory.
- Built-in hardware failsafe processor, can return-to-launch on radio loss.
- Two-way telemetry and in-flight command using the MAVLink protocol.

The gyro, accelerometer and magnetometer is used to determine the rotation, acceleration and direction of the copter. The APM 2.6 has replaced the on-board magnetometer with an external GPS and compass module to solve addressed problems due to magnetic interference from electronic components. The advantage with an external module is to give freedom due to the placement of the magnetometer. The ideal position of gyroscope and accelerometer is in the center of the vehicle where magnetic interference typically is highest. The external module makes it possible to move the magnetometer away from the problematic area keeping gyroscope and accelerometer at their optimal position.

2.1.4 Single-board Computer

The PandaBoard is the on-board processing unit in the multicopter system. In the research to find the most optimal low-power and low-cost single-board computer (SBC) to be used in UAVs, Leira [2013] concluded that the PandaBoard is most optimal. A SBC is a complete computer integrated on a single circuit board. As the SBC has high level of integration of all components it allows the system to be compact, lightweight and power efficient which makes them ideal for the use in UAVs. The PandaBoard is found as the most optimal SBC due to the criteria of CPU speed, amount of RAM, weight, size, power consumption and compatibility with GNU/Linux software. The specification of the PandaBoard ES is given in Table 2.2.

Table 2.2: Technical specifications of PandaBoard ES.

CPU	1.2 GHz Dual Core
Processor	ARM Cortext-A9
Weight	82 g
Size	11.4 cm×10.1 cm×3 cm
Power	DC Jack / USB On-The-Go - 5 V ∼ 1 A
Connectivity	10/100 Ethernet and 802.11 b/g/n
Memory	1 GB RAM and full size SD/MMC card cage
Expansion	USB 2.0 High-Speed host port

The range of the original on-board wifi module at the PandaBoard is tested to be approximately 150 meters. This was tested with the DLINK DIR-615 wireless N300 router. Figure 2.4 illustrate the results of the test.

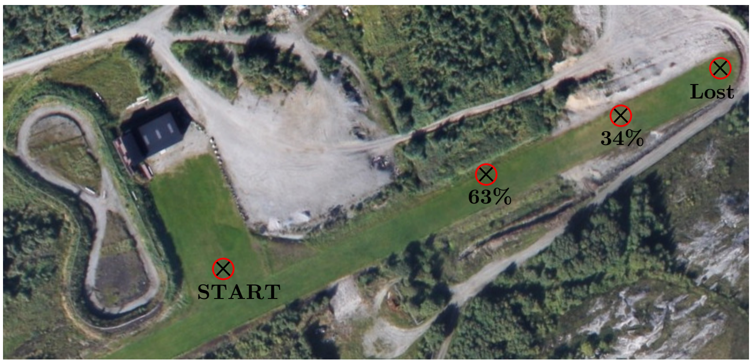


Figure 2.4: The range of the original wifi module at the PandaBoard was tested in field. The airstrip at the picture is approximately 200 meters from start to the end of the runway where the communication was lost. The map data is retrieved from `http://norgeibilder.no`

2.1.5 Gimbal

Horizontal motion of a multicopter is accomplished by an increase of thrust on one side and a reduce of thrust on the other side causing the copter to tilt in the desired direction of motion. Consequently, the frame will almost be in constant motion in order to move and further stabilize the copter. In turbulent areas, the motion will increase as the copter automatically tilts against the direction of the disturbance. As a result, a camera mounted directly on the multicopter frame will obtain a lot of unwanted movement. This can be solved by the use of a gimbal (see Figure 2.5).

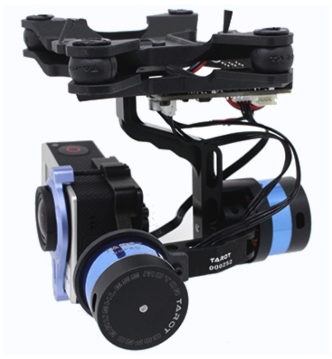


Figure 2.5: The Tarot 2D brushless camera gimbal with a camera mounted. Retrieved from `http://www.droneshop.com`

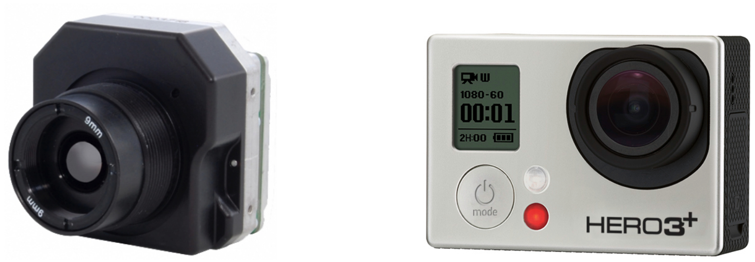
A gimbal is a platform that can pivot, meaning that instead of being fixed to a base, the platform can rotate along at least one axis. It is adequate to have a 2-axis gimbal on a multicopter neutralizing the motion in roll and pitch. The yaw axis of the camera can be kept fixed with the multicopter frame allowing the pilot to control the camera direction by rotating the copter in yaw.

The Tarot 2D camera gimbal is specially designed for the GoPro 3 and is tested with great success. It has a separate control board with an IMU (inertial measurement unit) mounted on the back of the mounting block for the GoPro. The platform is turned with brushless motors giving smooth and steady results. Although the Tarot 2D is designed for the GoPro, it can be used for other cameras at the same size and weight. The infrared camera is about the same as the GoPro in weight but is slightly thicker. The motors can handle the unbalanced weight but would use less power with a perfectly balanced mounting block for the infrared camera.

2.1.6 Video Streaming

To support the search and rescue operations with multicopters, the camera is a vital part of the system. To utilize the video stream for automatically object recognition, computer vision algorithm can be used to process, analyze and understand the images. The techniques for extracting information that can be used at the single-board computer are limited as they often are computational demanding. To produce robust and adequate results in a color-varying environment which search location typical are, thermal imaging hold an advantage over color cameras showing heat signatures and therefore neglecting color variations. It is possible to buy infrared cameras in a size that can be mounted on a multicopter to the price range of 3000 EUR.

The Flir Tau 2 long-wavelength infrared (LWIR) thermal imaging camera and GoPro 3+ color camera was tested in this thesis (see Figure 2.6).



(a) The Flir Tau 2 LWIR thermal imaging camera. Retrieved from `http://www.flir.com`

(b) The GoPro 3+ Black Edition. Retrieved from `http://gopro.com`

Figure 2.6: The cameras used in the experiments. The real-time outputs of both cameras are analog.

In a fully autonomous SAR multicopter system the camera is used as a sensor to detect objects. A digital camera stream is necessary to process images and do computer vision analysis on-board the multicopter at the single-board computer or at a ground station. The video stream can be transmitted by wireless communication at 5.8 GHz with a module specialized for FPV (see Figure 2.7). This transmission is analog outputting audio/video (AV) signals.

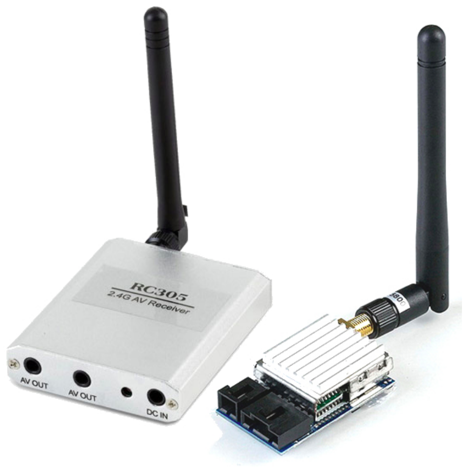


Figure 2.7: 5.8 GHz 200 mW FPV wireless AV Tx and Rx set. It has a range at 500 m with the supplied antenna. An optional directional antenna can extend the range to over 1 km. Retrieved from `http://www.uavobjects.com`

When a analog video stream is used, either from the GoPro or IR-Camera, it must be converted before a digital unit can utilize the stream. Frame grabbers are devices that inhibit this feature. There are several types on the market, which has different outputs.

EasyCap DC60 is a USB audio and video grabber that can capture high-quality video (see Figure 2.8a). External power is unnecessary and it is a simple solution that works directly on most (linux-based) computers.

The AXIS M7001 video encoder can deliver two simultaneous video streams at full frame rate, one in *H.264* and another in *Motion JPEG* (see Figure 2.8b). This video encoder is powered over ethernet using the same cable as for data transmission and is specified to need minimum 44 V. A *DC DC boost* can be used to step-up the voltage to the required level from a 3 cell LiPo battery which is same that is used to power the hexacopter.

(a) Easycap DC60. Retrieved from `http://www.mercadolivre.com.br`

(b) AXIS M7001 video encoder. Collected from `http://www.axis.com`

Figure 2.8: Frame grabbers.

2.2 Software

2.2.1 The DUNE Framework

DUNE or *DUNE Unified Navigational Environment* is a runtime environment written in C++ for unmanned systems developed by LSTS [Pinto et al., 2013]. It provides an operating system and architecture independent platform, which makes it work with different payloads and solutions. Together with its portability, DUNE is designed to be both modular and versatile to make it easy to adapt functionality to many types of operations and applications.

To provide the modular environment in DUNE, related logical operations are isolated from each other in different tasks. The tasks can easily be added or removed to customize for individual mission setups. All running instances of DUNE share the same code base but run under different configurations. A single initialization file specifies the configurations for each setup. The modular architecture is based on a set of divided tasks that has separate responsibilities. Some tasks are responsible for the interaction with sensors and actuators related to navigation and control of unmanned systems, and other interact with the control architecture that comprises DUNE's navigation filter, autopilot, maneuvering controllers and supervisors.

2.2.2 The IMC Protocol

The inter-module communication (IMC) protocol is a message-oriented protocol designed and implemented at LSTS for communication between vehicles, sensors and human operators [Martins et al., 2009]. The IMC protocol comprises different logical message groups for networked vehicle and sensor operations. It defines an infrastructure that provides different layers for control and sensing. The IMC protocol serializes a shared set of messages and adds a header enabling it to keep track of the messages. It also checks if the messages arrive at their given destination where the IMC message is reconstructed to the original set of messages through deserialization. The protocol allows different tasks, from sensor drivers to guidance controllers, that run independently from each other on separate threads or processes, to exchange data using a message bus mechanism.

The entire IMC protocol definition is given in a single XML with detailed documentation. This creates a flexible solution in creating new types of messages. All IMC messages contain a header with information like its type, version, time stamp, origin and destination.

2.2.3 The MAVLink Protocol

MAVLink or *Micro Air Vehicle Link* is a protocol specialized for communicating with small UAVs. The protocol is a light-weight, header-only message marshalling library which can pack C-structs over serial channels with high effiency. The protocol can serve as the communication backbone for the IMU and ground link communication as well as for inter-process communication in Linux exchanging data among multiple threads [QGroundControl, 2014].

MAVLink is used as the communication protocol between the APM autopilot and a ground control station. It is also used in the inter-communication between the subsystems of the autopilot. The MAVLink mission interface is a data format for storing missions to be carried out by UAVs. The items of the mission data format can be transmitted using the waypoint protocol or as individual actions using the MAVLink *command* message. The waypoint protocol describes how waypoints are sent to, and read from a UAV. The intention is to ensure a consistent state between sender and receiver.

A transaction between two communication parties can only be initiated when no other transaction is active. This means that both communication parties have to be in state *idle* before a transaction can be initiated. The waypoint protocol supports the following functionality:

- Read, write and clear the waypoint list to the UAV.
- Set a new current waypoint to the UAV.
- A status message is broadcasted from the UAV if it reaches a waypoint or a new waypoint is selected.

Furthermore, the mission interface has a parameter protocol and an image transmission protocol. The on-board parameter interface allows to read and write parameters into the current memory and permanent storage. This can be used to write e.g. PID gains. The image transmission protocol consists of a module for both image and video streaming.

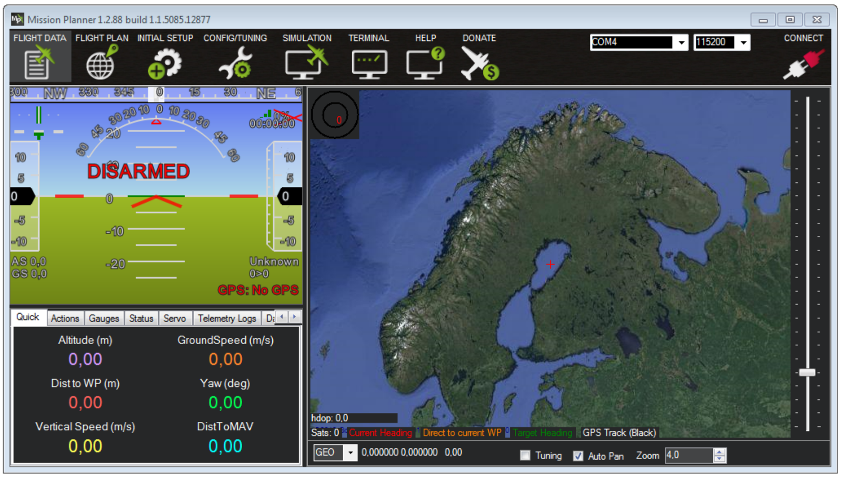
2.2.4 Ground Control Stations

Mission Planner and APM Planner

Mission Planner and APM Planner are ground control stations (GCSs) designed for APM:Plane, APM:Copter and APM:Rover. They are more and less the same tool written in two different languages. *Mission Planner Ground Control Station* is a .NET framework that primarily runs on Windows while *APM Planner Ground Control Station* is written in C++ using Qt. Since it is practical to do development in Ubuntu, Mission Planner is unsuited with its lack of compatibility to other operating system than Windows. Qt however, is a cross-platform application for developers using C++, meaning that it has support for many operation systems including Ubuntu. APM Planner is superior due to speed but Mission Planner works better for logging and flight analysis. As both GCSs basically do the same, a dual boot laptop running both Windows and Ubuntu has shown to be excellent in field, as one GCS sometimes is preferred. Figure 2.9 shows a screenshot of Mission Planner.

Mission Planner and APM Planner have the same features as a configuration utility. They can load firmware into the APM and further setup, configure, and tune the vehicle for optimum performance. As control stations they can plan, save and load autonomous missions into the APM with a simple point-and-click waypoint entry on data maps. Furthermore, they can download and analyze mission logs created by the APM. As control supplements for the UAV, they can monitor the APM's status in operations and record more extended telemetry logs than the on-board datalogging memory.

The support for autonomous control has its limitation when it comes to Mission Planner and APM Planner. They are not designed for more advanced, specialized or larger operations involving networked or multiple vehicles. Mission Planner and APM Planner can only execute predefined flight plans based on basic maneuvers. Features for fully autonomous operations with self-governed decisions on-board the multicopter is not supported by these GCSs. Although it may be



COM4 115200

AS 0,0
GS 0,0

Altitude (m) 0,00
GroundSpeed (m/s) 0,00
Dist to WP (m) 0,00
Yaw (deg) 0,00
Vertical Speed (m/s) 0,00
DistToMAV 0,00

hdop: 0,0
Sats: 0

GEO 0,000000 0,000000 0,00
Zoom 4,0

Figure 2.9: The user interface of Mission Planner.

possible to extend these GCSs supporting such features, it does not appear to be the right direction when other tools are made suitable for such modifications. However, Mission Planner and APM Planner can still be used as supplement to another preferred control station as an external APM configuration tool.

Neptus

Neptus is a command and control software developed by LSTS that allows operators to interact with a dynamic set of available assets in real-time by commanding plans and receiving data from the network. It is written in Java and it run in Ubuntu and Windows. The main communication interface in Neptus is IMC, making it interoperable with DUNE and any other IMC-based peer. Neptus is designed to have adaptability and flexibility to comprise needs from diverse vehicles, scenarios and different operator experiences. As a result, Neptus could be customized according to the search and rescue mission and provide the tools needed for operators and UAVs. Figure 2.10 shows a screenshot of Neptus.

2.3 Overall System Description

Various hardware components and software solutions have now been presented with the objective of together composing the search and rescue multicopter system. A overview is given in Figure 2.11 to present how these together form the complete system.

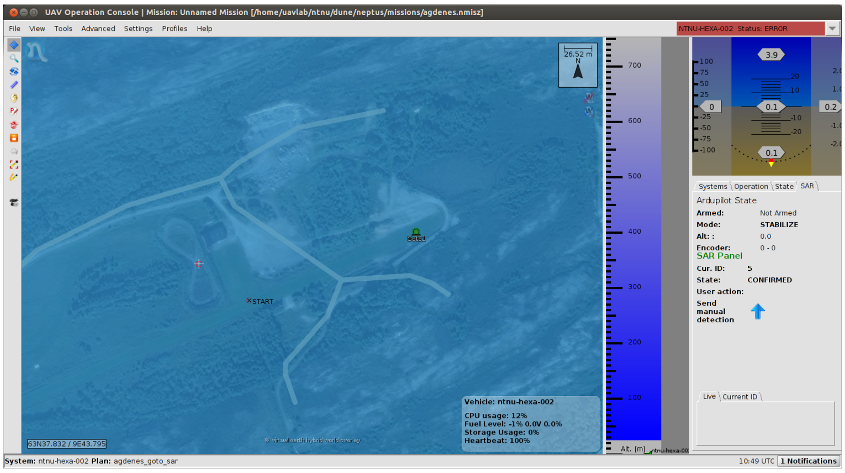

UAV Operation Console | Mission: Unnamed Mission [/home/uavlab/ntnu/dune/neptus/missions/agdenes.nmisz]
File View Tools Advanced Settings Profiles Help
NTNU-HEXA-002 Status: ERROR
START
Goto1
Systems Operation State SAR
Ardupilot State
Armed: Not Armed
Mode: STABILIZE
Alt: : 0.0
Encoder: 0 - 0
SAR Panel
Cur. ID: 5
State: CONFIRMED
User action:
Send manual detection
Live Current ID
Vehicle: ntnu-hexa-002
CPU usage: 12%
Fuel Level: -1% 0.0V 0.0%
Storage Usage: 0%
Heartbeat: 100%
63N37.832 / 9E43.795
System: ntnu-hexa-002 Plan: agdenes_goto_sar
10:49 UTC 1 Notifications

Figure 2.10: The user interface of Neptus.

The search and rescue system can be divided into two main parts, the multicopter with payload and the ground control station with laptop and RC transmitter. In the presented configuration of the system an FPV module is used to stream video to the ground station. This makes an observer needed to manually look for possible targets on ground. A gimbal is used to increase the stability and quality of the video stream.

DUNE is utilized as the runtime environment running at the PandaBoard and can performs autonomous tasks in flight. It receives commands from Neptus and monitors the state of the multicopter transmitting the information to ground. If Neptus is used DUNE is responsible for sending waypoints to APM by the MAVLink protocol. If the two-way telemetry link is used together with Misson Planner or APM Planner, DUNE and Neptus is redundant.

