NTNU – Trondheim
Norwegian University of
Science and Technology

Surface Mapping using Quadcopter

Kenneth Eide Haugen

Master of Science in Cybernetics and Robotics
Submission date: June 2014
Supervisor: Lars Imsland, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

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

PROJECT DESCRIPTION SHEET

Name of the candidate: Kenneth Eide Haugen

Thesis title (English): Surface mapping using quadcopter

Background

As offshore oil- and gas production enters arctic seas, the presence of ice becomes a substantial challenge in station-keeping operations. An important part of such operations in the Arctic, is a system for doing *ice management*, that is, gather information about, and physically control, the ice environment. The required information about ice properties may come from several sources, but of interest in this project is sensors (e.g. optical or radar) that are carried by unmanned aerial vehicles (UAVs).

As a possible testbed for UAV guidance and estimation algorithms, it is proposed to use the quadcopter Parrot AR.Drone 2.0. This project should build on previous tasks on control and positioning of the quadcopter, and build a system for doing surface object mapping (to emulate e.g. mapping of icebergs, growlers and/or ice floes).

Work description

1. Give a brief overview of ice management challenges, and how UAVs can be used for this.
2. Describe the quadcopter, and how it might be used as a possible testbed for the purposes described above.
3. Give a brief introduction to surface mapping and UAV guidance (and navigation and control), and describe algorithms that may be used for UAV guidance for mapping of ice (icebergs, growlers or ice floes).
4. Interface a camera-based indoor positioning system for use with the quadcopter.
5. Implement a chosen guidance strategy for the quadcopter.
6. Use the guidance solution in a surface mapping strategy. By doing some simplifying assumptions, use images from the quadcopter to build up a “iceberg map”, using a computer vision library (OpenCV).
7. Do illustrative experiments, discuss the solution/implementation, and make proposals for extensions.

Start date: January 6, 20134 **Due date:** June 10, 2014

Supervisor: Lars Imsland
Co-advisor(s): Joakim Haugen

Trondheim, __March 20th, 2014_____

Lars Imsland
Supervisor

Address Sem Sælandsvei 5 NO-7491 Trondheim
Org.no. 974 767 880 E-mail: postmottak@itk.ntnu.no http://www.itk.ntnu.no
Location O.S. Bragstads plass 2D NO-7034 Trondheim
Phone + 47 73 59 43 76
Fax + 47 73 59 45 99
Phone: + 47 47 23 19 49

Abstract

This thesis studies the use of unmanned aerial vehicles to perform *ice management* in the Arctic Ocean by gathering information about and physically control the ice environment. Such a system is needed for safety reasons as marine operations are moving further north. In order to gather information about the ice environment, a UAV will be used for *surface mapping*. The quadcopter *Parrot AR. Drone 2.0* will be used as a testbed for implementing proposed strategies for guidance, navigation and control while doing surface mapping using a camera.

A guidance and navigation system is designed and implemented using measurements from onboard sensors and the camera system *OptiTrack*, which is used to measure position, velocity and orientation of the quadcopter. Using these estimated states as parameters and inputs to a proportional-integral-derivative controller, the position will be controlled. Waypoints are calculated according to desired parameters provided by an operator. An autonomous guidance, navigation and control system that moves the drone in a search pattern inside a desired area requested by the operator, is the result of the designed surface mapping strategy. An algorithm that performs object detection and mapping is implemented for the onboard camera to be able to detect objects in the lab setup. Back-projection of a 2D pixel point to respective world coordinates is implemented. *C++* is used for all modules.

Sub modules are simulated in *Matlab* and *Simulink* before tested with the AR. Drone. Simulations and measurements from lab testing are compared for performance evaluation. Results for the overall implementation shows that a UAV platform for doing object mapping is indeed a concept to pursue. However, this lab setup would not be applicable in a real world experiment. The AR. Drone will, due to its weight and limited power, not be able to operate under heavy wind and weather conditions. Also, detection of ice is more complicated than the suggested implementation, due to factors like weather and light reflections. It should be clear that this system design is rather a prototype illustration of a concept than a system to be used.

Sammendrag

Bakgrunnen for denne hovedoppgaven er bruk av ubemannede luftfartøy for å samle informasjon om isforhold i Nordishavet. Et slikt system vil kunne være svært nyttig av sikkerhetsmessige grunner ettersom marine operasjoner beveger seg lenger nord. For å kunne samle informasjon om isforholdene vil et ubemannet luftfartøy brukes for å utføre kartlegging av havoverflaten. Kvadrokopteret *Parrot AR. Drone 2.0* brukes som plattform for testing av implementerte strategier for styring og navigasjon mens den kartlegger overflaten med et kamera.