The control station on ground provides the operational overview of the search and rescue operation. A RC transmitter is essential for manually controlling the multicopter while a laptop is used for monitoring and as a control supplement. The video stream is integrated and shown at the laptop, but could also be streamed on an external screen.

Table 2.3 summarizes the overall system of hardware and software components with protocols. The computer vision algorithms are missing from completing the fully autonomous SAR multicopter system. The system is therefore relying on an observer watching the video stream to spot targets on ground. The fully autonomous system can be illustrated in Figure 2.11 with a supplemented communication link between the camera and the PandaBoard running the computer vision algorithms in DUNE.

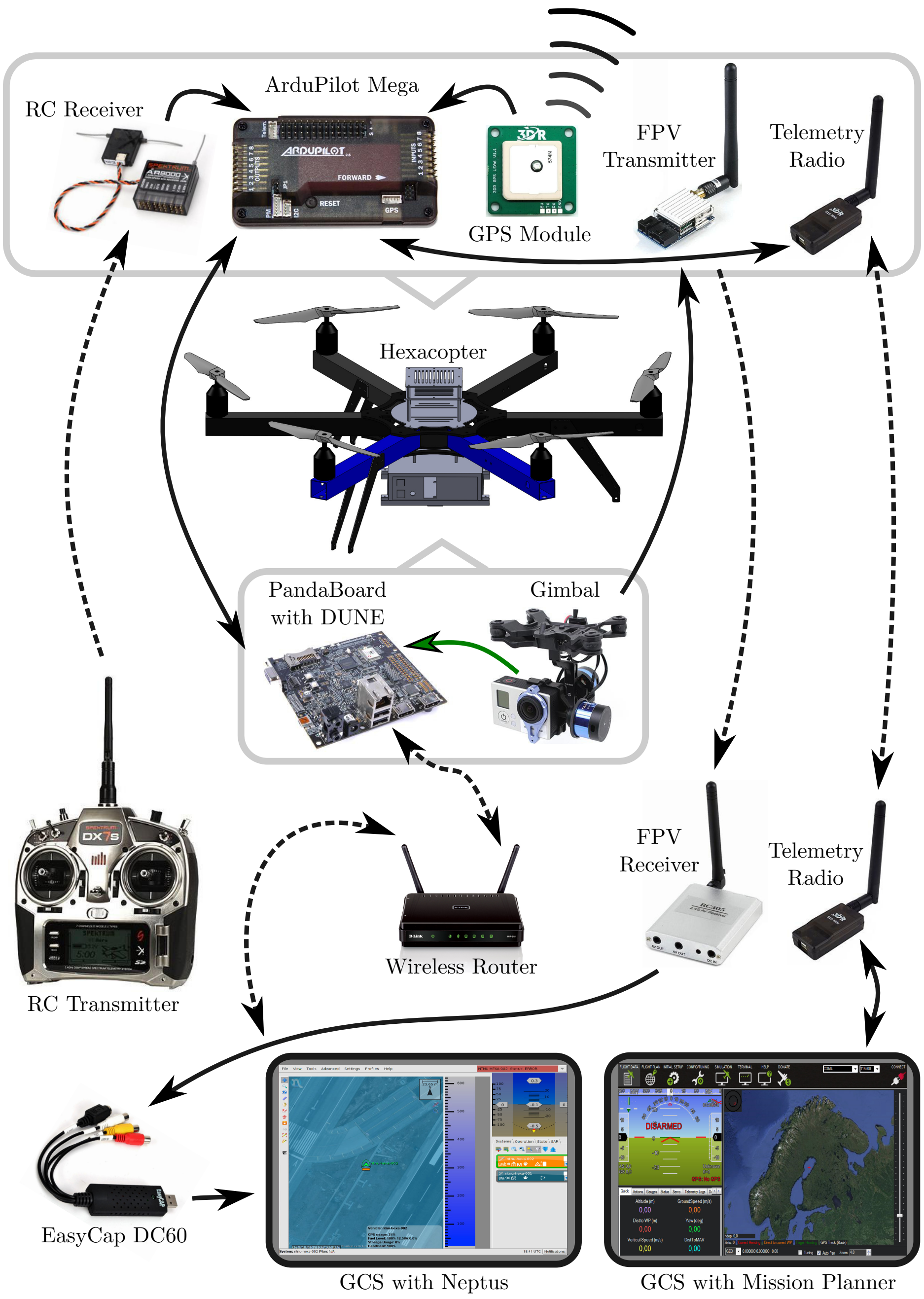


Figure 2.11: The search and rescue multicopter system. A solid line illustrates a wired connection while a dash line illustrates a wireless connection. The communication link from the camera to the PandaBoard illustrate the fully autonomous system.

Table 2.3: The technical specifications of the communication links in the SAR multicopter system. Two-ways communication is illustrated by a double arrow while one-way communication is illustrated by a single arrow. A non-specialized wired communication link is called cabled.

Module	Link	Protocol
RC link	2.4 GHz	PWM
RC receiver $\rightarrow$ APM	Cabled	PWM
uBlox GPS $\rightarrow$ APM	Cabled	UBX/NMEA
DUNE $\leftrightarrow$ APM	Micro USB	MAVLink
PandaBoard with DUNE $\leftrightarrow$ Neptus	2.4 GHz	IMC
GoPro $\rightarrow$ FPV transmitter	Cabled	AV
FPV link	5.8 GHz	AV
Telemetry radio $\leftrightarrow$ APM	Cabled	MAVLink
Telemetry link	433 MHz	MAVLink
Telemetry radio $\rightarrow$ Mission Planner	USB	MAVLink

2.4 Summary

The hexacopter mounted as illustrated in Figure 2.11 is approaching the limit of weight threshold. The gimbal with the GoPro has itself a total weight of about 265 gram. This emphasize that the optimized system setup needs carefully planning when it comes to maximizing flight time without scarifying required tools for performing the search operation.

During the setup of the SAR system, the PandaBoard have been questioned several times whether it is the right choice of single-board computer. Problems with the wifi module, drivers and package dependencies are some of the time consuming issues that have been experienced. An alternative single-board computer, the BeagleBoard, has been tested briefly with promising results and can be considered for future use.

Chapter 3

Introduction to APM:Copter and DUNE

In this chapter, an introduction to the structure in APM:Copter and DUNE is presented together with some of the features they provide. These software projects are the most important modules for realizing low-level control of the multicopter.

3.1 The APM:Copter Software Project

APM:Copter combines personal multicopters with advanced autopilot technology providing an autonomous aircraft for both new and experienced pilots. APM:Copter is developed as a complete UAV solution with specialized hardware and software that provides configuration, mission-planning, mission-operation, and post-mission analysis. It is constantly maintained, improved and updated by a dedicated group of volunteers from the open source community. The entire package is designed to be easily approachable for the novice, while remaining open-ended for custom applications, education, and research use.

3.1.1 Flight Modes and Features

The first official public release of "ArduCopter Alpha 1.0" in October 2010 featured the flight modes *stable* and *acro*. From this limited set of functions APM:Copter has now become a powerful platform with support for GPS, various sensors and many different features. The DIY Drones community is one of the leading communities for personal UAVs, and is one of the groups that have seen the potential for new features and participated in the development of the source code. This is one of the benefits with open source code, that teams of developers from around the globe together can improve and refine the performance and capabilities of APM:Copter at the same time.

APM:Copter states to be a powerful platform and easy to use. One of the features, *simple mode*, supports this statement allowing the pilot not to worry about the orientation of the multicopter when flying. The autopilot automatically decomposes the movement of the copter from the pilot's point of view regardless of which way the copter is facing. This is useful for new pilots that have not mastered adjusting their roll and pitch inputs depending upon which way the vehicle is facing. It is also convenient in cases where the copter is too far away to see the yaw orientation since a stick movement towards the pilot will bring the copter back.

Simple mode can be used in combination with some of the flight modes available in APM:Copter. In "ArduCopter V3.1.1" there are 14 flight modes where about 10 are regularly used. A short explanation of the most important flight modes due to the SAR mission is given in the description that follows. Note that pilot input refers to a stick position at the RC.

Stabilize
: Since multicopters always are in need of automatic control, *stabilize* is the closest flight mode that may be referred to as "manual control". The lean angle of the copter is controlled to the desired angles from the pilot input in roll and pitch. This means that if roll and pitch sticks are centered the copter will automatically levels itself. The pilot input in yaw controls the rate of change in angular orientation meaning that the copter will maintain its current orientation with a centered stick position. The pilot input in throttle controls the average speed of all motors. However, since the vertically lift is dependent of the tilt angle of the copter, the throttle in stabilize mode is tilt compensated making it easier for the pilot to maintain the same altitude regardless of roll and pitch angles.

AltHold
: *AltHold*, which refers to altitude hold, automatically controls the throttle to maintain the current altitude. However, the current altitude is determined from barometer measurements and is therefore sensitive to changes in air pressure and location. Automatic altitude hold is also utilized as throttle mode by other flight modes and can be combined together with different roll-pitch and yaw control algorithms. In the specific flight mode AltHold roll, pitch and yaw is operated as in stabilize mode.

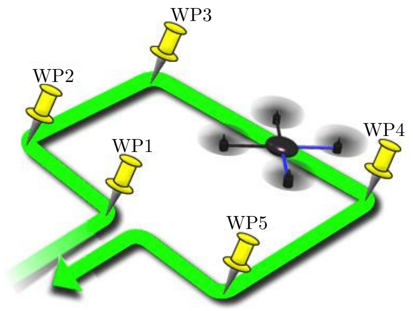
Loiter
: In *loiter* the copter maintains a consistent location, orientation, and altitude. The flight controller uses GPS coordinates and compass angle to keep fixed at a specified location. However, the pilot can move this specified location using the RC and are able to control the copter both horizontally and vertically.

Auto

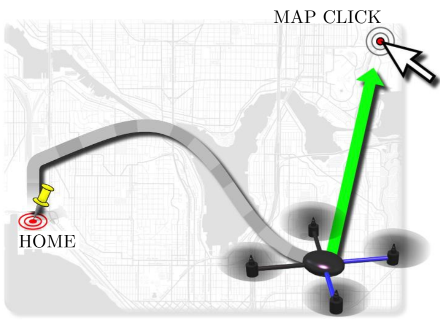
The autonomous multicopter can be pre-programmed and execute missions in *auto* mode. The copter will follow predefined flight routes based on a set of waypoints and be programmed to trigger camera shutters or do other various actions during flight (see Figure 3.1a).

Guided

Guided mode differs from auto mode by allowing the user to interactively command the copter to travel to a new target location instead of following a predefined set of waypoints. A ground control station can be used to define the new target location by clicking on a point on the flight data map. Once the location is reached, the copter will loiter at that location, waiting for the next target (see Figure 3.1b).



(a) Auto mode.



(b) Guided mode.

Figure 3.1: Autonomous flight modes in APM:Copter. Retrieved from `http://ardupilot.com`

APM:Copter is designed to be used together with a ground control station. Mission Planner and APM Planner are two GCSs that have good support for mission planning and operation. All communication can be done over a two-way telemetry link where flight modes, waypoints and parameters can be controlled while the copter is in the air.

3.2 The Software Architecture in DUNE

The open source LSTS software toolchain consisting of Neptus, the IMC protocol and DUNE is designed for supporting networked vehicle systems. Although the SAR system presented in this thesis focuses on how to utilize a single multicopter, it is possible to imagine fleets of UAVs, both multicopters and planes, performing a large scaled SAR mission. With such operations in mind, an ongoing collaboration with LSTS and its support for UAVs, the DUNE framework was a natural choice for the vehicle on-board software.

3.2.1 Control Layers

The underlying architecture of DUNE needs to be flexible to encompass diverse vehicle hardware, different communication methods and various mission scenarios. This is achieved by allowing access to the different systems through different control layers. The layers, as seen in Figure 3.2, gives the system interoperability at different levels keeping the possibilities of both low-level and high-level control.

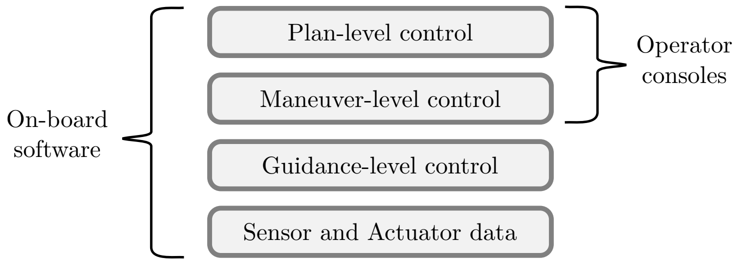


Figure 3.2: The different control layers in DUNE. Plan-level and maneuver-level control can both be done from an external operator console, while the vehicle is within communication reach, or it can be done on-board the vehicle making it a fully autonomous operation. Based on Pinto et al. [2013].

3.2.2 DUNE Tasks

In DUNE, the modularity and flexibility is accomplished by a well-organized structure. All related operations are separated into individual tasks that fit into one of the following categories [Pinto et al., 2013]:

Sensors are device driver tasks, associated with some hardware that measures the environment.

Actuators are device driver tasks for hardware that allows the vehicle to move and interact with the environment.

Estimators are tasks that based on information from sensors calculate state estimates of the observed data. A typically example of an estimator is the navigation task which determines the position, velocity and angular orientation of a vehicle.

Controllers are tasks that handle high-level commands and transform them into low-level commands based on the estimated vehicle state. For instance, a given go-to-here, loiter or circle maneuver starts the corresponding maneuver controller.

Monitors are tasks that constantly check vital parts of the system. They receive information from other tasks and may change the vehicle state accordingly.

For instance, the multicopter's battery voltage is an important parameter that needs continuous monitoring. The monitor tasks are essential for providing information to the supervisor tasks.

Supervisors are tasks that enable and disable other tasks according to the current vehicle state. For example, if the battery voltage is critical low, the vehicle supervisor can stop the current plan from executing, replacing it by a landing maneuver.

Transports are tasks in charge of forwarding messages between tasks using the message bus. Logging is a special transport task that listens to a set of messages and records their serialized states to persistent storage. Transport tasks are in charge of for example communication over different protocols, such as UDP, TCP or HTTP.

Each task is based on a set of inherited methods which form the common structure of the tasks. Although the structure is the same for each tasks, any of the methods can be overridden and customized for specific purposes. A set of the most important methods are given here:

- onResourceRelease() / onResourceInitialization()
- onUpdateParameters()
- onRequestActivation() / onRequestDeactivation()
- onActivation() / onDeactivation()
- onMain()
- consume($< M >$ message)
- dispatch($< M >$ message)

The methods are triggered at a specified time phase, periodically or whenever a specific message receives. For example, *onResourceInitialization* and *onResourceRelease* are methods that follow timed phases in the life-cycle. The *onMain* method is called periodically by the scheduler and executes something at timed delays. The methods *onActivation* and *onDeactivation* can, as requested by other tasks, turn on and off a payload module to save energy or start and stop a code sequence that use a lot of computational effort.

Tasks can also implement *onUpdateParameters* that is triggered whenever their configuration gets changed. Finally, all tasks can *consume* messages generated by any other tasks by implementing consume methods that listen for specific types of messages. Correspondingly can all tasks *dipatch* messages to any other tasks that are listening for this type of message. The communication between each task is done using the message bus, which is responsible for forwarding IMC messages from the producer to all their registered receivers.

3.2.3 Configurations

All instances of DUNE run the same basis of code, but with different configurations. The different configurations are easily managed in configuration files where tasks are enabled together with their initial parameters. Using a referencing mechanism, a configuration file may include parts of other configuration files. This allows the creation of vehicle specific configuration files and makes it easy to add more functionality, just by adding a task. Further, each configuration can run with different profiles such as *Hardware* and *Simulation*. These profiles allow different tasks to be enabled whether real hardware should be connected or simulated sensor and actuator data should be produced.

As seen in the example configuration file in Listing 3.1, the *ArduCopter* task is defined with different TCP ports for different profiles. The basic UAV configuration file is included to run the required tasks for a UAV, in this case the *ntnu-hexa-001*. Tasks can also be defined to run for every configuration with the profile *Always* or they can be disabled by the profile *Never*.

Listing 3.1: Example of configuration file based on *ntnu-hexa-001.ini*.

```
[Require uav/basic.ini]

[General]
Vehicle                                  = ntnu-hexa-001

[Control.UAV.SARGuidanceController]
Enabled                                  = Always
Entity Label                             = Guidance Controller
Debug Level                              = Debug

[Control.UAV.ArduCopter/Hardware]
Enabled                                  = Hardware
Entity Label                             = Autopilot
TCP - Address                            = 127.0.0.1
TCP - Port                               = 9999

[Control.UAV.ArduCopter/Simulation]
Enabled                                  = Simulation
Entity Label                             = Autopilot
TCP - Address                            = 127.0.0.1
TCP - Port                               = 5760
```

Chapter 4

Multicopter Motion Control

A vital part of every multicopter is the autopilot. The autopilot or *automatic pilot* is a system that assists a human operator in controlling a vehicle and is used in a broad aspect of applications. The first automatic pilots, in the early days of aviation, was designed to reduce the pilot's work load and relieved the pilot from maintaining continuously attention of the aircraft. These autopilots could make the aircraft fly straight at a given heading. Autopilots have evolved significantly over time and today modern autopilots can execute complex maneuvers and enable the control of highly unstable vehicles, such as multicopters. To get an overview of how these autopilots work, they can be broken into three independent blocks denoted as the *guidance*, *navigation* and *control* (GNC) systems.

4.1 Guidance, Navigation and Control

Guidance, navigation and control deals with the design of systems to control the movement of devices or vehicles. The benefit from using GNC systems might include sophisticated features such as automatic or remote control, fuel optimization, minimum time navigation and collision avoidance and are essential in all autonomous systems.

A GNC system is usually constructed as three independent blocks but can also be represented by one block. Loose and tight coupling is a trade-off between modularity and high performance. A loosely coupled system is attractive in environments focused on development since this allows for software updates of single blocks at a time. To boost the performance of a GNC system it can be necessary to make it more tightly coupled with the possible effect of making it hard to differ the three blocks from each other. The GNC systems interact with each other through data and signal transmission as illustrated in Figure 4.1, where the APM autopilot is sketched. In the most basic form, an autopilot itself is a GNC system [Fossen, 2011].

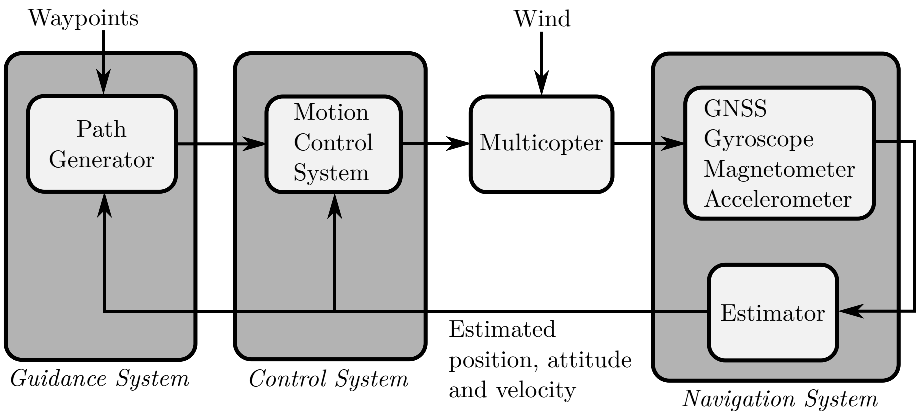


Figure 4.1: The APM autopilot system consists of a trajectory generator (guidance system), a gyrocompass/observer (navigation system) and a motion control system. Based on Fossen [2011].