Et system for styring og navigasjon er designet og implementert ved hjelp av målinger fra sensorer ombord i dronen og kamerasystemet *OptiTrack*. Sensorene og kamerasystemet gir målinger av posisjon, hastighet og orientering. Disse estimerte tilstandene brukes som parametre og input til en PID-regulator for å styre posisjonen. Viapunkter beregnes i henhold til ønskede parametre gitt av en operatør. Et autonomt navigasjon- og reguleringssystem sørger for at dronen flyr i et søkemønster innenfor et ønsket område basert på den utarbeidede metoden for kartlegging av overaten. Algoritmen som utfører objektgjenkjenning og kartlegging er implementert ved bruk av bilder fra kameraet ombord kvadrokopteret, slik at objekter på bakken blir detektert. Projisering av en 2D piksel til et punkt i verdenskoordinater er implementert. *C++* benyttes for alle moduler.

Delmoduler er simulert og tester i *Matlab* og *Simulink* før de er testet med dronen. Simuleringer og målinger fra testing er sammenlignet for å evaluere ytelsen til modulene i systemet. Resultatene for den fullstendige løsningen viser at konseptet har et stort potensiale. Imidlertid må det bemerkes at dette oppsettet ikke er anvendelig i et virkelig eksperiment over havoveraten siden kvadrokopteret ikke er egnet til operasjoner i vanskelige vind- og værforhold på grunn av dens vekt og effekt. Deteksjon av is er mer omfattende enn hva som fremkommer i foreslått løsning på grunn av faktorer som blant annet vær og lysrefleksjoner. Dette systemet er ment som en prototype som viser konseptets potensiale, og ikke et system som kan anvendes direkte i virkeligheten.

Acknowledgements

This thesis reflects the work done in my Master's Project in the field of Engineering Cybernetics at the Norwegian University of Science and Technology.

I would like to express my grattitude to my supervisor Lars Imsland for useful guidance including ideas and feedback on my work. The interesting topic of my thesis and the way of addressing the problem at hand has motivated me to keep on working towards the ultimate goal despite a great number of bumps in the road. To me, this is really what the Master's thesis is about. I really appreciate how my supervisor let me explore various ways of solving technical problems while at the same time helping me stay on track with the problem in question.

Another factor that made this work possible is that I got to use the Sintef Snake Robot Lab. The OptiTrack camera setup at the lab has been of great value for the project. I am very thankful to Sintef and NTNU for letting me use the lab.

Furthermore I would like to thank PhD Candidate at NTNU, Eleni Kelasidi. She provided a lot of inputs on the OptiTrack Camera System. I really appreciate your kindness and eager to help. I also appreciate disussions on various topics with my co-advisor Joakim Haugen.

Contents

List of Figures

List of Tables

Abbreviations

UAV	**U**nmanned **A**erial **V**ehicle
GNC	**G**uidance **N**avigation and **C**ontrol
FOV	**F**ield **O**f and **V**iew
INS	**I**nertial **N**avigation **S**ystem
ODE	**O**rdinary **D**ifferential **E**quation
IMU	**I**nertial **M**easurement **U**nit
GPS	**G**lobal **P**ositioning **S**ystem
PID	**P**roportional-**I**ntegral-**D**erivative
NED	**N**orth-**E**ast-**D**own
UML	**U**nified **M**odeling **L**anguage
API	**A**pplication **P**rogramming **I**nterface
OpenCV	**O**pen source **C**omputer **V**ision
HSV	**H**ue **S**aturation **V**alue
ESSID	**E**xtended **S**ervice **S**et **I**dentification
IP	**I**nternet **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
CoG	**C**enter **o**f **G**ravity
MPC	**M**odel **P**redictive **C**ontrol
LQR	**L**inear-**q**adratic **R**egulator

Chapter 1

Introduction

Offshore gas and oil companies are constantly exploring new environments and locations in the search for energy sources. Going north in the Arctic seas, major challenges arise. Icebergs, growlers or ice floes occure frequently. Ships and platforms used to retrieve the energy sources in question are exposed. The story about the Titanic is well known, and even though technology has come far, there is still a way to go before such operations are safe.

Today, there are many instruments and strategies used to avoid large icebergs. Several of these are reliable systems, i.e. using a radar onboard the ship one can detect close icebergs. Detecting icebergs, growlers or ice floes further away might be challenging with this technology. Another idea is to use *unmanned aerial vehicles* (UAVs) with onboard sensors to spot icebergs, growlers or ice floes. Using the data and estimations from the UAV, it is possible to find where the ice is located and if it pose a threat. Figure 1.1[1] illustartes a ship with surrounding ice.

The motivation for this thesis is based on the problem stated above. A *quadcopter* is a UAV with four rotors. It can be remotely or autonomously controlled, and has therefore become popular both in research and for educational purposes. A quadcopter interfaced with a positioning system and a camera to capture images

[1]Figure taken from http://www.quarkexpeditions.com/en/arctic/expeditions/iceland-circumnavigation/ship-information

FIGURE 1.1: Thesis motivated by marine operations in the Arctic Ocean

from the drone can be used to map a wanted area in the ocean. Images from the camera can be used to determine sizes of the ice. A surface mapping system will require an autonomous motion control system that is used to send commands to the drone. It will also need a navigation system to determine where the ice is located. When the location of ice is known, an algorithm to plan a path for the ship can be applied. Doing *ice management* is an application for the surface mapping system that is to be designed.