4.1.1 Definitions

The definitions of guidance, navigation and control based on Fossen [2011] are as follow:

Guidance

Guidance is the action or the system that continuously computes the desired behavior of a vehicle, usually without direct or continuous human control. Typically, it is responsible for the calculation of the desired trajectory from the vehicle's current position to a designated target in addition to the determination of the needed changes in velocity, rotation and acceleration for following that path.

The guidance system takes input from sensors (the navigation system) and uses target information to form a desired control objective. External data such as the speed and direction of the wind can also be used as input in advanced guidance systems. A computer collects and processes the information, and then feeds the results to the flight control system. Advanced optimization techniques can be used to compute the optimal trajectory to follow.

Navigation

Navigation is the science of determining the movement of a vehicle at a given time. This includes continuously monitoring of the vehicle's position, velocity and acceleration as well as its attitude. A global navigation satellite system (GNSS) combined with motion sensors such as accelerometers and gyroscopes is a common navigational technique. GPS is a fully operational GNSS.

Control
Flight control or motion control refers to the manipulation of the necessary control forces and moments to be provided by the vehicle in order to satisfy a certain control objective while maintaining its stability. The desired control objective is usually seen in conjunction with the guidance system. Examples of control objectives are minimum energy, setpoint regulation, trajectory-tracking and path-following. The design of basic motion control systems is typically based on proportional, integral and derivative (PID) design methods while more advanced control systems is using optimal and nonlinear control theory.

4.2 APM:Copter Control Overview

The various flight modes in APM:Copter are all made up of the same underlying controllers. At the bottom of the controller stack there are four low-level controllers called *rate controllers* that are always running. These are responsible for providing a roll, pitch or yaw rate of change or, in the case of the throttle controller, a vertical acceleration. The low-level body-frame rate targets are given from a set of upper controllers for roll-pitch, yaw and throttle.

The upper controllers are responsible for passing a desired rate or acceleration to the rate controllers based on higher-level objectives. There are implemented a set of different controllers in APM:Copter for roll-pitch, yaw and throttle which can be combined to form a flight mode. For example, the flight mode *AltHold* utilize the automatic altitude hold controller to control throttle while the stabilization controller is used for both roll-pitch and yaw. To illustrate the flow of command to motor output, Figure 4.2 shows the command chain for roll control.

Given the background on how the controllers are structured in APM:Copter, it explains why the first release of "ArduCopter" had a second flight mode in addition to stabilize mode. The other mode, acrobatic or *acro mode*, is a rate control mode and thus already implemented in the control stack. In acro mode the pilot is directly controlling the rate of rotation in all axis meaning that the copter will remain in its current attitude if the RC sticks are released. This mode is the most difficult flight mode to master, but allows the pilot to perform flips and other acrobatic maneuvers.

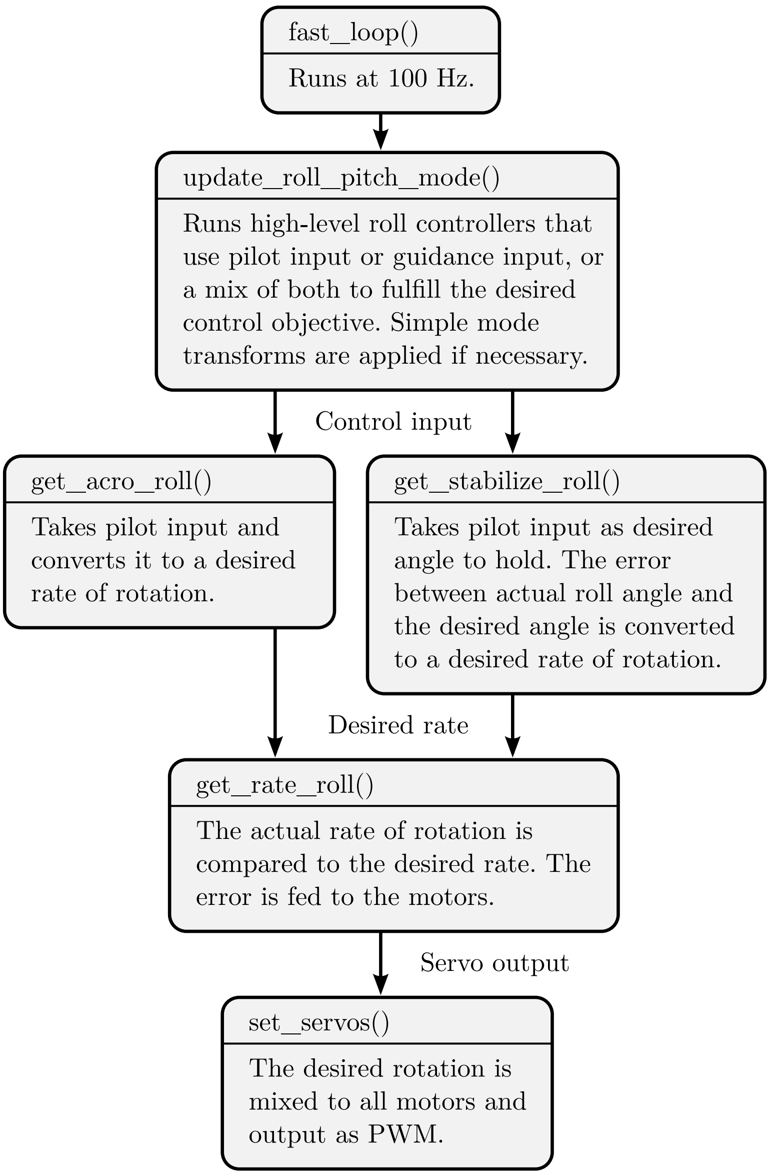

fast_loop()
Runs at 100 Hz.
update_roll_pitch_mode()
Runs high-level roll controllers that use pilot input or guidance input, or a mix of both to fulfill the desired control objective. Simple mode transforms are applied if necessary.
Control input
get_acro_roll()
Takes pilot input and converts it to a desired rate of rotation.
get_stabilize_roll()
Takes pilot input as desired angle to hold. The error between actual roll angle and the desired angle is converted to a desired rate of rotation.
Desired rate
get_rate_roll()
The actual rate of rotation is compared to the desired rate. The error is fed to the motors.
Servo output
set_servos()
The desired rotation is mixed to all motors and output as PWM.

Figure 4.2: The flow of command from user input to motor output for roll control in APM:Copter. Based on `http://ardupilot.com`

4.2.1 Attitude Control

The attitude of the copter is controlled in cascade by a nested PI and PID loop. In cascade control there are two control loops where the outer loop controls the setpoints to the inner loop. In the case of attitude control the outer and inner loops are the upper and low-level controllers, respectively, where the upper controllers set desired rates of angular rotation to the rate controllers (see Figure 4.3). The advantage with cascade control is due to the general response time in the system. It gives better control of the desired angle input, is less affected by disturbances and improves the dynamics. A disadvantage with cascade control is that the complexity of tuning increases.

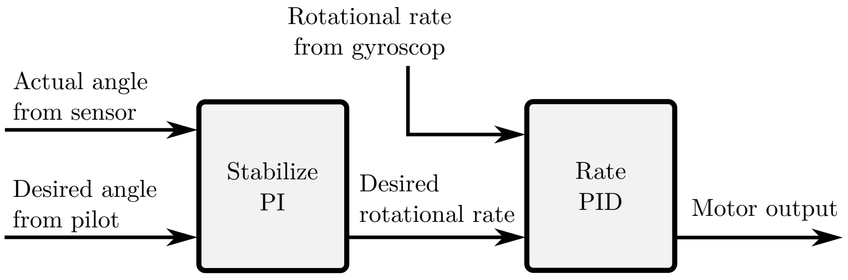


Figure 4.3: Cascaded PID structure of stabilized control per axis. Acro/rate control utilizes only the rate block where the pilot input the desired rotational rate directly. Based on `http://ghowen.me/build-your-own-quadcopter-autopilot`

Although there are many gains that can be tuned in APM:Copter to get optimal performance, the most critical is the proportional gain for roll and pitch rate. A well tuned rate controller should give good result for the other flight modes. Especially, stabilize mode should at least have good performance [APM, 2014b]. The tuned parameters in Table 4.1 showed good results for the 3DR ArduCopter Hexa.

Table 4.1: Tuned parameters for the 3DR ArduCopter Hexa.

Stabilize Roll		Stabilize Pitch		Stabilize Yaw		Loiter	
P	4.0	P	4.0	P	3.0	P	1.0
I	0.0	I	0.0	I	0.0		
Rate Roll		**Rate Pitch**		**Rate Yaw**		**Rate Loiter**	
P	0.138	P	0.138	P	0.20	P	1.0
I	0.099	I	0.099	I	0.02	I	0.4
D	0.020	D	0.020	D	0.00	D	0.0

4.2.2 Mission Commands

The APM:Copter has two different types of commands that can be executed in a mission. The first group affects the location of the vehicle while the second can be referred to as “do” commands which enable auxiliary functions. *Return-to-launch* (RTL) is one of the location commands that brings back the copter to a specified location while *set-cam-trigg-dist* is an example of a do command that trigger the camera shutter at regular distance intervals. All these mission commands can be run as part of a mission plan in *auto* flight mode. A list of some available mission commands are given in Table 4.2.

Table 4.2: Some of the available mission commands in APM:Copter. The location commands should be self-explainable.

Location command	Do command	Do command explanation
Takeoff	Condition-Delay	Delays the start of the next “do” command a number of seconds.
Waypoint	Condition-Dist	Delays the start of the next “do” command a given distance.
Loiter	Condition-Yaw	Point the nose of the vehicle at a given yaw angle.
RTL	Set-ROI	Points the nose of the vehicle and gimbal at the “region of interest”.
Land	Change-Speed	Change the target horizontal speed.
Circle	Set-Home	Change the home position used by RTL.
	Set-Servo	Move a servo to a particular PWM value.
	Set-Relay	Set the relay pin’s voltage high or low.
	Set-Cam-Trigg-Dist	Trigger the camera shutter at regular distance intervals.
	Digicam-Control	Trigger the camera shutter once.

Some of the mission commands are available through a designated flight mode, such as loiter and guided mode with the point-and-click waypoint guidance. Flight modes can be customized to yield specific guidance laws by changing the desired controllers in for example roll-pitch and yaw.

All mission commands must be pre-programmed into a mission script and sequentially executed. This means that the fully autonomous multicopter is best designed with an external on-board unit sending new mission commands based on sensor input from for example a camera module.

4.3 Motion Control Scenarios

The motion control system works in close interaction with the guidance system in order to achieve a given control objective. The outline of the control objectives varies from different scenarios and requirements. Fossen [2011] distinguishes between *setpoint regulation*, *path-following* and *trajectory tracking* scenarios. The latter uses methods to compute a desired trajectory to the target where a control system that forces the system output to track the desired output solves a trajectory tracking problem. The advantage of using trajectory tracking is to incorporate constraints due to for example obstacle avoidance or minimum and maximum time problems by generating feasible trajectories. Such optimization is not considered in the setpoint regulation and path following scenarios, which makes them simpler to work with. Trajectory tracking is therefore not considered further in this thesis keeping the focus on the two other scenarios, setpoint regulation and path following, that are now presented:

Setpoint regulation
The most basic guidance system is a special case where the desired position and attitude are chosen to be constant. Examples of setpoint regulation are constant altitude, attitude and speed control.

Path following
Path following is the task where the objective is to follow a predefined path independent of time. No restrictions are placed on the temporal propagation along the path. The path following problem can be divided into *straight-line paths* and *curved paths*. For the straight-line path problem line-of-sight (LOS) guidance is a popular method for heading control. LOS guidance uses a method for tracking a path between to waypoints (see Section 4.4). The path-following controller for curved paths is a *kinematic controller* that generates the desired states for the motion control system using the parameterization of the path. The drawback is that the path must be parameterized and known in advance. In many cases this is not practical and a simpler path consisting of waypoints and straight lines is preferred. A search pattern based on waypoints are illustrated in Figure 4.4.

For a multicopter and other aircraft, the guidance and control system usually consists of an attitude control system and a path-following control system. The main function of the attitude feedback control system is to maintain the multicopter in a desired attitude by controlling roll, pitch and yaw. The task of the path-following controller is to keep the multicopter on the pre-described path with some predefined dynamics, for instance a speed control system generating orders to the attitude control system. Supplementing the guidance and control system with an altitude controller makes it complete.

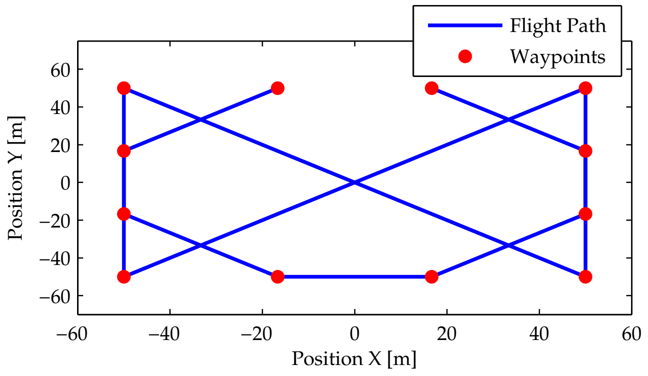


Figure 4.4: Straight lines forming a *barrier patrol search* used for waypoint guidance. Based on Andersen [2014].

4.4 Guidance Strategies

There are different guidance strategies for achieving a given control objective. A guidance strategy is a set of rules that make the control objectives be fulfilled in a certain way. LOS guidance belongs to a three-point guidance scheme and can be explained with a typically stationary reference point in addition to the location of the UAV and the target (see figure Figure 4.5). Suppose there is a straight line between the reference point and the target, the LOS guidance principle is based on the fact that the UAV is supposed to achieve an intercept of this straight line by constraining its motion along the line. LOS guidance can be applied to both track targets and paths.

A more straightforward guidance strategy is the pure pursuit guidance, which belongs to a two-point guidance scheme. It involves the UAV aligning its velocity pointing directly at the target. This strategy is equivalent to a predator chasing a prey in the animal world, and results very often in a tail chase when a moving object is targeted. The pure pursuit guidance is illustrated in Figure 4.5. The APM:Copter uses pure pursuit for the waypoint guidance.

4.4.1 Using DUNE as a Guidance Framework

Since DUNE has been adapted and already is utilized for autonomous control in Chapter 6, it is naturally to also use DUNE as a guidance framework if the waypoint scheme in APM:Copter should be discarded for an optimized guidance scheme. This will give the possibility for customizing a guidance scheme specially designed for the search and rescue operations. There are developed several guidance strategies in DUNE for use in autonomous underwater vehicles (AUV), which can be used as a good starting point for UAVs. Integral LOS and pure pursuit are both implemented and although they are not tested, it should be fair

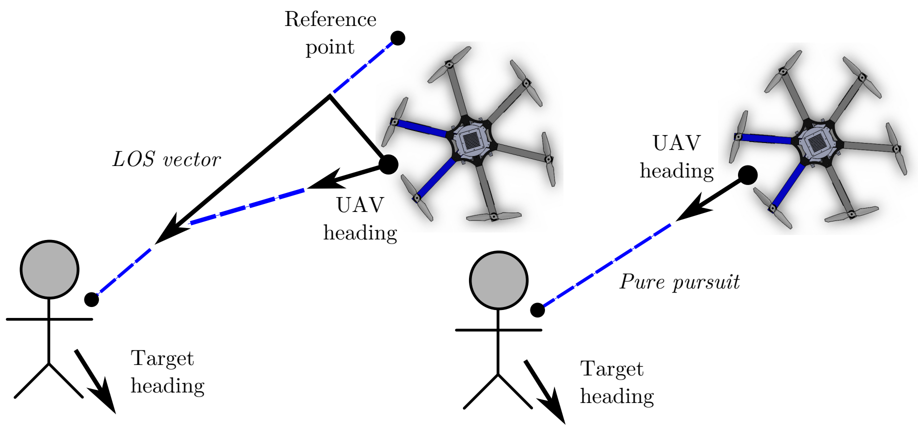


Figure 4.5: Illustration of two- and three-pointed guidance schemes. The vector pointing directly at the target represents the pure pursuit guidance principle while the line-of-sight guidance is represented with the reference point and the LOS vector. Based on Fossen [2011].

to assume that they would be working straightforward also with UAVs.

The advantage of introducing DUNE as a guidance framework is that all guidance can be done by the same unit giving more freedom for customizing with respect to the SAR operation. However, along with more responsibility it increases the importance of a well functional platform. The demand with respect to latency between DUNE and APM:Copter will increase, since more low-level control are to be done in DUNE. The low-level control interface presented in Chapter 5 can be utilized as the bridge between DUNE and APM:Copter. Although the performance of customized guidance control not has been tested in a search and rescue operation, the interface is utilized by Høglund [2014] and Voldsund [2014] in other applications showing good results.

4.5 Navigation in APM:Copter

The navigation system in APM:Copter is responsible for determining the position, velocity and angular orientation of the multicopter. The APM is equipped with gyroscope, accelerometer, magnetometer and barometer to estimate the movements. The important libraries for navigation in APM are AP_AHRS and *AP_InertialNav*. The AHRS or *attitude heading reference system* library is responsible for estimating the attitude and angular orientation. The InertialNav or INS library blends accelerometer data with GPS and barometer data to improve altitude and position hold. Since accelerometer values are integrated over time to approximate velocity and position, inaccuracy of these estimates grow due to sensor noise. The barometer and GPS readings are used to improve this accuracy by calculating an error value between the last position estimation and the measurement from the sensors. This value is then weighted with a gain factor and incorporated into the new estimation.

4.5.1 Compass Challenges

Loiter, RTL, auto and other flight modes based on waypoint commands are dependent on good navigation performance. However, magnetic interference, vibrations and bad GPS positions are well known factors that will influence the performance.

My experiments (see Chapter 9) faced some of these factors in the start-up phase of the experiments in field. They encountered in loiter mode and made the copter starting to spin in circles with a fixed yaw. The problem is typically due to the compass. A way to solve this problem is to run *compassmot*, a setup that helps compensate for magnetic interference from the power-distribution-board, wires, electronic speed controllers, motors and batteries. It calculates a compass offset based on the fact that the interference is linear with current drawn. This makes it possible to compensate for the interference during flight, regarded that it is not to high. Another solution is to move the compass away from the affected area using a external compass module. This can be done on both APM 2.5 and APM 2.6, but as APM 2.5 is equipped with an on-board compass the circuit board needs to be modified in the installation of the stand-alone compass. The APM 2.6 can use the combined GPS and compass module directly. This is known to normally resolve the compass problem. Other possibilities that can cause compass problems include bad compass offsets due to the live calibration process or incorrect compass orientation.