1.1 Ice Management

Ice management can be defined as gathering information about and physically control the ice environment. In the specific problem addressed in this thesis, the required information about ice properties will be retrieved from a camera. The properties of interest may be size and thickness of the ice, its position and the velocity if moving. Using this information, one can conclude whether or not the ice is a threat, and what to do next.

This thesis will not focus on strategies to physically intervene with the ice environment. Some of the challenges will be stated, though.

If there is solid ice between two coordinates, and a ship want to travel from one coordinate to the other, there are two alternatives:

- Move around the ice in an alternative route.
- Create a way through the ice.

In some cases, it is not possible to find another route. Physical ice management such as ice breaking can be done. This requires external equipment like ice breaker vessels. In a case where the ship or platform is stationary, the same principle holds if ice is floating towards the object. Algorithms to determine the direction and velocity of moving ice must be implemented. This is not the goal for this project, but a UAV carrying various sensors will be used to gather the required information in order to be able to move around the ice.

1.2 UAV as Testbed for Ice Management

To be able to illustrate how the system can work in practice, the concept will be implemented in an indoor lab with the *Parrot AR. Drone 2.0*. The AR. Drone was originally created as a toy to be mannually controlled from a smartphone. Soon, developers started to implement their own software, and this will also be done in the thesis at hand.

Ultrasonic sensors, onboard camera and a camera-based indoor positioning system will be interfaced with the AR. Drone program for guidance, navigation and control. Commands will be sent wirelessly from a computer to the quadcopter, and the quadcopter will send navigation information and images back to the remote computer. Image recognition will be implemented using a library for the *C++* programming language, *OpenCV*, and used to detect objects emulating ice.

A path tracking controller will be implemented to autonomously control the velocity and position of the drone. Waypoint guidance will be used to define a wanted

search pattern that should cover the area where the operator want to search for icebergs. Throughout the thesis, icebergs, growlers or ice floes are mostly referred to as *objects*, as various objects are placed on the floor when testing the overall surface mapping strategy.

1.3 Project Description and Limitations

Approaching a complex project dependent on several main modules and functionalities to communicate and work together, a superior project description and limitations will shape a picture of the work to be done. In general, surface mapping from a UAV aircraft includes guidance, navigation and control of the vehicle, interaction with a positioning system, computers as well as hardware interaction and image processing. An important factor for beeing able to successfully implement such system is to devide it into sub modules. The thesis focuses on, but is not limited to the following main goals and modules:

- The Parrot AR. Drone will be used as a testbed for surface mapping using a camera.
- Guidance, navigation and control strategies will be implemented for the quadcopter in order to be able to perform object detection and surface mapping.
- The camera-based positioning system OptiTrack will be interfaced for use with the quadcopter as its main position, velocity and orientation measurement source.
- C++ and OpenCV will be used in the implementation of the overall concept.
- All tests are performed in the *Sintef Snake Robot Lab* at NTNU.

Several modules that one could expect to be part of this thesis are simplified or neglected. Some of those are given next.

- 3D mapping and modeling of detected icebergs. Mapping is done in 2D.
- Estimation of object or iceberg movements.
- Route planning and GNC for a ship that will move from one position to another using mapping data.

1.4 Outline of Thesis

The thesis is devided into 9 chapters. Each chapter will focus on different stages of the work, which means the same topic may be visited multiple times. The structure is chosen to seperate description of a chosen method from design, implementation, testing and results.

Next chapter will do a brief literature review of previous work related to the subject of this thesis. Short summaries and critique on relevant parts will be given. Chapter 3 describes the AR. Drone platform included hardware and software. Coordinated systems and important notation is given before a mathematical model is presented. Structure of the quadcopter control is also proposed. Chapter 4 introduces methods and theory for indoor positioning and surface mapping necessary to understand the system design, included the chosen method for camera-based indoor positioning. It also provides an introduction to surface mapping using image recognition techniques.

Chapter 5 presents the overall system design. Drone positioning, path tracking controller design and object detection and identification are main focuses. As the strategies are presented, Chapter 6 presents how the overall strategy is implemented. Guidance, navigation, control and object detection will be revisited. In Chapter 7, tests are performed for positioning and object detection. Test results are discussed in Chapter 8, before conclusions and recommendations for future work concludes the thesis in Chapter 9. Though a separate chapter is dedicated for discussion, result and finding will be briefly discussed throughout the thesis.

Chapter 2

Literature Review

UAVs have during the last few decades been used for various purposes. Many researchers have worked on developing maneuvering algorithms and applications for quadcopters. This chapter will present relevant previous work on the field. In addition to presented literature, various references are used throughout the thesis, and will be presented when convenient.

2.1 Software Infrastructure for a UAV Testbed for Ice Management Guidance and Estimation Algorithms

The thesis at hand is a continuation of the work done in the master thesis *Software Infrastructure for a UAV Testbed For Ice Management Guidance and Estimation Algorithms*[2] by Raaen.

Guidance, navigation and control is defined in the thesis as:

- Guidance is the action or the system that continuously computes the desired position, velocity and acceleration of a marine craft to be used by the motion control system.

- Navigation is the science of directing a craft by determining its position/attitude, course and distance travelled.
- Control, or more specifically motion control, is the action of determining the necessary control forces and moments to be provided by the craft in order to satisfy a certain control objective.

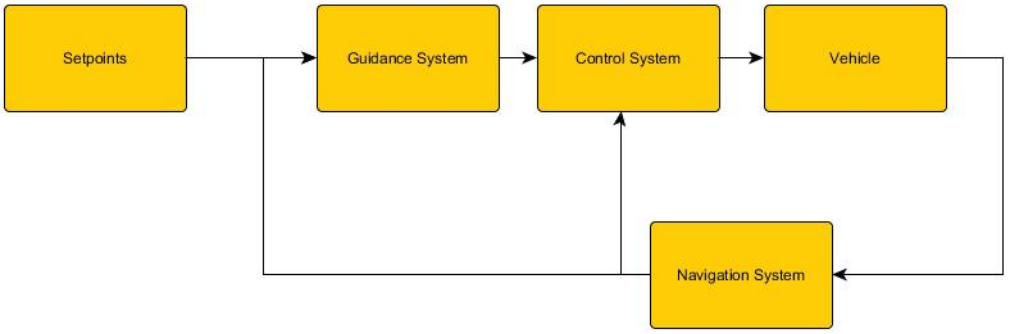

FIGURE 2.1: Block diagram of a GNC system for a given vehicle [2]

A strategy for surface mapping is described, but this is not directly relevant for the indoor lab setup to be implemented in this thesis. An algorithm for sea ice boundary detection from satellite images is presented, where it is made clear the the difference between an edge and a boundary must be defined. A boundary consisting of edges must be detected in order to decide if an object is detected. The described surface mapping strategy has been used on satellite images of floating ice, and the general algorithm can be of use in outdoor experiments.

The thesis at hand will not make use of the incomplete implementation done in the thesis covered in this section. Due to equipment failure, no implementation was tested, and the report is thus not fully reliable. Several findings will though be taken into account. Concluding that Matlab is not a well fitted programming language for the purpose of this project is one of those useful findings. Even though no results were achieved for the complete scenario, some suggested methods and research are helpful.

2.2 Camera-Based Integrated Indoor Positioning

Positioning of the quadcopter is an important aspect in order to be able to fly autonomously and send coordinates for locations of floating ice. In the thesis *Camera-Based Integrated Indoor Positioning* [3], Vintervold designed and implemented an integrated positioning algorithm for the AR. Drone inside a lab setup. The algorithm will not be used, but results and conclusions are of interest.

A camera-based positioning system serves as the main measurement source in this thesis. The system delivers position and orientation measurements based on marker tracking through the use of cameras placed along the ceiling in a lab setup. The chosen camera system is *OptiTrack* from *NaturalPoint*. The measurements are used by two implemented positioning algorithms. Equipment from OptiTrack is used to track the quadcopter while streaming the obtained information to Matlab for use in an extended Kalman filter. The Optitrack tracking software *Tracking Tools* is used from a computer running a Windows operating system, and measurements of interest are sent to a Matlab session for further processing.

To be able to track the drone in the lab, at least three cameras must see a reflective marker at all times. A capture volume for the cameras tracking the drone is shown in Figure 2.2.

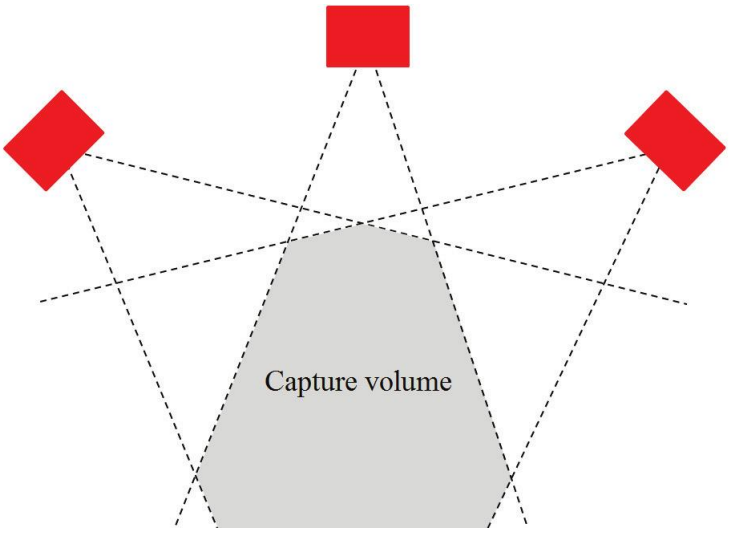

FIGURE 2.2: The resulting capture volume from three (red) cameras [3]

The thesis at hand will use the OptiTrack camera system as its position and orientation estimates, which are inputs to a position controller. Unlike measurements from an IMU, position estimates will not drift with time in such indoor navigation system. However, the drone must stay within the defined capture volume. In general, an *integrated* navigation system that combines a global positioning system with the IMU has the best performance for position estimates. Vintervold investigates this assumption.