However, after all normal ways of solving the problem was tested, it still occurred. The final solving factor was most likely a reset of static memory of the APM:Copter called the EEPROM or *electrically erasable programmable read-only memory* in addition to a recalibration of the complete system. Without knowing the exactly reason behind the problem, a conclusion can not be stated, but a reset of the EEPROM together with a calibration of the entire system can definitely be regarded as two important factors.

4.5.2 Extended Kalman Filter

The next generation of autopilot systems, such as the PX4 and Pixhawk, feature faster processors, multithreading and a Unix/Linux-like programming environment. This has enabled for developing more advanced estimation algorithms to estimate the angular orientation, velocity and position of the multicopter. Based on the initial work by Riseborough [2014], an extended Kalman filter (EKF) algorithm that uses rate gyroscopes, accelerometer, magnetometer, GPS, airspeed and barometric pressure measurements has been developed. This algorithm is implemented in the *AP_NavEKF* library.

Compared to the simpler complementary filter algorithms as the AHRS and INS library, the EKF algorithm can fuse all available measurements and is therefore better able to reject measurements with significant errors APM [2014b]. This can make the vehicle less susceptible to faults that affect a single sensor. The EKF algorithm is also able to estimate offsets in the vehicles magnetometer readings and estimate the earth's magnetic field. This makes it less sensitive to compass calibration errors which may have been the source to the unknown problems in the field tests conducted. In addition, the EKF provides greater support for adding more sensors to provide further improvements in accuracy and robustness. Optional sensors such as optical flow and laser range finders are examples of sensors that could be utilized to give better performance.

Chapter 5

Extensions to APM:Copter

The existing features in APM:Copter widely cover the usage for most pilots with the support for both RC control and waypoint mission planning. However, in systems where a higher level of autonomy are wanted utilizing input from external systems or sensors, a feature allowing low-level guidance control of the copter is needed.

In this work, it has been clear that several applications can benefit from the development of low-level guidance control. Attitude control where roll, pitch and yaw can be controlled individually in addition to velocity control in body or earth frame has been found attractive as control objectives. Seen in the perspective of the search and rescue mission, velocity control is most likely the easiest way to do motion control of the multicopter in a fully autonomous task based on camera input. This makes it easy to command the copter to whether move forward or sideways regarded to the body frame. The velocity control combined with the existing waypoint guidance in APM:Copter makes a good platform for doing guidance control in SAR operations. However, if a separate velocity controller is desirable or the APM:Copter waypoint guidance scheme is not found suitable, attitude control is valuable.

This chapter presents contributions to APM:Copter and how the interface can be utilized using the MAVLink protocol which is intended and adapted for enabling low-level control from DUNE.

5.1 Low-Level Guidance Control

To make the extensions as modular as possible in order to simplify the further process with additional updates and make the work transferable to new

APM:Copter releases, a new flight mode is created for the low-level control. This is in line with the structure in APM:Copter where flight modes are defined with specialized roll-pitch, yaw and throttle controllers. As DUNE is utilized as the runtime environment in the SAR multicopter system and will be used to send MAVLink commands to APM:Copter, the flight mode for low-level control is called *DUNE mode*. The following code in Listing 5.1 is given as an example of how the mode can be selected autonomously using the MAVLink protocol.

Listing 5.1: Example of how to set new mode in ArduCopter.

```
uint8_t buf[512];
mavlink_message_t* msg = new mavlink_message_t;
mavlink_msg_set_mode_pack(255, 0, msg,
                          m_sysid,
                          1,
                          CP_MODE_DUNE);
uint16_t n = mavlink_msg_to_send_buffer(buf, msg);
sendData(buf, n);
```

Note that `CP_MODE_DUNE` is an integer enumerator with the APM:Copter modes, meaning that every flight modes can be selected in this manner. This is the equivalent message that is sent to the APM when a mode is selected in Mission Planner or APM Planner.

5.1.1 The Interface for Low-level Control

The interface for enabling low-level control of APM:Copter is based on a single MAVLink message for all kind of commands. To generalize and make the interface applicable for every system that want to utilize some kind of low-level control, the interface is designed to make it possible to target different controllers based on individual needs. To be able to use the same message for each individual needs, a target mask is used to distinguish the control objective of the message from each other. The different controllers that can be combined is given in Table 5.1.

Table 5.1: APM target masks used in the low-level control interface. A mask set to none means that the correspondent target is unchanged and will not be updated.

Velocity in X-Y	Throttle	Roll-pitch	Yaw
None	None	None	None
Velocity in BODY	Altitude	Roll and pitch angle	Yaw angle
Velocity in NED	Thrust		RC
	RC		

The MAVLink protocol is build with all kind of scenarios in mind and holds a large library with different message types. However, there do not exist a specific message that fits the description for sending velocity, throttle, roll-pitch and yaw control commands in the same task. This can be solved in two ways, either there is a new message type made in order to get the description right or an existing message is chosen that will work flawlessly but will fail describing the message sent. Although it is a straightforward process to add new message definition in MAVLink, this introduce changes to the APM:Copter that strictly speaking is unnecessary. In line with the idea of keeping the contributions to APM:Copter as simple and portable as possible, the existing *6DOF setpoint* message type is chosen for sending desired low-level control commands. It is originally designed for sending setpoints to a system with 6 degrees of freedom (DOF).

The 6DOF setpoint message is structured with six *float* variables and a single *uint8* variable. When a message is sent to the APM, the floats store the desired control variables while the uint8 is used to hold the target mask. An example of how the 6DOF setpoint message can be sent is given in Listing 5.2.

Listing 5.2: Example of how to send low-level control commands.

```
uint8_t buf[512];
mavlink_message_t* msg = new mavlink_message_t;
mavlink_msg_setpoint_6dof_pack(255, 0, msg,
                               target_mask,
                               trans_x, // Translation in x
                               trans_y, // Translation in y
                               trans_z, // Translation in z
                               roll,    // Roll angle
                               pitch,   // Pitch angle
                               yaw);    // Yaw angle
uint16_t n = mavlink_msg_to_send_buffer(buf, msg);
sendData(buf, n);
```

The modular design of the interface becomes convenient in a development phase where it is practical to have the possibility of testing different parts individually, doing debugging easier. For example, lets assume that a system is developed with autonomous controllers designed for thrust, roll-pitch and yaw. In the test phase, if a problem occur, it can be hard to determine which control system that causes the problem. Each controller can easily be tested individually by changing the target mask. To test the roll-pitch controller individually, the target mask for throttle and yaw can be set to RC making the pilot in control of these.

All external communication to the APM through the MAVLink protocol is handled in the source file *GCS_Mavlink.pde* in APM:Copter. The file is structured with several switch-cases where one of them handles MAVLink communication. The segment of *GCS_Mavlink.pde* in Listing 5.3 shows the principle of how the the target mask is used to distinguish the different controllers, in this case the yaw target. Note how the different yaw controllers are set with the *set_yaw_mode()* function.

Listing 5.3: Segment of *GCS_Mavlink.pde* where yaw control is managed.

```
void GCS_MAVLINK::handleMessage(mavlink_message_t* msg) {
    switch (msg->msgid) {
    case MAVLINK_MSG_ID_SETPOINT_6DOF:
        switch (packet.target_system & TARGET_MASK_YAW) {
            case TARGET_YAW_NONE:
                break;
            case TARGET_YAW_SET:
                if (mode_yaw != TARGET_YAW_SET){
                    mode_yaw = TARGET_YAW_SET;
                    set_yaw_mode(DUNE_YAW_SET);
                }
                yaw_cd = (int32_t)(ToDeg(packet.rot_z * 100.0));
                break;
            case TARGET_YAW_RC:
                if (mode_yaw != TARGET_YAW_RC){
                    mode_yaw = TARGET_YAW_RC;
                    set_yaw_mode(DUNE_YAW_RC);
                }
                break;
```

5.2 Security

The possibility of controlling all multicopter parameters using the MAVLink protocol creates a security breach that can bring along chances for unwanted control by strangers. Potentially, other system can transmit packets with the purpose of gaining control of the aircraft. The APM:Copter is not only vulnerable for hackers, but also for unintentionally commands sent from your own GCS that can make the aircraft fly away or crash. This can accidentally happen when for instance the operator is exploring the functionality in Mission Planner and by mistake sends a flight command to the vehicle, which is probably more likely to be a bigger problem than hackers. All in all, it is essential to have a system that can deal with these security threats.

Based on a episode from early testing where unintentionally flight commands was given to an aircraft, it became clear that a security guard should be made making the system unreachable in a specific flight mode. This is the origin to an designated secure flight mode named *dictator mode*. This flight mode is basically the original stabilize mode without the interface against MAVLink messages, and any communication attempts will be denied as illustrated in Figure 5.1. Using this, the pilot at any time switch to dictator mode, regaining manual control over the UAV even though Mission Planner or DUNE are spamming the system with MAVLink messages.

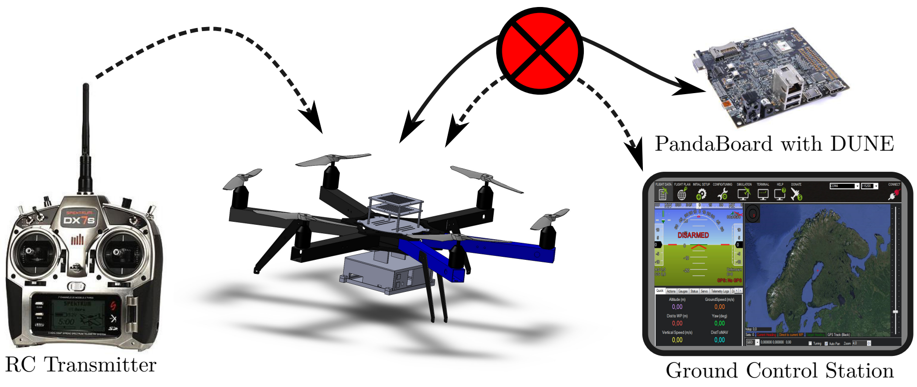


Figure 5.1: The principle of *dictator mode* where all MAVLink messages are rejected.

Although the dictator mode was developed to cut off all communication to and from a ground station for security reasons, it introduced a new situation regarding problems with reading parameters. Since all MAVLink messages were disabled, it made the pilot unable to receive any vital flight information from the copter. It may therefore be more practical adjust the flight mode to allow reading of parameters, but deny writing.

Chapter 6

Autonomous Behavior in Flight

The fully autonomous search and rescue multicopter is a system that, on its own, can scan areas for objects, and identify and report findings in addition to perform actions as takeoff, obstacle avoidance and landing. The cornerstone in such a system is an on-board unit that can perform autonomous guidance and control.

Autonomous means having the power for self-government. The role of the autonomous guidance and control system is to act as an extension of the human operators to assure reliable and continuous operations of the UAV over an extended period of time. This means that the UAV should be able to perform well under significant uncertainties in the system and environment.

The serious challenge in fully autonomous UAVs is the real-time optimization in uncertain environment without human intervention. The perfect system should be able to deal with unexpected situations, new control tasks, and failures within limits. Since search and rescue missions introduce unstructured environments, uncertainties, complex situations and never have two identical missions, the robust system is a big challenge.

6.1 Autonomous Guidance and Control of the Multicopter

In this thesis, autonomous guidance control is distinguished in two different scenarios. The first one relies only on input from the navigation system during the flight of a pre-programmed route, and can perform predefined actions triggered by time or distance. The second is based on real-time input from a camera. This requires an on-board unit to process the video stream and further calculate a desired behavior to be executed, referred to as a fully autonomous system. The structure on how a fully autonomous search and rescue mission is conducted from the perspective of guidance and control is illustrated in Figure 6.1.

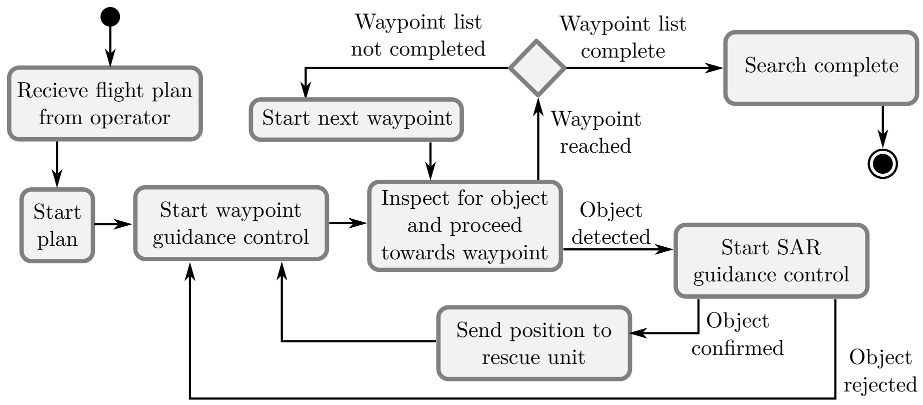

Recieve flight plan from operator
Start plan
Start waypoint guidance control
Waypoint list not completed
Start next waypoint
Inspect for object and proceed towards waypoint
Waypoint reached
Waypoint list complete
Search complete
Object detected
Start SAR guidance control
Object confirmed
Send position to rescue unit
Object rejected

Figure 6.1: State diagram of how guidance control is conducted in a SAR mission.

6.1.1 Waypoint Guidance Scheme

The functionality for a pre-programmed flight has good support in APM, where a waypoint scheme can be used for a given search pattern. A waypoint can be sent using the MAVLink protocol as shown in Listing 6.1.

Listing 6.1: Example of how to send waypoint to APM.

```
uint8_t buf[512];
mavlink_message_t* msg = new mavlink_message_t;
mavlink_msg_mission_item_pack(255, 0, msg,
                    sysid,                // System ID
                    0,                    // Component ID
                    1,                    // Sequence
                    MAV_FRAME_GLOBAL,     // Coordinate system
                    MAV_CMD_NAV_LOITER_UNLIM, // Scheduled action
                    2,                    // Current
                    0,                    // Autocontinue next WP
                    0,                    // Not used
                    0,                    // Not used
                    -1,                   // CCW loiter
                    0,                    // Not used
                    lat,                  // Latitude
                    long,                 // Longitude
                    alt);                 // Altitude
n = mavlink_msg_to_send_buffer(buf, msg);
sendData(buf, n);
```

To summarize the MAVLink message *mission item*, which is used for sending waypoints, the interesting parameters sent with every action are identification, four parameters and the 3D position of the copter given by latitude, longitude and relative altitude to home location. The four parameters can be some sort of custom action as for camera setting, camera trigger, loiter time, etc.

The example in Listing 6.1 illustrates how DUNE sends a waypoint packet to the APM. When a "GoTo" manuever or other waypoint-based maneuvers are selected in Neptus, it follows the specified command chain illustrated in Figure 3.2 where DUNE at the end sends the next waypoint to the APM. Andersen [2014] has done research on how different flight paths can be utilized in the search and rescue mission. Two of these search patterns are illustrated in Figure 6.2.

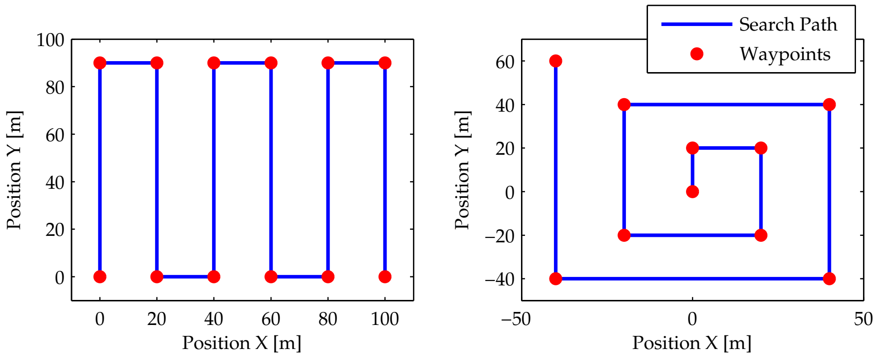
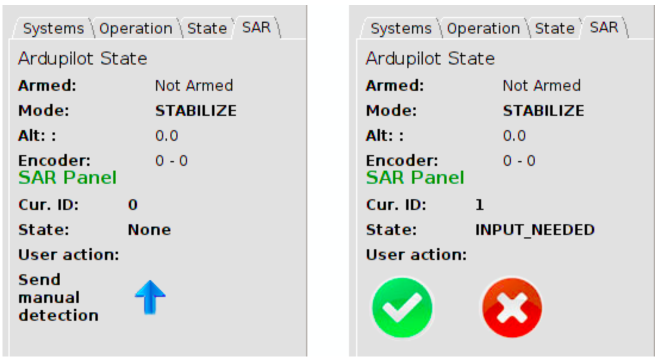


Figure 6.2: The *creeping line search* and the *square search*. Based on Andersen [2014].

6.1.2 The Fully Autonomous Guidance Scheme

The purpose of the fully autonomous guidance system is to be able to perform actions based on input regarding the object location. The control objective when a target is detected is important for identifying the object and for further search. A logical operation to do when a object is detected is to verify that the detection is correct. This can be done by closing in at the object taking close-up pictures, if possible, from different angles. A guidance system where the multicopter starts loitering whenever a object is found and further let an operator using manual control to inspect the detected object can itself be found interesting. However, the fully autonomous guidance system that automatically executes this operation, making human intervention redundant, completes the SAR multicopter system.

The motivation behind all development in this thesis is to provide support for the fully autonomous multicopter. Although computer vision algorithms has not been merged into this work, the framework is made so that this functionality easily can be included in the future. To substitute for the lack of a computer vision algorithm providing automatically object detection, a SAR panel is made in Neptus making manual object detection possible (see figure Figure 6.3). The panel sends the IMC message *detected object* with a flag indicating whether a object is detected, input is needed, a detection is confirmed or a detection is rejected.


Systems | Operation | State | SAR

Ardupilot State

Armed:	Not Armed
Mode:	**STABILIZE**
Alt: :	0.0
Encoder:	0 - 0

SAR Panel

Cur. ID:	**0**
State:	**None**
User action:	
Send manual detection	

Systems | Operation | State | SAR

Ardupilot State

Armed:	Not Armed
Mode:	**STABILIZE**
Alt: :	0.0
Encoder:	0 - 0

SAR Panel

Cur. ID:	**1**
State:	**INPUT_NEEDED**
User action:	

Figure 6.3: The search and rescue panel in Neptus for manual detection of objects. The panel to the left shows the button for sending the manual detection while the panel to the right shows how the object can be verified or rejected after a manual detection is sent.