An integrated system is implemented to combine the camera and INS algorithms. The intention is to provide redundancy in the sensors while achieving equally frequent state estimate updates with respect to the results obtained using the camera system only. In an outdoor experiment with a drone, the camera setup would have been replaced by a GPS. GPS/INS integration is often used for similar purposes. Details on this subject will not be investigated, but the findings are of interest.

Tests of the camera measurement algorithm were compared with the integrated navigation system. No definite conclusions were drawn, but results implied that the accuracy and measurement noise of the integrated navigation system was not improved compared to the camera system measurements. Results were that the integrated system had increased noise in the velocity estimates when compared to the camera measurement algorithm. Though further tuning and testing of the integration filter was recommended, the conclusion was a bit astonishing at first. When taking into account the high update rate of the camera measurements and the high accuracy resolution of the camera system, it is easier to understand the results. Updating and correcting measurements that are already accurate to millimeters and updated about 100 times per second with dead reckoning measurements that might not be as accurate on this scale, might be a source of increased noise.

Although above arguments are not verified by thorough testing in this thesis, it is concluded that the camera measurements will provide sufficient accuracy in

position and orientation estimates. An intregrated navigation system will though not be implemented in the thesis at hand, as it will not provide any advantages in the indoor lab tests. For outdoor use with GPS, an integrated system should be implemented.

2.3 Quadrotor Helicopter Trajectory Tracking Control

The Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) is a quadrotor helicopter project developed by Hoffmann et al. as a testbed for several algorithms that enable autonomous operation of UAVs. The paper [4] develops a trajectory tracking algorithm as well as a time optimal feasible trajectory generator. Trajectory tracking control is often designed as a nonlinear optimization problem. This paper presents an alternative approach that might be easier to implement with UAVs.

Quadcopter control included attitude control and a path tracking controller is designed and implemented. For path tracking, a PID controller is applied. Test results are illustrated and shows that the controller keeps the position close to a reference trajectory. Figure 2.3 is taken from the paper, and indicates that the path is not perfectly smooth. Turns are not handeled explicitly, and overshoots due to infeasible reference trajectories occure. The generic PID controller structure is an advantage for such project, and may with minor improvements prove to perform according to expectations.

A desired speed profile is generated in a dynamically feasible manner. A space-indexed speed profile that traverses waypoints in minimum time while satisfying speed and acceleration constraints are designed and simulated, but not thoroughly tested. This way of generating new reference paths and velocities is more useful when time optimality is an important factor. However, the algorithm is more complex and takes more computation time. Computation time is not a critical concern, as most calculations are done on a remote computer in this thesis. Time

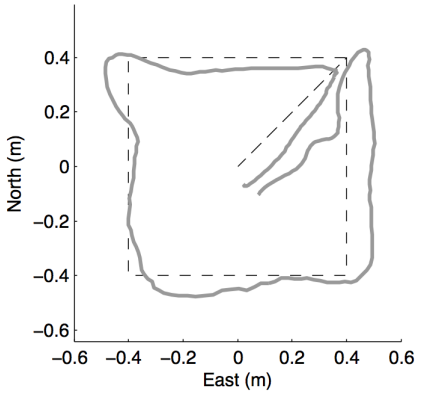

FIGURE 2.3: Measured path of position using PID controller

optimization during a flight is not crusial for the surface mapping strategy. For simplicity in lab testing, path planning will be simplified compared to the final product of the STARMAC project.

Tests of the trajectory tracking algorithm are performed both in an indoor lab and outside at Stamford University, California. Indoor performance is tested to have an accuracy of about 10 *cm*. A goal for the position controller in this thesis is to achieve the same, or even higher accuracy.

Chapter 3

Parrot AR. Drone

As mentioned, the Parrot AR. Drone was originally created as a toy for smartphone games. Developers realized that others could design their own programs and functionality by accessing the source code for the drone, which resulted in people using it for educational purposes. To be able to understand how to implement new functionality to the platform, one must look into how the drone operates and how it can cummunicate with a remote computer. The *inertial measurement unit* will be introduced along with other hardware and software of the drone. To be able to design and implement an autonomous object mapping strategy, important notation and terminology as well as theory and methods for mathematical modeling of the drone is presented. Basic quadcoper control concludes the chapter.

3.1 AR. Drone as Testbed for Object Mapping

Object mapping at sea surface can be done in different ways. One example is using algorithms for object detection on satellite images. It could also be possible to use the same technique with a camera mounted on a ship or a remote boat.

For both examples, there are major challenges. Satellite images are not updated at a high rate and are thus unreliable. As long as the icebergs in question are moving, this strategy would not be ideal. The same goes if the weather prevents the camera from being able to perform object detection. Mapping from a ship would make it possible to see if the ship is appoaching an iceberg, but an effective strategy for finding alternative routes would be problematic if not impossible. Existing technology like using a radar instead of a camera is applied for the purpose, and provides more reliable information than a camera due to weather.

Using an aerial vehicle for this purpose would on the other hand provide the advantage with mapping the area from above, as well as being able to equip the drone with different sensors. Doing this in an indoor lab with the toy quadcopter from Parrot will not be representative for outside conditions, but the concept will be presented. Onboard sensors for measuring orientation and location as well as onboard cameras makes the drone a good fit as a testbed for illustrating the experiment. Figure 3.1[1] shows the quadcopter.

FIGURE 3.1: The AR. Drone 2.0 used as a testbed for the surface mapping implementation.

[1]Figure taken from http://www.clubic.com/mobilite-et-telephonie/objets-connectes/

3.2 Hardware

3.2.1 Moving the Quadcopter

For now, it is clear that a quadcopter has four rotors that decide how it can move in 3 dimensions. A quadcopter has four controllable degrees of freedom: roll, pitch, yaw and altitude. Each degree of freedom can be controlled by adjusting rotor thrusts.

Two of the four rotors rotate clockwise, while the other two rotate counterclockwise. Each pair of propellers that rotate the same direction are placed diagonally opposite each other. Figure 3.2 clarifies the placements. If all propellers rotate at the same rate, the net torque will be zero. This results in the quadcopter hovering.

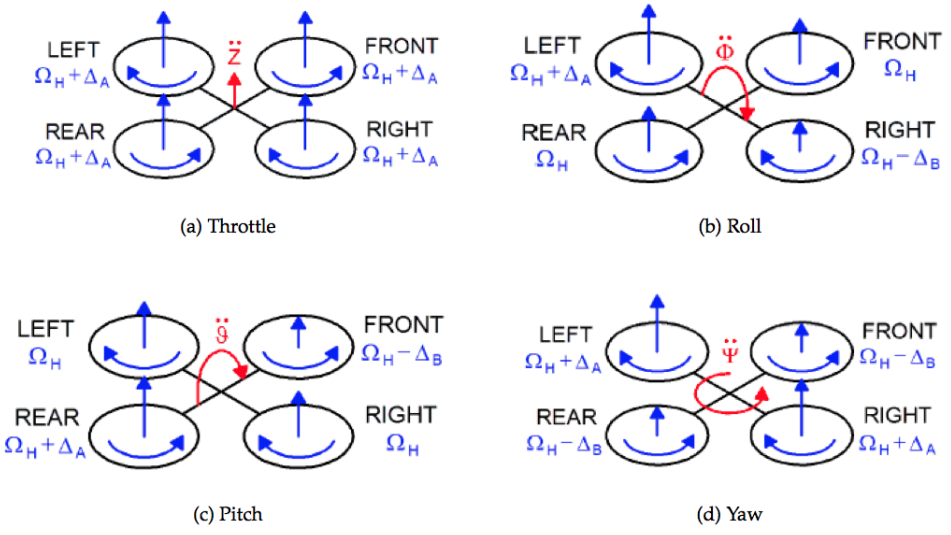

FIGURE 3.2: Changing the different degrees of freedom [5].

Figure 3.2 shows how changing the degrees of freedom makes the drone move. The curly blue arrows represent the direction of rotation for the propellers. Straight blue arrows represent a thrust difference from the rotors needed to achieve a change in the specified degree of freedom represented by red arrows.

Ω_H in the illustration are constant angular velocities for the propellers. Δ_A and Δ_B represent different changes in the angular velocity. To change the altitude of the

drone in air, all propellers must rotate at the same frequency. By increasing the thrust for all rotors, the altitude will increase. This is illustrated in case *a* in the figure. Case *b* shows that the left and right rotor velocities must change according to each other to make the quadcopter *roll*, that is, tilt left or right. For the drone to move forward or backward, the *pitch* angle must be changed. This is done by putting a higher thrust on the rear rotor and a lower thrust on the front rotor. This means that

$$\Omega_H + \Delta_A > \Omega_H - \Delta_B$$

The last controllable degree of freedom is the *yaw*. The quadcopter will turn left or right about its own axis if both the front and rear (or left and right) rotor velocities are decreased/increased with the same rate. The reason why the drone will rotate in this case is because the net torque is no longer zero. In case all rotors were spinning in the same direction, the drone itself would spin at the same velocity.