When a detected object message with the “object is detected” flag is received at the message bus in DUNE it activates the *SAR guidance controller* task. The current flight mode in APM is set to *DUNE mode* and the vehicle is controlled by the task sending desired control messages using the interface for low-level control. The SAR guidance controller is in charge of control until a object is either confirmed or rejected. The SAR guidance controller is then deactivated and control is returned to the waypoint guidance scheme that continues the search. The structure of the SAR guidance controller is given in Listing 6.2 where the most important segments of the task are shown. The different control loops activate specific modules and function as a security barrier together as they activate the correspondingly control commands to be sent to the APM. In the example provided in Listing 6.2, fixed yaw and altitude control is conducted by sending (dispatching) the correspondingly desired IMC messages. This message is received (consumed) by a second important task for autonomous control, the *ArduCopter* task responsible for the two-ways communication between DUNE and APM. A small segment of ArduCopter shows how the yaw and altitude is handled in Listing 6.3

The SAR guidance controller is a draft to an autonomous guidance controller that can consume information from a computer vision task, calculate desired control commands and further send these to the multicopter. The ArduCopter task is based on the *Ardupilot* task from LSTS but customized and adapted to APM:Copter as the original is developed for APM:Plane.

Listing 6.2: Segment of *SARGuidanceController* for autonomous guidance.

```
namespace Control {
  namespace UAV {
    namespace SARGuidanceController {
      using DUNE_NAMESPACES;

      //! Controllable loops: Which loop makes this task starts
      static const uint32_t c_controllable = IMC::CL_GUIDANCE;
      //! Required loops: This task starts the task dependent on:
      static const uint32_t c_required = CL_YAW | CL_ALTITUDE;

      struct Task: public DUNE::Control::BasicGuidanceController
          {

        //!! Here goes a lot of methods and initialization.

        //! Control is computed when a EstimatedState messages is
             received
        void onEstimatedState(const double timestep, const IMC::
            EstimatedState* msg)
        {
          // Do SAR guidance control

          IMC::DesiredYaw yaw;
          yaw.value = Angles::radians(90);
          dispatch(vel);

          IMC::DesiredZ alt;
          alt.z_units = Z_ALTITUDE;
          alt.value = 30;
          dispatch(alt);
        }
```

Listing 6.3: Segment of *ArduCopter* for APM communication.

```
namespace Control {
  namespace UAV {
    namespace ArduCopter {
      using DUNE_NAMESPACES;

      //!! Here goes initalization of enumerators e.g:
      //!! APM_c_loops, APM_copterModes, APM_target_masks

      struct Task: public DUNE::Tasks::Task {

        //!! Here goes a lot of methods and initialization.

        void consume(const IMC::DesiredZ* alt) {
          if (!(m_cloops & IMC::CL_ALTITUDE)) {
           debug("altitude control of APM is NOT active");
           return;}
          // Update internal variable for alt control
          if(alt->z_units == Z_ALTITUDE) {
            //m_extCtrl.z = alt->value;
            //m_extCtrl.z_units = alt->z_units;
            m_d_z = *alt;
            update(ACL_ALTITUDE);
          }
        }
        void consume(const IMC::DesiredYaw* yaw) {
          if (!(m_cloops & IMC::CL_YAW)) {
           debug("yaw control of APM is NOT active");
           return;}
          // Update internal variable for yaw control
          m_d_yaw = *yaw;
          update(ACL_YAW);
        }
        bool update(APM_c_loops cloop) {
          if( m_updated_desired[ACL_YAW]) {
            target |= TARGET_YAW_SET;
            yaw    = m_d_yaw.value;
            //! Mark as sent and ready
            send = true;
            m_updated_desired[ACL_YAW] = false;
          }
          if( m_updated_desired[ACL_ALTITUDE]) {
            target &= ~ TARGET_MASK_Z;
            target |=   TARGET_Z_ALT;
            trans_z = m_d_z.value;
            // Mark as sent and ready
            send = true;
            m_updated_desired[ACL_ALTITUDE] = false;
          }
```

The high-level control objective of the fully autonomous guidance control task is to identify and report the position of objects. However, the task can be extended to encompass more advanced analysis. Imagine a search and rescue operation where several persons are missing in the same area and assumed to be moving. This increases the complexity for the search mission. The high-level control objective can now be extended to determine the position of all objects and preferably keep them as updated as possible. The SAR operation has developed to a complicated situation where object first should be detected and then kept under surveillance simultaneously as the ongoing search for more objects should continue.

To realize such an autonomous control task, several different parts are needed to fulfill the complete system. First of all, it is important to estimate the motion of the objects. An approach is presented in Chapter 7 where the motion of a target object is described and calculated. Furthermore, it is vital to integrate a sensor for determining the object's position. In the following section, the integration of a camera is discussed where the object is located in the image frame and real world coordinates are calculated.

One of the key advantages of estimating the real world coordinates of the tracked objects, is that if the tracking of the object is lost, the estimate of the last known real world position is still known. This makes the process of regaining tracking of the object easier as the UAV is more aware of where it should search for the lost target.

6.2 Camera Integration

The functionality of the camera in a SAR system is dual; it functions as a sensor device for recognition of objects, but also as a sensor for determining the location of the object. The payload designed by Leira [2013] performs real-time object detection and object tracking based on an infrared camera. The implemented object detection algorithm utilizes pre-trained classifiers to perform detection, which implies that the algorithm will be more robust for a specific camera angle. The implemented object tracking algorithm is based on an estimator that tracks objects in the camera frame. The results showed that the algorithm occasionally failed to keep track of the object during fast, abrupt and relatively large displacement rates in the position of the tracked targets. As the tracking algorithm was developed with a linear motion model, this is reasonable since abrupt change in velocity is not possible.

Since the natural movement of a human violate against large movement, the linear motion model is a fair assumption. However, as the tracking algorithm is working in the image frame, it will be sensitive to abrupt changes from the camera, even though the object is standing still. The limited computational power of the single-board computer implies that non-linear estimators are difficult to utilize. With this in mind, it is important to make the best working condition for the tracking algorithm by using a gimbal to stabilize the camera. The use of the gimbal will also simplify and increase the performance of the algorithm that determines the object's real world coordinates.

6.2.1 Image to Global Frame

Suppose that the object detection and object tracking work best with a camera angle at $-45°$, pointing forward to get a better visual view of the ground and downward to make it possible to determine an object position. This is easily conducted with the use of a gimbal. The part of the world that is visible through the camera is called field of view. Since the use of gimbal gives a fixed camera angle that is known it simplifies the calculation of how a pixel in the image frame is transferred to global coordinates. Assume that the tracked target is positioned at the ground. Furthermore, if the elevation on the ground is neglected, all tracked object can be estimated at the same altitude. Now, with the estimate of the object in the image frame together with an estimated altitude of the UAV, the process of estimating the real world coordinates of the object is a simple problem of triangulation (see figure Figure 6.4).

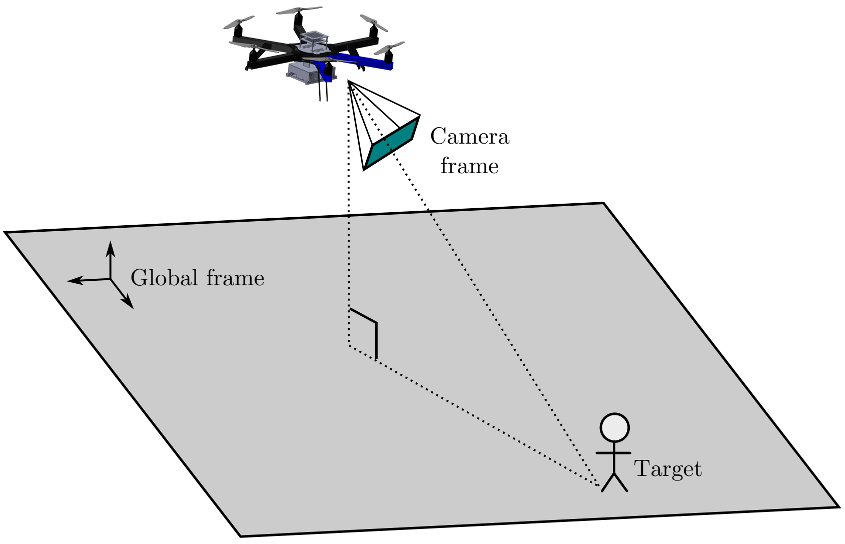


Figure 6.4: FOV and triangulation of object.

The less forward and more downward a camera is pointed at the ground, the less will elevation on ground affect the accuracy of the triangulation. This can be illustrated with a simple example. Lets assume that the camera is pointing straight downward at the ground where an object is localized in the center of the image frame. The position of the object and the UAV will then be exactly the same, only at different altitude. The disadvantage with the downward pointing camera is that it gives a small field of view compared to a horizontal camera position. However, a camera pointing straight forward looses depth vision, which means that a small error in the image frame can result in a large miscalculation of the position. As a result, there needs to be a compromise between field of view and the quality of the calculated object position.

With measurements of real world coordinates it should be noted that to keep good tracking performance in the image plane, it might be necessary to keep two separate estimates. That is, having one estimator for the position in the image plane, and another estimator for the real world coordinates.

6.2.2 Operational Altitude

The angle of view, or field of view (FOV) in a case of a camera, is expressed as the angular size of the view cone and is relative to the focal length of the camera. This means that objects outside the FOV when the picture is taken are not recorded in the photograph. The FOV of the camera and the search altitude of the multicopter are two important parameter in a search. These two parameters will together determine the size of the observation area. However, the FOV of the camera is a fixed parameter while the altitude can be changed in flight. This gives the opportunity to dynamically change the size of the observation area, which introduces a new control objective. High altitude gives a large observation area with low level of details while low altitude provides a small observation area with high level of details. The search speed can also affect the detection result at different altitudes, assuming that the frame rate of the computer algorithm is limited, since low altitude will give larger changes in the image frame than high altitude. These observations illustrate that altitude is important for optimizing the quality of the search process, and emphasize that the guidance problem is a three dimensional problem.

The guidance problem can also include temporal constraints by looking into search strategies that minimize the time to find targets. Environmental factors as wind and terrain can also affect the optimal guidance problem.

Chapter 7

Motion of Target Object

This chapter presents a method that describes the motion of a target object. The *measure-and-estimate* approach is a method that is able to run in real-time with limited computational power on-board the UAV hardware. This method estimates the states of an object into the future using measurements available at the present, and is referred to as the prediction step. Further is the estimated states corrected by the measured position of the object position recorded by the UAV which is referred to as the correction step. The extended Kalman filter (EKF) is introduced as a measure-and-estimate approach to estimate the position of an object detected by a UAV.

7.1 Problem Statement

A UAV detects a moving object and records its position based on sensor data. To be able to estimate the position of the object, the general dynamics of the object need to be modeled as a system of known dynamics. Knowing that the model used only is an estimation of the true object dynamics means that *model error* is introduced. Noise from the UAV sensor and the following algorithm used to determine the position of the object introduces *measurement error* to the problem. The problem describing the object motion is stated on a non-linear state space representation

$$\begin{aligned} \mathbf{X}(k+1) &= \mathbf{f}[\mathbf{X}(k)] + \boldsymbol{\omega}(k) \\ \mathbf{Y}(k) &= \mathbf{h}[\mathbf{X}(k)] + \boldsymbol{\nu}(k) \end{aligned} \tag{7.1}$$

where $\mathbf{X}$ is the state vector and $\mathbf{Y}$ is the output vector. The vectors $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$ are white-noise processes with zero cross correlation and represent the uncertainty of the model and measurements respectively. To estimate the states of the object detected by the UAV an extended Kalman filter (EKF) is proposed as a measure-and-estimate approach.

7.2 Kalman Filter

The Kalman filter is a recursive process introduced in 1960 by R. E. Kalman and was early recognized as new and important contribution to least square filtering. The linear as well as the extended Kalman filter are recursive processes divided into a prediction and a correction part. In the case of linear system dynamics, a Kalman filter will produce optimal estimates with respect to minimum variance. The EKF is an extension of this optimal state estimator, designed to account for the fact that most real systems inhabit nonlinear characteristics. This is more in line with the estimate-and-measure approach needed to describe the motion of the object. The discrete EKF is defined to further explain the theory behind the Kalman filter, which is based on the theory outlined by Brown & Hwang [2012].

7.2.1 Discrete EKF

In the extended Kalman filter, the state transition and observation (measurement) of the process are defined as the non-linear state-space representation in (7.1). The process and measurement noise $\boldsymbol{\omega}(k)$ and $\boldsymbol{\nu}(k)$ are defined as independent random variables, Gaussian distributed and described through the covariance matrices $\mathbf{Q}(k)$ and $\mathbf{R}(k)$:

$$E\left[\boldsymbol{\omega}(k)\boldsymbol{\omega}(k)^T\right] = \begin{cases} \mathbf{Q}(k) & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \tag{7.2}$$

$$E\left[\boldsymbol{\nu}(k)\boldsymbol{\nu}(k)^T\right] = \begin{cases} \mathbf{R}(k) & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \tag{7.3}$$

$$E\left[\boldsymbol{\omega}(k)\boldsymbol{\nu}(k)^T\right] = 0 \quad \text{for all } k \text{ and } i \tag{7.4}$$

These diagonal covariance matrices are suspect to tuning when the Kalman filter is implemented. The matrices represent assumptions made regarded to the system, and may be corrected if found necessary. They are therefore often referred to as the design matrices of a Kalman filter.

Assume now that at some point in time $t(k)$, the initial estimate of the process is given, and that this estimate is based on all our knowledge about the process prior to $t(k)$. This prior (or a priori) estimate will be denoted as $\hat{\mathbf{X}}^-(k)$ where the "hat" denotes estimate, and the "super minus" denotes that this is our best estimate prior to assimilating the measurement at $t(k)$. Further, lets assume that the error covariance matrix associated with $\hat{\mathbf{X}}^-(k)$ is known. By defining the estimation error as

$$\mathbf{e}^-(k) = \mathbf{X}(k) - \hat{\mathbf{X}}^-(k) \tag{7.5}$$

the associated error covariance matrix is defined

$$\mathbf{P}^-(k) = E[\mathbf{e}^-(k)\mathbf{e}^-(k)^T] = E[(\mathbf{X}(k) - \hat{\mathbf{X}}^-(k))(\mathbf{X}(k) - \hat{\mathbf{X}}^-(k))^T] \tag{7.6}$$

As the prior estimate $\hat{\mathbf{X}}^-(k)$ is now defined, the measurement $\mathbf{Y}(k)$ is used to improve the prior estimate by a linear blending factor $\mathbf{K}(k)$. The updated estimate $\hat{\mathbf{X}}(k)$ is then derived by the update equation

$$\hat{\mathbf{X}}(k) = \hat{\mathbf{X}}^-(k) + \mathbf{K}(k)\Big(\mathbf{Y}(k) - \hat{\mathbf{Y}}^-(k)\Big) \tag{7.7}$$

where the optimized linear blending factor is called the *Kalman gain*. A high blending factor makes the estimates follow the measurements closely. While a low blending factor emphasizes the modeled behavior, and thus makes the estimates become less responsive to new inputs.

Note that Equation (7.7) is written in terms of total rather than incremental quantities as in Equation (7.13), which is the more familiar linear estimate update equation. The equations says that the a priori estimate is corrected by adding the measurement residual appropriately weighted by the Kalman gain $\mathbf{K}(k)$.

The linear blending factor $\mathbf{K}(k)$ must now be derived in a way that makes the updated estimate optimal. The Kalman filter is optimal in most cases that makes sense, but is mostly known as optimal with respect to *minimum variance* [Vik, 2012]. Toward this optimization problem, the error covariance matrix associated with the updated (a posteriori) estimate is now defined

$$\mathbf{P}(k) = E[(\mathbf{X}(k) - \hat{\mathbf{X}}(k))(\mathbf{X}(k) - \hat{\mathbf{X}}(k))^T] \tag{7.8}$$

The Kalman filter solves the optimization problem that gives the particular $\mathbf{K}(k)$ that minimizes the mean-square estimation error. The estimation error variances for the elements of the state vector being estimated is the individual terms along the major diagonal of $\mathbf{P}(k)$. The optimization using a differential calculus approach leads further to an optimal Kalman gain $\mathbf{K}(k)$. The reader is referred to Brown & Hwang [2012] for more detailed derivation of the Kalman gain. To summarize the extended Kalman filter, the equations for the prediction and correction part are now listed.

Extended Kalman Filter Equations

Design matrices:

$$\mathbf{Q}(k) = \mathbf{Q}^T(k) > 0, \quad \mathbf{R}(k) = \mathbf{R}^T(k) > 0 \tag{7.9}$$

Initial conditions:

$$\mathbf{X}(0) = \mathbf{X}_0 \tag{7.10}$$

$$\mathbf{P}^-(0) = E[(\mathbf{X}(0) - \hat{\mathbf{X}}^-(0))(\mathbf{X}(0) - \hat{\mathbf{X}}^-(0))^T] = \mathbf{P}_0 \tag{7.11}$$

Corrector equations:

$$\mathbf{H}(k) = \frac{\partial \mathbf{h}}{\partial \mathbf{X}}\Big|_{\mathbf{X}=\hat{\mathbf{X}}^-(k)} \tag{7.12}$$

$$\hat{\mathbf{X}}(k) = \hat{\mathbf{X}}^-(k) + \mathbf{K}(k)\left[\mathbf{Y}(k) - \mathbf{h}\left(\hat{\mathbf{X}}^-(k)\right)\right] \tag{7.13}$$

$$\mathbf{K}(k) = \mathbf{P}^-(k)\mathbf{H}^T(k)\left[\mathbf{H}(k)\mathbf{P}^-(k)\mathbf{H}^T(k) + \mathbf{R}(k)\right]^{-1} \tag{7.14}$$

$$\mathbf{P}(k) = \left[\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)\right]\mathbf{P}^-(k)\left[\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k)\right]^T + \mathbf{K}(k)\mathbf{R}(k)\mathbf{K}^T(k) \tag{7.15}$$

Predictor equations:

$$\mathbf{\Phi}(k) = \frac{\partial \mathbf{f}}{\partial \mathbf{X}}\Big|_{\mathbf{X}=\hat{\mathbf{X}}(k)} \tag{7.16}$$

$$\hat{\mathbf{X}}^-(k+1) = \mathbf{f}[\hat{\mathbf{X}}(k)] \tag{7.17}$$

$$\mathbf{P}^-(k+1) = \mathbf{\Phi}(k)\mathbf{P}(k)\mathbf{\Phi}^T(k) + \mathbf{Q}(k) \tag{7.18}$$

Note that the symmetric form of the **P**-update is used in Equation (7.15). In some cases this form is sufficient to ward off divergence [Brown & Hwang, 2012].

7.2.2 Analysis and Functionality of the EKF

The EKF is mentioned as the nonlinear version of the Kalman filter, but it is important to remember that the EKF is working in the world of linear dynamics. The EKF linearizes about the trajectory that is continually updated with the state estimate resulting from the measurement, as shown in Figure 7.1. The filter gain sequence will depend on the sample measurement sequence realized on a particular run of the experiment. Thus, the gain sequence is not predetermined by the process model assumptions as in the usual Kalman filter.

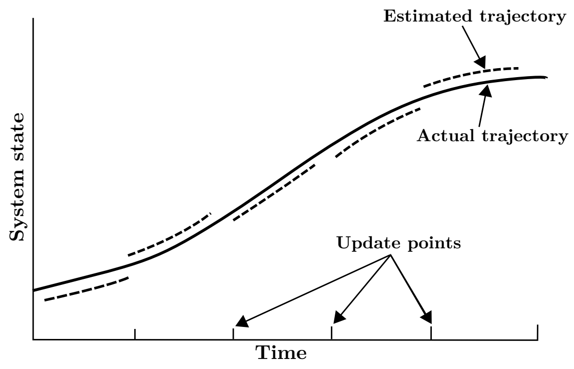


Figure 7.1: Reference and actual trajectories for an EKF. Based on Brown & Hwang [2012].

As discussed by Brown & Hwang [2012] the general analysis of the EKF is difficult because of the feedback of the measurement sequence into the process model. However, it is possible to discuss how the update of the state estimate is done. The estimated trajectory, which the EKF uses, is only statistical the best trajectory. There is a chance (and maybe a good one) that the updated trajectory will be a poor estimate of the true states. This could eventual lead to divergence of the filter. Especially in situations where the initial uncertainty and measurement errors are large this could be a problem. In general do not the extended Kalman filter have the optimal properties of the linear Kalman filter, but is still used extensively in practice with good results.

7.3 State Estimation

The extended Kalman filter is now presented as an approach to estimate the states of a moving object. For such Kalman filtering, a non-linear state-space representation as presented in (7.1) is required to describe the object motion. This section presents the model used for the state estimation based on Prevost et al. [2007].

7.3.1 Variable Definition

Let the motion of the object be modeled as a series of setpoints $\mathbf{X}_r$ where the dynamics are modeled as a transfer function $\mathbf{G}(z)$ and a set of motion equations further provide the positions. The variables are shown in Figure 7.2, which illustrates the model.

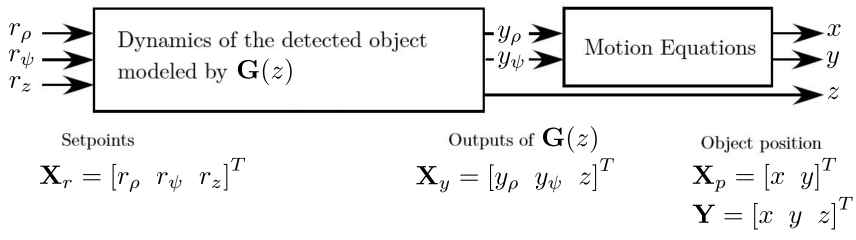

r_ρ
r_ψ
r_z
Dynamics of the detected object modeled by $\mathbf{G}(z)$
y_ρ
y_ψ
Motion Equations
x
y
z
Setpoints
$\mathbf{X}_r = [r_\rho \;\; r_\psi \;\; r_z]^T$
Outputs of $\mathbf{G}(z)$
$\mathbf{X}_y = [y_\rho \;\; y_\psi \;\; z]^T$
Object position
$\mathbf{X}_p = [x \;\; y]^T$
$\mathbf{Y} = [x \;\; y \;\; z]^T$

Figure 7.2: Motion model of object under observation. Based on Prevost et al. [2007].

The states of the model is provided by the

Object setpoints:
Speed in the xy-plane r_ρ, heading in the xy-plane r_ψ, altitude r_z

Outputs of model $\mathbf{G}(z)$:
Speed in the xy-plane y_ψ, heading in the xy-plane y_ψ, altitude z

Corresponding position:
x-axis x, y-axis y

The transfer function $\mathbf{G}(z)$ models the response of the object dynamics. If the UAV can identify the detected object, known responses can be used to better approximate the true underlying object dynamics. Since the transfer function needs to be known, general dynamics must be chosen if the UAV is unable to identify the detected object. The estimated internal states $\mathbf{X}_m$ of $\mathbf{G}(z)$ must be included in the state vector for Kalman filtering.

The object setpoints, the outputs of model $\mathbf{G}(z)$ and the corresponding position are respectively given as the vectors

$$\mathbf{X}_r = [r_\rho \;\; r_\psi \;\; r_z]^T \tag{7.19}$$

$$\mathbf{X}_y = [y_\rho \;\; y_\psi \;\; z]^T \tag{7.20}$$

$$\mathbf{X}_p = [x \;\; y]^T \tag{7.21}$$

The state vector in order to completely characterize the behavior of the detected object is thus given as

$$\mathbf{X} = \left[\mathbf{X}_r^T \;\; \mathbf{X}_m^T \;\; \mathbf{X}_y^T \;\; \mathbf{X}_p^T\right]^T \tag{7.22}$$

The position of the object is given by the spatial coordinates (x, y, z) which form the measurement vector $\mathbf{Y}$:

$$\mathbf{Y} = [x \;\; y \;\; z]^T \tag{7.23}$$

Finally, let the random vector $\boldsymbol{\omega}$ be defined

$$\boldsymbol{\omega} = \left[\boldsymbol{\omega}_r^T \;\; \boldsymbol{\omega}_m^T \;\; \boldsymbol{\nu}_m^T \;\; \boldsymbol{\omega}_p^T\right]^T \tag{7.24}$$

where $\boldsymbol{\omega}_r$, $\boldsymbol{\omega}_m$, $\boldsymbol{\nu}_m$ and $\boldsymbol{\omega}_p$ are random vectors composed by independent, zero-mean, white noise, Gaussian random variables specified as follows:

$\boldsymbol{\omega}_r = [\omega_\rho \;\; \omega_\psi \;\; \omega_z]^T$ model variations in the setpoints between time samples.

$\boldsymbol{\omega}_m$ model uncertainties on the internal states.

$\boldsymbol{\nu}_m$ model uncertainties on the outputs of model $\mathbf{G}(z)$.

$\boldsymbol{\omega}_p = [\omega_x \;\; \nu_y]^T$ model errors due to approximating the non-linear motion equations by their linear counterparts.

7.3.2 State Equation

The state equation describes the evolution of all states in the system. As shown in (7.22), the evolution of four distinct sets of states must be defined. These sets are the setpoints $\mathbf{X}_r$, the internal states and outputs of model $\mathbf{G}(z)$,respectively $\mathbf{X}_m$ and $\mathbf{X}_y$, and the position of the object in space $\mathbf{X}_p$. The evolution can be obtained by the following steps [Prevost et al., 2007]:

Setpoint Evolution

The object setpoints are completely unknown and are modeled as independent random walks

$$\mathbf{X}_r(k+1) = \mathbf{X}_r(k) + \boldsymbol{\omega}_r(k) \tag{7.25}$$

Setting $\boldsymbol{\omega}_r = 0$ assumes that the object setpoints remain constant throughout time.

Object Dynamics

The object dynamics are given as the output of the modeled dynamics given by the setpoints

$$\mathbf{X}_y(k) = Z\left\{\mathbf{G}(s)\right\}\mathbf{X}_r(k) \tag{7.26}$$

where $Z\{\cdot\}$ denotes the z-transform of the continuous-time transfer function and where

$$\mathbf{G}(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) & G_{13}(s) \\ G_{21}(s) & G_{22}(s) & G_{23}(s) \\ G_{31}(s) & G_{32}(s) & G_{33}(s) \end{bmatrix} \tag{7.27}$$

Secondary order low pass filters represent the dynamics in the direct branches:

$$G_{ii}(s) = \frac{1}{\left(1 + T_{ii}s\right)^2} \tag{7.28}$$

where $i \in 1, 2, 3$. The dynamics in the indirect (coupling) branches are modeled as secondary order band pass filters:

$$G_{ij}(s) = \frac{s}{(1 + T_{ij}s)^2} \tag{7.29}$$

where $i, j \in 1, 2, 3$ and $i \neq j$.

The use of second order low pass filter is assumed to be sufficient to model the dynamics in the system. Although higher-order filters can encompass more dynamics, they will most likely not increase the performance in this application since there are to many uncertainties in the system, as the only measurement is the object position.

The continuous-time transfer function matrix $\mathbf{G}(s)$ needs to be discretized to a state-space representation $(\mathbf{A}_m, \mathbf{B}_m, \mathbf{C}_m)$. Using this discrete state-space representation with the setpoints $\mathbf{X}_r$ as an input vector the evolution of the internal states is computed:

$$\mathbf{X}_m(k+1) = \mathbf{A}_m\mathbf{X}_m(k) + \mathbf{B}_m\mathbf{X}_r(k) + \boldsymbol{\omega}_m(k) \tag{7.30}$$

$$\mathbf{X}_y(k) = \mathbf{C}_m\mathbf{X}_m(k) + \boldsymbol{\nu}_m(k) \tag{7.31}$$

where the state vector $\mathbf{X}_m$ is composed of the internal states in $\mathbf{G}(z)$, and where $\mathbf{X}_y$ is the output vector.

Motion equations

The object position in the xy-plane is governed by the motion equations

$$x(k+1) = x(k) + T_s y_\rho(k) \cos y_\psi(k) \tag{7.32}$$

$$y(k+1) = y(k) + T_s y_\rho(k) \sin y_\psi(k) \tag{7.33}$$

where T_s is the sampling period, and x and y are the positions of the object along the x-axis and y-axis respectively. However, the object speed and heading in the xy-plane y_ρ and y_ψ respectively are not known, only estimated. Since equations (7.32) and (7.33) are non-linear, described by

$$\mathbf{X}_p(k+1) = \mathbf{f}_p\left[\mathbf{X}_p(k), \mathbf{X}_y(k)\right] \tag{7.34}$$

a linearization of these equations are done at each iteration using a first-order Taylor series expansion about the operating point $(\mathbf{X}_y^*, \mathbf{X}_p^*)$:

$$\begin{aligned}\mathbf{X}_p(k+1) \approx \mathbf{f}_p\left[\mathbf{X}_p(k)^*, \mathbf{X}_y(k)^*\right] &+ \frac{\partial \mathbf{f}_p\left[\mathbf{X}_p(k)^*, \mathbf{X}_y(k)^*\right]}{\partial \mathbf{X}_p(k)} \delta \mathbf{X}_p(k) \\ &+ \frac{\partial \mathbf{f}_p\left[\mathbf{X}_p(k)^*, \mathbf{X}_y(k)^*\right]}{\partial \mathbf{X}_y(k)} \delta \mathbf{X}_y(k)\end{aligned} \tag{7.35}$$

where $\delta\mathbf{X}_y(k)$ and $\delta\mathbf{X}_p(k)$ are the perturbation about the operating point. As a result, the following equation is obtained:

$$\mathbf{X}_p(k+1) = \mathbf{X}_p(k) + \mathbf{B}_p \mathbf{X}_y(k) + \boldsymbol{\omega}_p(k) \tag{7.36}$$

where

$$\mathbf{B}_p = \begin{bmatrix} T_s \cos y_\psi^*(k) & -T_s y_\rho^*(k) \sin y_\psi^*(k) & 0 \\ T_s \sin y_\psi^*(k) & T_s y_\rho^*(k) \cos y_\psi^*(k) & 0 \end{bmatrix} \tag{7.37}$$

and where the operating points y_ψ^* and y_ρ^* are used to linearize the motion equations correspond to the states predicted/estimated by the extended Kalman filter at the previous sample time.

Note that the evolution of the object position along the z-axis has already been described by the evolution of the altitude output z.

Complete State Equation

The evolution of the four distinct sets of states are now defined, and by combining (7.25), (7.30), (7.31) and (7.36), yields the linear state equation governing the evolution of all states in the system

$$\mathbf{X}(k+1) = \mathbf{A}\mathbf{X}(k) + \boldsymbol{\omega}(k) \tag{7.38}$$

where $\mathbf{X}$ is the complete state vector given in (7.22), and where

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & 0 & 0 & 0 \\ \mathbf{B}_m & \mathbf{A}_m & 0 & 0 \\ \mathbf{C}_m\mathbf{B}_m & \mathbf{C}_m\mathbf{A}_m & 0 & 0 \\ 0 & 0 & \mathbf{B}_p & \mathbf{I} \end{bmatrix} \tag{7.39}$$

and $\boldsymbol{\omega}(k)$ with covariance matrix $\mathbf{Q}$ models the uncertainty on all system states. Note that the linearization of (7.34) made it possible to obtain a linear state equation to be used in the extended Kalman filter. However, if the nonlinear equation (7.34) is combined with (7.25), (7.30) and (7.31), a non-linear state equation describing the object motion is obtained, written on compact form:

$$\mathbf{X}(k+1) = \mathbf{F}[\mathbf{X}(k)] + \boldsymbol{\omega}(k) \tag{7.40}$$

7.3.3 The Measurement Equation

The measurement equation expresses the position of the object measured by the UAV in terms of the state vector defined in (7.22). It is defined

$$\begin{bmatrix} \mathbf{X}_p^m(k) \\ z^m(k) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \mathbf{I} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{X}_r(k) \\ \mathbf{X}_m(k) \\ y_\rho(k) \\ y_\psi(k) \\ z(k) \\ \mathbf{X}_p(k) \end{bmatrix} + \boldsymbol{\nu}(k) \tag{7.41}$$

where the vector $\begin{bmatrix} \mathbf{X}_p^m & z^m \end{bmatrix}^T$ are the position measurements of the object. To model the uncertainty caused by sensor noise the random vector $\boldsymbol{\nu}$ with covariance matrix $\mathbf{R}$ is added to the measurement equation. Equation (7.41) can be written in a more compact form

$$\mathbf{Y}(k) = \mathbf{C}\mathbf{X}(k) + \boldsymbol{\nu}(k) \tag{7.42}$$

7.4 Simulation

To demonstrate the performance of the extended Kalman filter presented in Section 7.2, a simulated test case, given in the next sections, is made in MATLAB.

7.4.1 Simulation Setup

The true dynamics of the moving object detected by the UAV are given:

$$\mathbf{X}_y^*(k) = Z\left\{\mathbf{G}^*(s)\right\}\mathbf{X}_r^*(k) \tag{7.43}$$

where

$$\mathbf{G}^*(s) = \begin{bmatrix} G^*_{11}(s) & 0 & G^*_{13}(s) \\ 0 & G^*_{22}(s) & 0 \\ G^*_{31}(s) & 0 & G^*_{33}(s) \end{bmatrix} \tag{7.44}$$

where

$$G^*_{ii}(s) = \frac{1}{(1+12s)^2} \tag{7.45}$$

$$G^*_{ij}(s) = \frac{2s}{(1+18s)^2} \tag{7.46}$$

where $i \in 1,2,3$, $j \in 1,3$ and $i \neq j$. This gives smooth object dynamics.

Suppose that the EKF incorrectly uses the following dynamics in the motion calculation of the observed object:

$$\mathbf{X}^*_y(k) = Z\left\{\mathbf{G}^*(s)\right\}\mathbf{X}^*_r(k) \tag{7.47}$$

where

$$\mathbf{G}(s) = \begin{bmatrix} G_{11}(s) & 0 & G_{13}(s) \\ 0 & G_{22}(s) & 0 \\ G_{31}(s) & 0 & G_{33}(s) \end{bmatrix} \tag{7.48}$$

where

$$G_{ii}(s) = \frac{1}{(1+10s)^2} \tag{7.49}$$

$$G_{ij}(s) = \frac{4s}{(1+24s)^2} \tag{7.50}$$

where $i \in 1,2,3$, $j \in 1,3$ and $i \neq j$. The sampling period for both process and model is $T_s = 1$ seconds. Finally, suppose that the sensor measuring the position of the object is noisy.

7.4.2 Implementation

The extended Kalman filter is used to estimate the true object setpoints, internal states of model $\mathbf{G}(s)$, outputs of model $\mathbf{G}(s)$ and object position. The filter is implemented assuming that setpoints are varying like random walks and the model of the object dynamics are uncertain. The measurements uncertainty are given by the *randn()* function in MATLAB, which gives pseudorandom values drawn from the standard normal distribution.

The discrete-time quantities $\mathbf{\Phi}(k)$, $\mathbf{f}[\hat{\mathbf{X}}(k)]$ and $\mathbf{h}\left(\hat{\mathbf{X}}^-(k)\right)$ in the extended

Kalman filter are given:

$$\mathbf{\Phi}(k) = \mathbf{A} \tag{7.51}$$

$$\mathbf{f}[\hat{\mathbf{X}}(k)] = \mathbf{A}\hat{\mathbf{X}}(k) \tag{7.52}$$

$$\mathbf{h}\left(\hat{\mathbf{X}}^{-}(k)\right) = \mathbf{C}\hat{\mathbf{X}}^{-}(k) \tag{7.53}$$

where **A** is given in (7.39) and **C** is defined in Equation (7.42).

The true initial states, the initial aposteriori state estimate and the covariance matrices are given in Listing 7.1. These are the tuning factors in the extended Kalman filter. The measurement noise is tuned high to simulate the difficulties obtaining the object position from a camera module.

Listing 7.1: Tuning parameters in EKF.

```
% True initial state
x(:,1) = [1; pi/180*100; 30; zeros(15,1)];

% Initial aposteriori state estimate
x_aposteriori(:,1) = [0.5; pi/180*50; 20; zeros(15,1)];

% State-error Jacobian
W = diag([1.6 0.4 2 ones(1,15)]);

% Measurement-error Jacobian
V = diag([1.6 0.8 1]);

% Process noise covariance
Q = W*diag([0.0025 0.0025 0.25, 0.005*ones(1,15)])*W';

% Measurement noise covariance
R = V*diag([100 100 100])*V';

% Initial aposteriori error covariance estimate
P_aposteriori(:,:,1) = diag([1 0.1 1, 0.1*ones(1,15)]);
```

7.4.3 Simulation Results

Test results of the simulation are illustrated in Figure 7.3, 7.4 and 7.5, estimating object setpoints, outputs of model $\mathbf{G}(s)$ and object positions, respectively. The results show that the EKF succeeds in estimating the object positions in spite of measurement and model uncertainty. However, the estimation of the setpoints and model outputs are not adequate, but can be tuned to increase the performance. The estimation error of the model outputs is lower than for the setpoints following the trends more closely. The best setpoint estimate is in altitude.