3.2.2 Motor

The quadcopter used as a testbed in this project has brushless motors with a microcontroller that controls the three-phase currents. A safety mechanism is implemented in the microcontroller that makes the propellers stop rotating if they are exposed to obstruction of a given amount. This means there is no need to implement a safety mechanism when testing the setup at the lab.

The four motors have an effect of 15 W each, and can rotate with a velocity of 35 000 rotations per minute. With a drone mass of only 420 g including the indoor hull [6], the motor power is sufficient for testing of the concept inside. For a working prototype to be tested in reasonable weather conditions, the light weight and low effect motors are not usable. Further arguments will be discussed in Chapter 8.

3.2.3 Battery

The AR. Drone applies a three cell lithium polymer battery with 1500 mAh and $11,1\,V$. The battery lasts for about 16 minutes in air under normal weather conditions. With today's battery technology there are challenges keeping the battery weight low while maximizing effect. In an outdoor experiment, there are challenges related to long-term operations in air. The thesis will not go into detail on this aspect, but it is important to be aware of the challenges.

3.2.4 Inertial Measurement Unit

Navigation is crucial to be able to perform autonomous flight. An *inertial navigation system (INS)* is a system consisting of an *inertial measurement unit (IMU)* and software to compute position, attitude and velocity from measurements retrieved from sensors. A system which is mounted on the object of interest is called a *strapdown* inertial navigation system. The AR. Drone is equipped with such system, which can be used to provide autonomous flight.

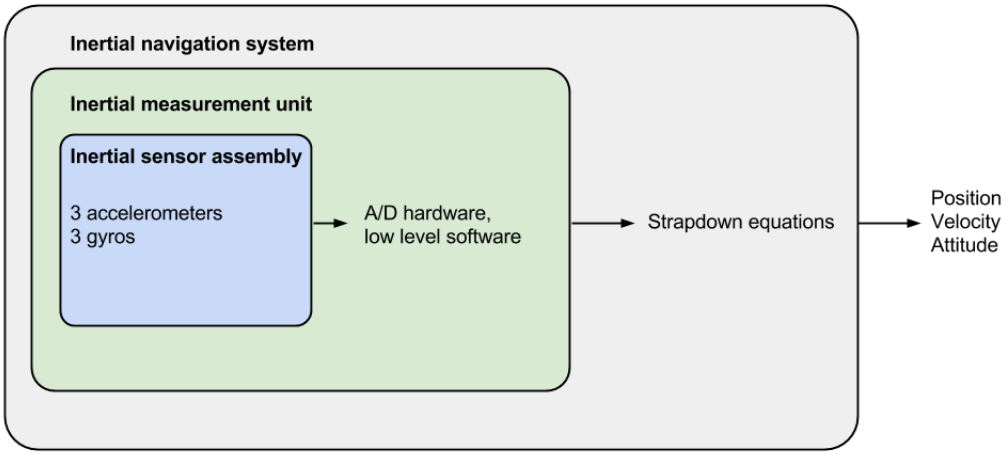

FIGURE 3.3: Schematic structure of an inertial navigation system.

The AR. Drone features a 6 DOF IMU, containing a 3 axis accelerometer, a 2 axis roll and pitch gyrometer and a single axis yaw gyrometer. That is, three gyroscopes for measuring angular velocities and three accelerometers for linear

acceleration measurements. Using data from these sensors, wanted parameters can be calculated. The implementation will include relevant calculations. This thesis will not focus on the structure and operation of the accelerometers and gyroscopes, but output models are rewritten from Vintervold[3], p. 47:

$$\mathbf{a}^b_{IMU} = \mathbf{a}^b_{meas} + \mathbf{b}^b_{acc} + \mathbf{w}^b_{acc} = [a_{IMU,x}, a_{IMU,y}, a_{IMU,z}]^T \tag{3.1}$$

where $\mathbf{a}^b_{meas}$ is the *specific force*, i.e the acceleration relative to free fall, decomposed in the *body* frame (to be defined in Section 3.4). The output is also influenced by a bias $\mathbf{b}^b_{acc}$ and measurement noise $\mathbf{w}^b_{acc}$. Accelerometer measurements are based on Newtons Second Law, using that the observed weight of a proof mass changes during acceleration.

The gyroscope output is given as:

$$\boldsymbol{\omega}^b_{IMU} = \boldsymbol{\omega}^b_{b/i} + \mathbf{b}^b_{gyro} + \mathbf{w}^b_{gyro} = [\omega_{IMU,x}, \omega_{IMU,y}, \omega_{IMU,z}]^T \tag{3.2}$$

Biases $\boldsymbol{\omega}^b_{b/i}$ and noise $\mathbf{w}^b_{gyro}$ is also included in these measurements. The gyroscope measures angular velocity in degrees per second, based on the principles of angular momentum. Most gyroscopes, especially for *gimbal* systems, use a spinning wheel that utilizes conservation of momentum to detect rotation. However, the AR. Drone is too small to use this technology. A vibratory gyro with a proof mass is used instead. The mass vibrates, and when the wehicle rotates, the proof mass gets displaced by Coriolis forces. Angular velocities are measured from the law of Coriolis. [7]

Measurements from the accelerometers and gyroscopes are used to stabilize the drone. It can also be used for inertial navigation. Further details will be given in chapters 5 and 6.