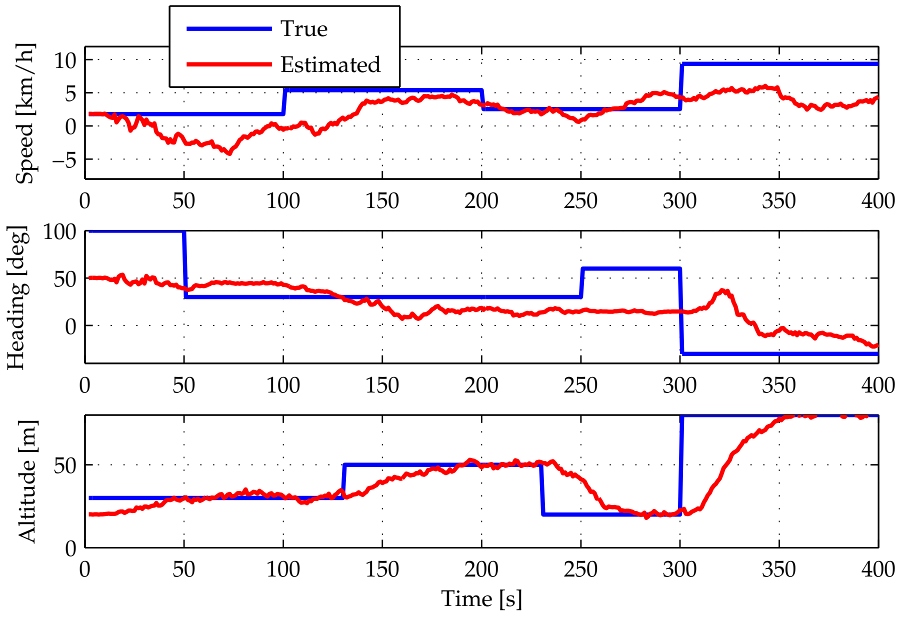


Figure 7.3: True and estimated setpoints.

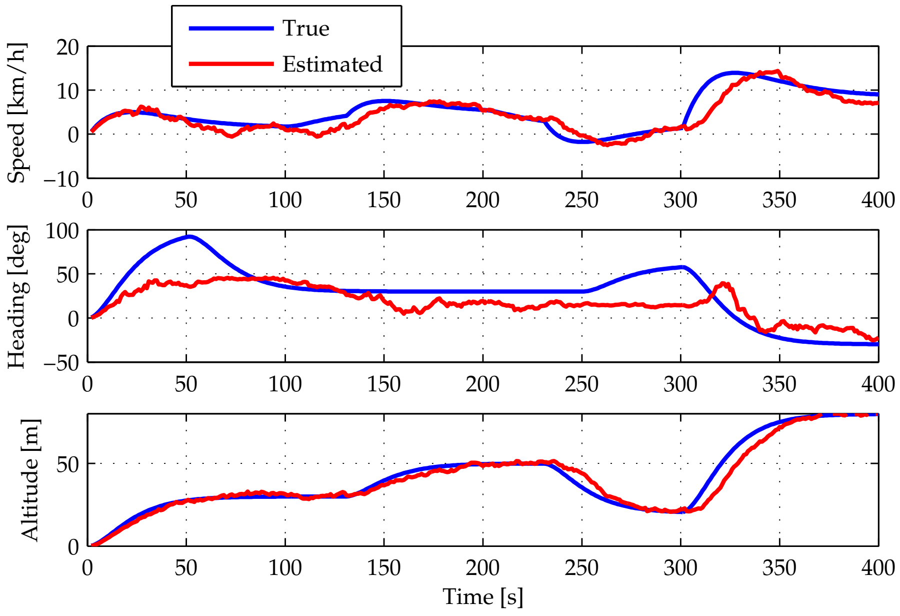


Figure 7.4: True and estimated model outputs.

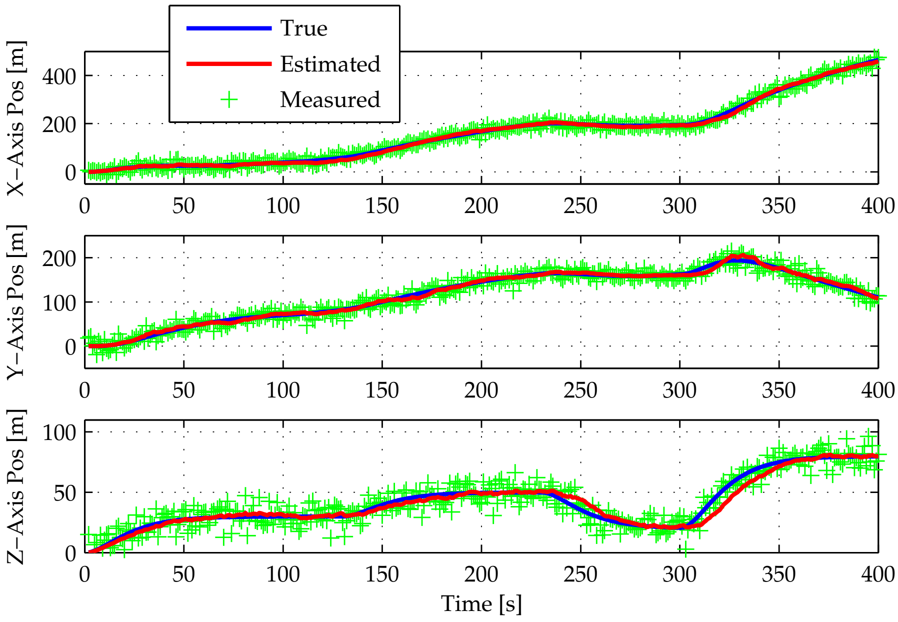


Figure 7.5: True, estimated and measured positions.

7.5 Summary

In the tuning process, the estimation of the position is generally producing good results. However, the estimation of the model outputs and especially the setpoints are not as robust as the position estimates. The tuning of design matrices and uncertainties are important for the final simulation result. However, the purpose of the simulation was to confirm that the EKF could be used to estimate the motion of a target object using the system represented by (7.38) and (7.42). As this was accomplished, the method can be used to support the autonomous SAR multicopter system.

A module that can support the autonomous guidance controller with information about the real world coordinates of a target can be of great advantage. Especially in a process of regaining tracking of a missing target, but also further in the search operation. Information about the motion of a target object can be used to optimize the search pattern, predict future position and better be able to keep hold of the object position while the search is resumed looking for other objects.

The extended Kalman filter should be implemented as a separate task based on the *navigation package* in DUNE, which already can handle Kalman filters, provided that the state matrix is updated accordingly. Although the Kalman filter class in DUNE is not valid for non-linear systems, the limited predict method in DUNE can be used in this case since the measurement equation (7.42) expresses the position of the object linearly.

Since there are limited computational power on the single-board computer and it turns out that the extended Kalman filter is to computational demanding, there are done other approaches to estimate motion of a target object. The extended Kalman filter itself should not be of any concern alone, but since the frame rate of the computer vision algorithms often wants to be maximized, it will limit all other systems. Another algorithm that should require less computational power is an approach by Zhiyuan et al. [2010] where the estimation is based on an adaptive law and low-pass filtering to estimate the target's velocity and heading angle.

Chapter 8

Experiments

In the development process, the motivation has been to make a system that autonomously can be used in a real search and rescue operation. Although that the fully autonomous system is not realized in this thesis, the focus has all along been to develop subsystems that will benefit this system. The subsystems are therefore designed to be modular and flexible to easily allow further development.

The modular design also made testing possible since the missing module from the fully autonomous system easily could be replaced by simple logic to prove the concept. The module that is missing from completing the fully autonomous system is the integration of a computer vision algorithm for automatically object detection. A simple panel in Neptus, where a manual detection instead can be given, solves this. The parts that have been extensively tested and examined in simulators and real flights are summarized by the following points:

- The features in APM:Copter.
- The extensions to APM:Copter.
- The low-level control interface.
- The interaction between APM:Copter, DUNE and Neptus.

These parts constitute the framework needed in order to test the principle of the fully autonomous search and rescue multicopter system.

Regardless of the missing computer vision algorithm, the experiments presented verify how the search operation can be executed using waypoint guidance and an observer spotting for objects on the ground. This system can definitely be a resource in search operations providing air support.

8.1 Test Strategy

In the development process, continuously and rapidly testing has been of great value. Especially, the possibility of testing parts of the system at an early stage speed up the process. Usually it is not practical or doable to perform field tests for each parts alone. For this reason, testing of the system has been conducted in three different stages:

Software In Loop
In the first stage of testing, *software in loop* or SIL simulation is conducted. The purpose behind software in loop simulation is to test the software, verifying that algorithms, communication and coding are working properly. Most of the testing in the development process is done in a software in loop simulator as it requires little effort and is quickly performed.

Target In Loop
After software in loop testing is conducted and the system appears to be well functional in the simulator, the next stage is the *target in loop* (TIL) test. In a target in loop simulation the software modules are installed at the designated hardware if possible. Normally a *hardware in loop* (HIL) simulation is conducted as the final stage before a field test is carried out. However, TIL has become the substitute due to problems and limitations running the APM code in HIL.

Field test
The final and ultimate test of the system is a complete field test. In real flight, all possible factors play into account, whether it is problems with the multicopter, software or as simple as bad weather.

The full description of the setup and how software in loop and target in loop simulations can be conducted is given in the following subsections.

8.1.1 Software in Loop

The software in loop simulator for the SAR multicopter system is running a simulated APM in software. In addition, several tasks added in DUNE are responsible for reading RC signals, running the simulated multicopter model and transporting messages between modules. It is in this phase that coding failures start to become evident. The modules in the software in loop simulator are illustrated in Figure 8.1.

8.1.2 Target in Loop

In the final test stage the software modules should preferably be fully installed into the designated hardware and only interact with the environment through the proper input and output of the system. The environment needs to be simulated

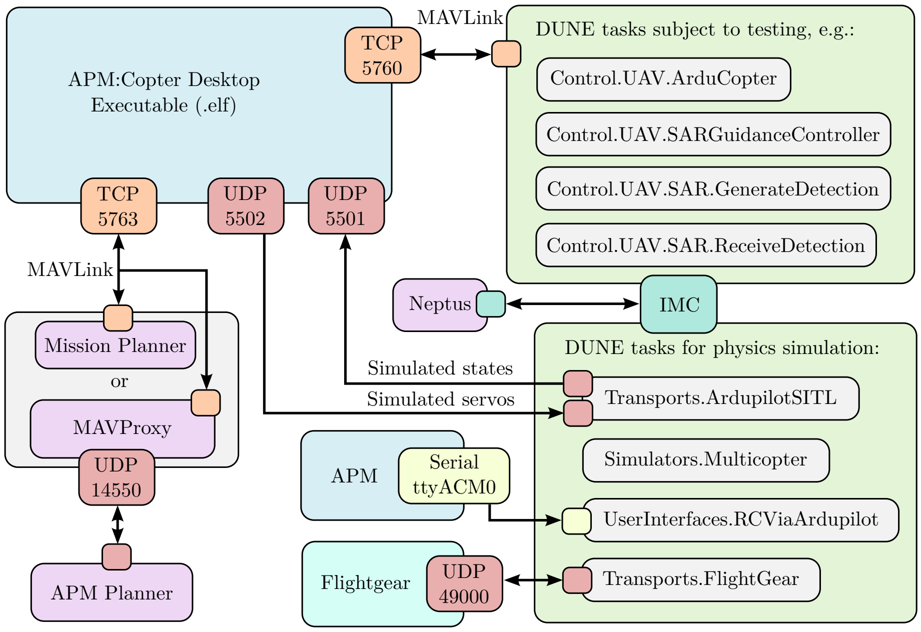


Figure 8.1: The interaction and setup of modules in SIL testing. Based on `https://code.google.com/p/ardupilot-mega/wiki/SITL`

with input and output simulation to fool the system into believing it is running a multicopter in the real world. In the ideal hardware in loop simulation, the difference to a real field test should be at a minimum. However, since it does not exist a good solution to fool the APM:Copter by input and output simulations, a HIL test is for the time being not achievable.

In the target in loop test, compared to the software in loop, DUNE is not running on the local computer, but at the PandaBoard itself. Since the APM:Copter needs to be simulated in software, it must be executed as in the software in loop test, at a desktop. This is the reason why it is a target in loop test and not a complete hardware in loop test. However, it is the test closest to a real field test and can expose problems with software interactions and particularly any problems with running DUNE at the PandaBoard. The setup of the TIL simulator is similar to the SIL test illustrated in Figure 8.1 but where the DUNE tasks subject to testing is run on the PandaBoard. Target in loop simulation can also be used as the final preparation before the field test, going through all points in the checklist (see Appendix A).

There have been conducted a great deal of research on how the APM could be included in the hardware in loop simulator setup [Steen, 2013]. At the moment, a well-tuned target in loop test is the best substitute. However, there will always occur more problems in the field that could have been discovered in a proper HIL test. Especially the failures regarded to the APM autopilot hardware are hard to discover without flying. With that said, since multicopters are relatively easy to deploy and only require a small designated test area, the test solutions with target in loop simulation might be good enough making the priority of solving the HIL problems low. The problems that can not be discovered in a target in loop simulation are mostly related to hardware in a start-up phase, and might in fact be easier to fix directly in the field.

8.2 Preliminary Field Tests

Already in February 2014 a test flight was performed with the objective to get out in the field and test some of the existing features. The importance of having the first test flight was to clarify what a successful mission required, regarding to equipment and personnel in addition to see if some unexpected factors needed to be taken into consideration. The features that were tested can be summarized in the following points:

- Basic flight maneuvers in APM:Copter.
- The USB communication link between DUNE and APM.
- The communication between DUNE and Neptus over wifi.
- That executed flight commands in Neptus was transmitted and executed on APM.

The test flight was defined a success based on good feedback on the test objectives. However, it should be mentioned that the *GoTo* command in Neptus had a set point at 200 meters, which made the command impossible to finish. The conclusion of a successfully executed command was therefore based on that the hexacopter started to gain altitude, not that it successfully reached the target, but that the command was transmitted as planned. The outcome of the first test flight regarding to the objective to clarify the mission needs turned out with few unexpected factors, with the exception on how altitude in a GoTo maneuver should be set.

The objective of the second test flight was to test the first outline of the low-level control interface. The test was conducted at Agdenes in moderate wind conditions. The interface setting roll, pitch and yaw angles was successfully conducted, although the results could not be proved due to absence of logging. However, the primary objective was to conclude whether it was possible to send attitude angles with a MAVLink message, or not. The results were therefore most easily verified with a change in yaw. The visual results were concluded promising.

Now, with promising results for the low-level control interface, the motivation for redefining the interface more in line with the structure in DUNE was high. Especially, since a well functional interface would not only benefit the SAR mission but also other and future applications. The structure of the interface was therefore reviewed and restructured using the different control layers in DUNE (see Figure 3.2). The autonomous task was also designed in this manner.

Finally, the test of the complete system was conducted after a series of navigation problems discussed in Section 4.5. The detailed setup and results are presented in the following sections.

8.3 Experimental Setup in Field

The test site for the final experiments of the complete system was at the airfield at Agdenes. The presented flights took place in moderate wind conditions over the airstrip (see Figure 8.2).

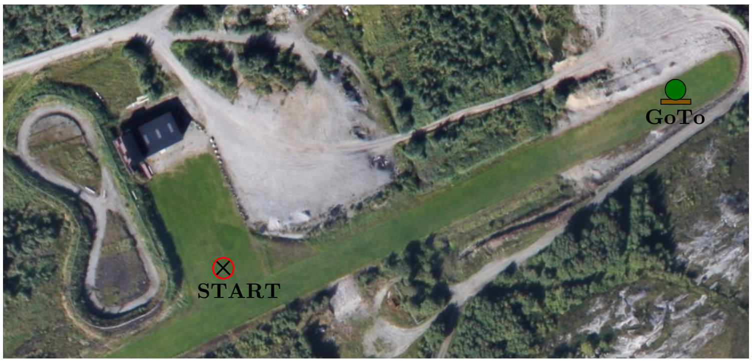


Figure 8.2: The flight map in Neptus showing the airstrip at Agdenes with the designated *GoTo* maneuver. The map data is retrieved from `http://norgeibilder.no`

The mission's general plan was to search the airstrip by moving towards a waypoint given by a *GoTo* maneuver in Neptus. The flight operator was in charge of observing objects by watching the video stream at the laptop (see Figure 8.3). To simulate camera detection, the operator clicked the manual detection button in Neptus starting the *SARGuidanceController* in DUNE. The first flight test was performed with a guidance objective to start loiter when the detection occurred. To be able to show it more illustratively on a map, the objective of the second test was designed to move the multicopter left for 15 seconds and then back moving right for 15 seconds, repeatedly. After an visual inspection of the object was conducted by the operator, a confirmed or reject button was pressed and the multicopter resumed the mission plan proceeding towards the designated waypoint.

Figure 8.3: The operator monitors the multicopter's status, mission and watches the video stream for objects, responsible for sending a manual detection.

8.4 Experimental Results in Field

The motivation and high-level objective for the final experiment was to confirm that all modules worked together forming the SAR multicopter system. This means that the following points should be achieved:

- Execute a mission plan based on waypoint (GoTo) maneuvers from Neptus.
- View the video feed from the multicopter on the laptop and monitor status.
- Send manual object detection from Neptus to DUNE over wifi.
- Start an autonomous guidance control task in DUNE triggered by the received object detection.
- Use the low-level control interface to perform an autonomous operation.
- Abort the autonomous operation by sending a confirmed object message from Neptus to DUNE.
- Autonomously resume the mission plan after the object is inspected.

The mission start and final waypoint is given in Figure 8.2. To execute the plan, the pilot manually flew the multicopter into the start position with an altitude at approximately 9 meters before switching the flight mode to loiter. The flight path of the executed mission is shown in Figure 8.4.

Given the direction of the multicopter after the autonomous operation where the mission plan is resumed, it can be seen that a pure pursuit algorithm is used for the waypoint guidance. If the LOS guidance algorithm had been used with the start point as reference point, the multicopter should have merged more towards the original course towards the waypoint.

Considering the disturbance caused by the wind, the performance of both waypoint guidance and autonomous operations is satisfying. The multicopter keeps straight at the target both before and after the autonomous operation. The autonomous operation, although it differs a few meters from a straight course, is

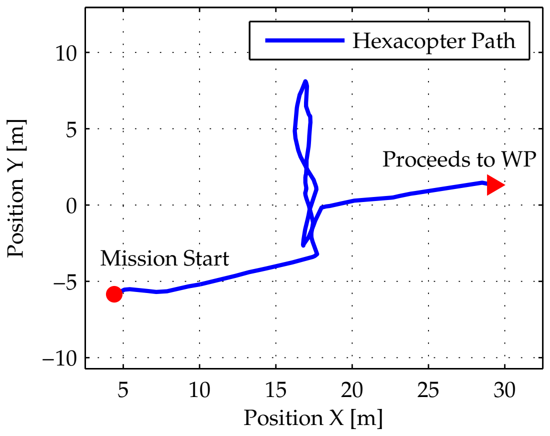


Figure 8.4: The flight path of the SAR operation. The autonomous guidance operation is easily seen, as the multicopter starts moving left, then to the right, before moving left again for some meters before the mission plan is resumed.

well executed. The performance can better be seen in Figure 8.5 where it is shown from the velocity perspective.

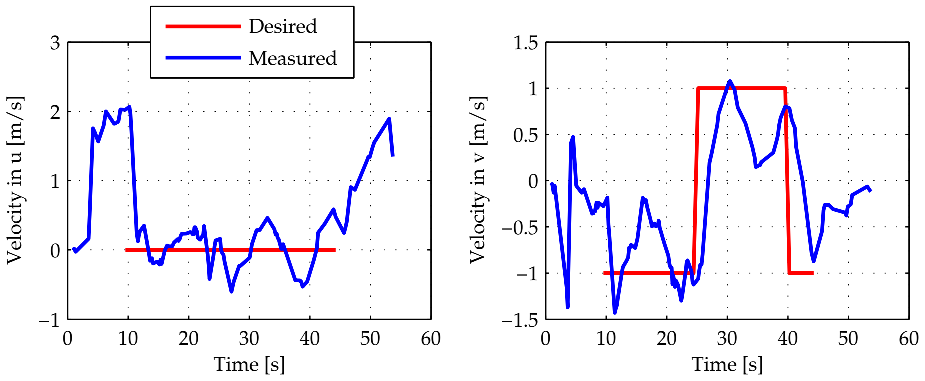


Figure 8.5: The measured and desired velocity in u and v defined as forward and sideways velocities in body, respectively. The desired sideways velocity v clearly shows the repeatedly movement to the left and right.

The maximum velocity during waypoint guidance is 2 m/s. The yaw angle is controlled to point the nose directly towards the waypoint. This means that the desired velocity during waypoint guidance should be the maximized forward in the body frame. In other words, ideally should $u = 2\,\mathrm{m/s}$ and $v = 0\,\mathrm{m/s}$. This is how the velocity is handled in the theory and correspond almost to the actual movement. However, as seen in Figure 8.5, the measured velocity differ from this theory for the first four seconds. The reason for this is that the multicopter was not facing the waypoint, which means that a velocity in v was towards the target. The change in yaw can be seen in Figure 8.6 making u the forward velocity.

The error between desired and measured velocity differ more at some point, typically before the multicopter has reach a steady state. The term steady state is here referring to the state where the multicopter has stabilized after attitude setpoints are changed, which is given when the multicopter changes direction. This flight test was not specially made with the purpose of testing the performance of the waypoint guidance, as the copter almost never achieves steady state. In addition, disturbances in the wind will influence the performance.

To give an impression of how the multicopter actually behaved during the flight, the attitude is shown together with ground speed in Figure 8.6. The fluctuations in both roll and pitch states that there have been some disturbances due to wind.

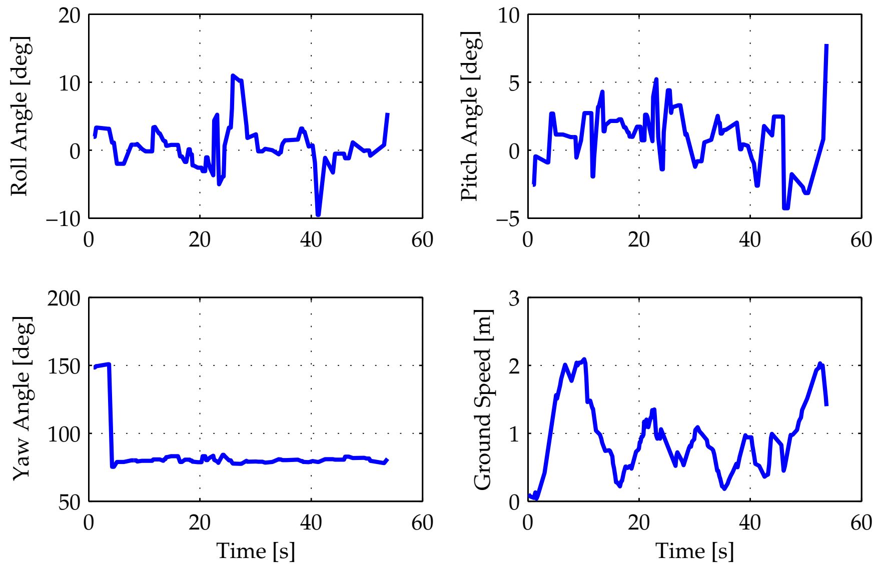


Figure 8.6: Roll, pitch, yaw and ground speed.

The altitude was set to altitude hold for the entire mission and is shown in Figure 8.7 together with measured current and voltage of the APM and the reported software CPU (central processing unit) usage of the PandaBoard. A small drop in altitude happens when the autonomous operation starts due to the change in attitude causing the hexacopter to tilt and correspondingly losing some lift. This is quickly compensated, but it can be argued for that the tilt compensation is too low based on the altitude dip. Or it can be interpreted quite opposite, that the tilt compensation during waypoint guidance is to aggressive making the multicopter to elevate. However, the barometer sensitivity at the APM can give larger fluctuations than one meter. A separate altitude sensor is needed to measure the accuracy of the barometer, for instance a sonar or a laser rangefinder.

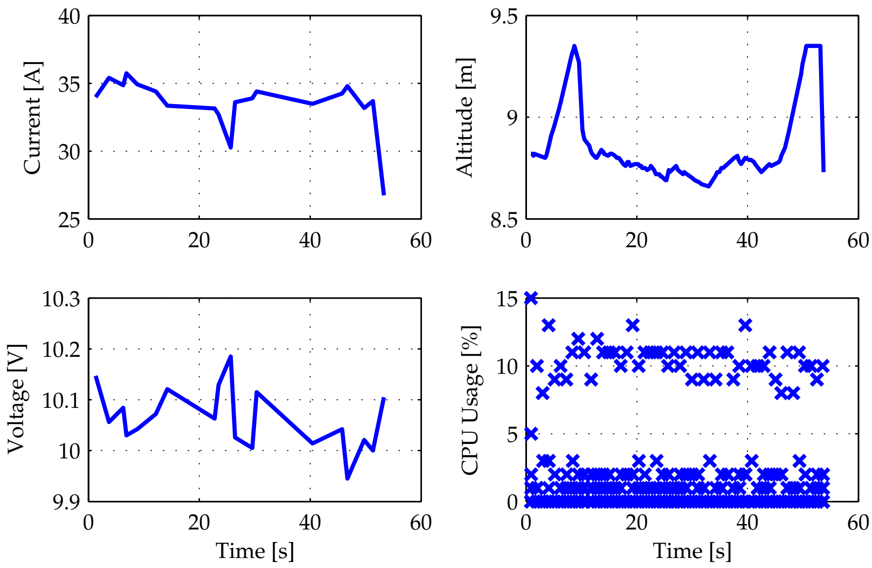


Figure 8.7: The altitude together with the monitored parameters current, voltage and CPU usage.

Analyses of current and voltage of the APM make logical sense in Figure 8.7. It is possible to see a falling trend of the voltage. Furthermore, if current is compared with voltage it can be seen that if the current level drops, for instance after 22 seconds, a correspondingly raise can be seen in the voltage level. Although the voltage level is continuously falling the same raise can be seen at the end of the flight, when the current level drops. The CPU usage of the PandaBoard shows that DUNE has more than enough power during the flight.

Chapter 9

Conclusion and Closing Discussion

Autonomous control, whose goal is to realize the autonomous flight of an unmanned aerial vehicle, is a great challenge. The fully autonomous multicopter is a vision that can contribute, not only to the search and rescue mission, but also to other applications. However, in the design that can perform self-governing flight without human intervention there are challenges due to both hardware and software. The goal of this thesis was to design and implement methods for the autonomous multicopter as a contribution to the search and rescue operation. The final system was verified by simulations and experiments to illustrate how the system can be utilized in practice.

First, the hardware and software modules that form the search and rescue multicopter was presented. The PandaBoard, as the single-board computer, was chosen to operate the runtime environment on-board the multicopter together with the ArduPilot Mega autopilot. Other hardware components were introduced for integration of a camera in addition to the necessary radio controller for manual control. On the software side, DUNE was briefly introduced as the on-board runtime environment due to the modular and versatile design. Furthermore, the protocols for communication between DUNE, APM and ground control stations were described. The two ground control stations in the APM autopilot suite were characterized as good configuration tools but hard to customize and with lack of support for more advanced operations. Neptus was therefore presented as the last piece in the LSTS Neptus-IMC-DUNE software toolchain. To summarize the search and rescue multicopter system a detailed figure was made illustrating communication links and the system in its entirety.

Followed by the overall system description of the SAR multicopter an introduction to APM:Copter and DUNE were given. To begin with, some of the flight modes was described in addition to the available features in APM:Copter. It was shown that mission planning was limited to pure pursuit waypoint guidance and predefined “do” commands. Next, a structural presentation of the functionality in DUNE was given to emphasize the advantages of a modular and flexible design.

For the purpose of autonomous control, an introduction to the design of three subsystems to control the movement of the multicoper was provided. The study of the guidance, navigation and control gave an overview of each subsystem from the APM:Copter’s perspective. The integration of DUNE as a guidance framework was discussed to highlight possibilities for further optimization of the guidance system. Furthermore, challenges due to navigation was discussed and presented.

With the existing features in APM:Copter presented, a low-level control interface was found necessary to extend the possibilities of autonomous behavior in flight. The result was an extension to APM:Copter using the MAVLink protocol for allowing both attitude and velocity control of the multicoper. The purpose of this interface was to enable low-level control from the on-board single-board computer at the multicopter. The interface was implemented with a modular approach allowing different controllers in roll-pitch, yaw and throttle to be combined and triggered by a single target mask. This benefits particularly the development phase since individual controllers can be tested separately. In addition, the general application with the need off low-level control can utilize this interface due to its generality.

The high-level goal of this thesis was to design and implement promising methods for the autonomous multicopter. It was emphasized that a camera module was one of the vital parts to achieve fully autonomous guidance and control. Since the integration of the computer vision module was not covered in this thesis, simple logic replaced the automatically object detection algorithm allowing manual detection possible from Neptus. However, with the objective of preparing for the fully autonomous multicopter, camera integration was discussed with the purpose of describing how a camera module could be utilized. The outcome of this was the transformation from the image frame to the global frame using the camera module as a sensor retrieving the real world coordinates of an object. The advantage of estimating the real world coordinates was concluded to make the process of regaining tracking of a lost object in the image frame easier, as the last known position would provide a good starting point.

To describe the motion of a target object, the object dynamics was represented by a state equation governing the evolution of all states in the system. The motion equations were linearized by Taylor approximation giving a linear state equation. The extended Kalman filter was thoroughly presented and introduced as a measure-and-estimate approach for the motion estimation. The results of the simulation demonstrated that the object position could be estimated in spite of uncertainties and noisy measurements.

Finally, the test results of the experiments concluded the potential for the fully autonomous multicopter system. The general mission was conducted with the high-level objective to confirm that all modules worked together. The experiments verified that the autonomous search and rescue multicoper could contribute in a search operation using an observer to spot for objects. Furthermore, the test results showed good performance of both waypoint guidance and the autonomous behavior in flight.

The importance of a complete field test was one of the cornerstones in this thesis, proving that the concept of the autonomous multicopter for use in search and rescue or other applications was viable. Furthermore, with the design of the autonomous multicopter with possibility for low-level operations, future work can be concentrated more towards the actual autonomous mission since the different parts now are united.

Appendix A

Hexacopter Checklist

Home

- Start the laptop with the ground control station (Neptus/Mission Planner).
- Pull all repositories (DUNE, Neptus, IMC, IMCjava and APM:Copter):
  - Same procedure for all repositories. Example of DUNE:
    - `cd  /dune/dune`
    - `git fetch all`
    - `git checkout YOUR-BRANCH`
- Check to build DUNE and APM:Copter.
  - Update APM tools if necessary with:
    - `ardupilot/Tools/scripts/install-prereqs-ubuntu.sh`
  - Install MAVProxy if necessary with:
    - `sudo apt-get install python-pip`
    - `sudo pip install MAVProxy`
- Create missions in Neptus.
- Load maps.

Pre-flight

ArduPilot Mega

- Copy old parameters away from target:
  - With MAVProxy:
    - `sudo mavproxy.py -master=/dev/ttyACM0, 115200`
    - `param save ArduCopter.TodaysDate.parm`
  - With Mission Planner or APM Planner:
    - Full-parameter list → save tofile: hexa-00x.TodaysDate.parm
- Check frame configuration for hexacopter in *ArduCopter/APM_config.h*:

  - It should contain the line: `#define FRAME_CONFIG HEXA_FRAME`
- Compile new version APM:Copter:
  - `make configure`
  - `make -j4`
  - `make upload`
- Open APM Planner or Mission Planner, upload old parameters back (or check for equality).
- Check frame type to be 1 for X configuration.
- Check flight-modes:
  - Example:
    - Mode 1-2: AltHold/Loiter
    - Mode 3-4: Dune
    - Mode 5-6: Stabilize/Dictator
- Check the parameter `ARMING_CHECK` to be 1.
- (Calibrate accelerometer and magnetometer if needed.)
- Calibrate Radio.

Target/PandaBoard

- Router: Connect, and verify that the wifi symbol is green.
- Power on Pandaboard. Verify two lights by SD-card.
- Neptus: Verify connected pandaboard with valid IP-address.
  - Troubleshoot:
    - Check IP of Neptus.
    - Connect serial port to target (or use SSH) to check IP.
    - Check dune service with:
      - `status dune`
    - Check logs in:
      - /home/share/dune/dune_upstart.log
      - /var/log/upstart/dune.conf
    - Start manually for test:
      - `/home/share/dune/bin/dune-launcher`
- Double-check `Auto Custom Maneuver` in `[Control.UAV.ArduCopter/Hardware]`
- Upload your DUNE:
  - `cd armbuild`
  - `cmake -DCROSS=arm-linux-gnueabihf- ../dune`
  - `make package`
  - `rsync -avz dune-2.*.tar.bz2 root@192.168.0.xxx:`
- The dune-service should update dune.
- Re-check Neptus status. If no IP, repeat steps above summarized by:
  - `stop/start/status dune`
- If you are using *Configure System* in Neptus, update configuration files:
  - `cd build`
  - `./dune -c ntnu-hexa-00x -p Hardware -X ../neptus/conf/params`
- Connect APM. Check for status, check pitch/roll, check state of APM.

In flight

- Monitor voltage. Minimum voltage is 9.9 V for 3 cell LiPo.
- If compass problems occur:
  - Perform factory reset and initialization of EEPROM values:
    - Run command line (CLI) in Mission Planner and type:
      - `erase`
      - `reset`
  - Recalibrate the APM (see above in Pre-flight).

After flight

- Fetch logs with something like:
  - `scp root@192.168.0.xxx:/home/share/dune/log .`
  - `sudo chown -R uavlab:uavlab log/`

Bibliography

Andersen, H. (2014). *Path Planning for Search and Rescue mission using multicopters.* Master thesis, Department of Engineering Cybernetics, NTNU.

APM (2014a). The ArduPilot Mega Development Site. [Online] Available: `http://dev.ardupilot.com/`.

APM (2014b). The ArduPilot Mega Multiplatform Autopilot. [Online] Available: `http://ardupilot.com/`.

Breivik, M. & Fossen, T. I. (2008). Guidance laws for planar motion control. In *47th IEEE Conference on Decision and Control - Cancun, Mexico*, (pp. 570–577).

Brown, R. G. & Hwang, P. Y. C. (2012). *Introduction to random signals and applied Kalman filtering: with MATLAB exercises* (4th ed.). Wiley.

Fossen, T. I. (2011). *Handbook of marine craft hydrodynamics and motion control* (1st ed.). Wiley.

Goodrich, M. A., Morse, B. S., Gerhardt, D., Cooper, J. L., Quigley, M., Adams, J. A., & Humphrey, C. (2008). Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, *25*(1-2), 89–110.

Hai, C., Xin-Min, W., & Yan, L. (2009). A Survey of Autonomous Control for UAV. In *International Conference on Artificial Intelligence and Computational Intelligence - Shanghai, China*, volume 2, (pp. 267–271).

Høglund, S. (2014). *Autonomous inspection of wind turbines and buildings using an UAV.* Master thesis, Department of Engineering Cybernetics, NTNU.

Joint Rescue Coordination Centers (2012). Detailed Statistics of the Joint Rescue Coordination Centers. [Online] Available: `http://www.hovedredningssentralen.no/`.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering*, *82*(Series D), 35–45.

Leira, F. S. (2013). *Infrared Object Detection & Tracking in UAVs.* Master thesis, Department of Engineering Cybernetics, NTNU.

Martins, R., Dias, P. S., Marques, E., Pinto, J., Sousa, J. B., & Pereira, F. L. (2009). IMC: A communication protocol for networked vehicles and sensors. In *OCEANS - Aberdeen, Scotland*, (pp. 1–6).

MoJ (2002). The Norwegian Search and Rescue Service. Department of Civil Emergency and Rescue Planning, the Ministry of Justice and Police (MoJ). [Online] Available:
`http://www.redningsnett.no/Litteratur/Den-norske-redningstjenesten.`

Naidoo, Y., Stopforth, R., & Bright, G. (2011). Development of an UAV for search & rescue applications. In *IEEE AFRICON - Livingstone, Zambia*, (pp. 1–6).

Pachter, M. & Chandler, P. R. (1998). Challenges of autonomous control. *IEEE Control Systems*, *18*(4), 92–97.

Pinto, J., Dias, P. S., Martins, R., Fortuna, J., Marques, E., & Sousa, J. (2013). The LSTS toolchain for networked vehicle systems. In *MTS/IEEE OCEANS - Bergen, Norway*, (pp. 1–9).

Prevost, C. G., Desbiens, A., & Gagnon, E. (2007). Extended Kalman Filter for State Estimation and Trajectory Prediction of a Moving Object Detected by an Unmanned Aerial Vehicle. In *American Control Conference - New York City, USA*, (pp. 1805–1810).

QGroundControl (2014). The MAVLink Protocol. [Online] Available:
`http://qgroundcontrol.org/mavlink/start.`

Riseborough, P. (2014). Inertial Navigation Filter. [Online] Available:
`https://github.com/priseborough/InertialNav.`

Rudol, P. & Doherty, P. (2008). Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery. In *IEEE Aerospace Conference - Big Sky, Montana, USA*, (pp. 1–8).

Steen, T. A. (2013). *Search and Rescue Mission Using Multicopters.* Project report, Department of Engineering Cybernetics, NTNU.

Vik, B. (2012). *Integrated Satellite and Inertial Navigation Systems.* Department of Engineering Cybernetics, NTNU.

Voldsund, V. (2014). *Drop and recovery of sensor nodes using UAVs.* Master thesis, Department of Engineering Cybernetics, NTNU.

Waharte, S., Symington, A., & Trigoni, N. (2010). Probabilistic search with agile UAVs. In *IEEE International Conference on Robotics and Automation - Anchorage, Alaska, USA*, (pp. 2840–2845).

Waharte, S. & Trigoni, N. (2010). Supporting Search and Rescue Operations with UAVs. In *International Conference on Emerging Security Technologies - Canterbury, UK*, (pp. 142–147).

Wu, Z., Ding, X., & Qin, X. (2009). Research on Video Imagery Moving Target Intelligent Tracking and Locating Based-on Combined Model. In *Third International Symposium on Intelligent Information Technology Application - Nanchang, China*, volume 3, (pp. 666–669).

Zhiyuan, L., Hovakimyan, N., Dobrokhodov, V., & Kaminer, I. (2010). Vision-based target tracking and motion estimation using a small UAV. In *49th IEEE Conference on Decision and Control - Atlanta, Georgia, USA*, (pp. 2505–2510).