Two ultrasonic sensors are used to measure the altitude of the drone. The sensor range is about 6 meters, and is thus applicable in the lab setup. The principle of this sensor is not described here, but it basically operates by sending ultrasonic waves

towards the ground while measuring the time until the reflection is reveived at the sensor.

3.2.5 Integrated IMU and Computer Vision Based Navigation Strategy

As the accelerometer and gyro measurments contain bias and noise, attitude estimates will be inaccurate. A vision based system using the drone camera is integrated to estimate and compensate for the issues with using gyro and accelerometer measurements only. At the same time, inertial sensors are used to compensate for micro-rotations in the images of the camera. The vision based system improves the velocity estimates. However, the vision-based velocity is also noisy and relatively slowly updated compared to the vehicle dynamics. Estimated velocity is improved with the help of an aerodynamics model. This section is based on research made by Bristeau, Callou, Vissiére and Petit [8].

Vision based (onboard camera) and inertial navigation are tightly integrated. When both vision based and inertial sensor velocity estimates are available, accelerometer bias are estimated and vision velocity is filtered. When vision velocity is unavailable, only the inertial sensor estimate is used with the last updated value of the bias. Figure 3.4 illustrates velocity estimates from navigation techniques used on the AR. Drone. The red line represents velocity estimate outputs that can be used with the dead reckoning positioning system presented in Section 5.2.2.

Dynamics of a quadcopter is quite complex. In particular, the aerodynamics of the propellers and the motion of the rigid frame is of interest. Linear drag term exists from the interaction between the rigid body and the rotors. A tilt phenomenon changes a lift force component in drag, which yield interesting information on the velocity of the system. This model will not be a focus in this thesis as it is implemented in the closed source firmware. Details on a proposed model can be found in [8]. Theory on quadcopter dynamics are given in Section 3.5.3.

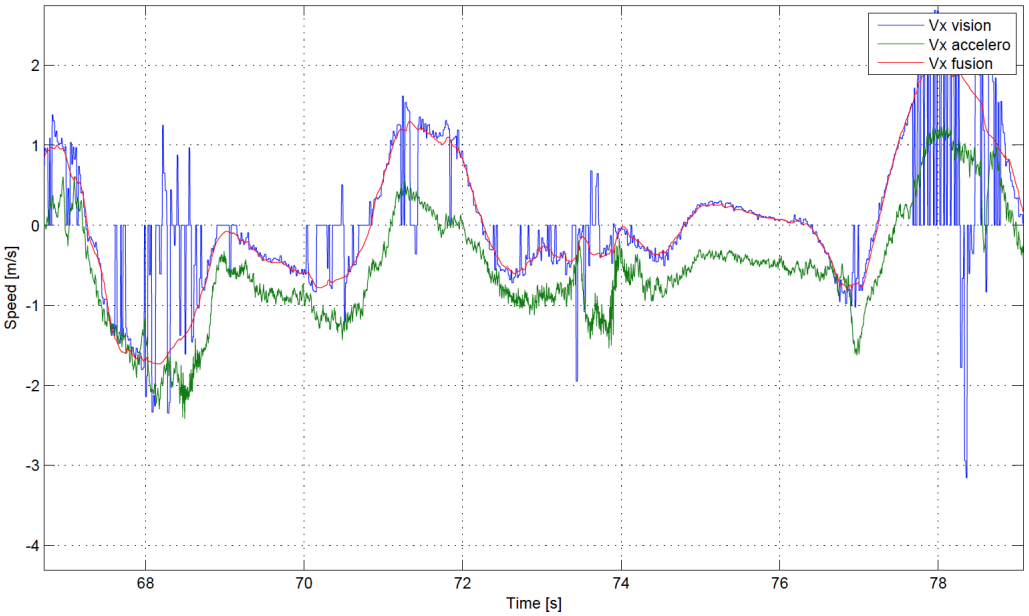

FIGURE 3.4: Example of velocity estimates: computer vision velocity estimate (blue), aerodynamics model velocity estimate from accelerometer (green) and integrated velocity estimate (red). Figure taken from [8].

3.2.6 Camera

Two cameras are mounted on the drone. One pointing horizontally in the X-direction while the other is pointing vertically downwards. The front camera has the highest resolution (1280×720 pixels) with a 93 degrees angle of sight. This camera will not be used for object mapping, but it would be well fitted for taking landscape pictures or for camera navigation algorithms. The vertical camera has a lower resolution (640×360 pixels) with 54 and 34 degrees angles of sight. The frame rate is 60 frames per second. These specifications are sufficient to perform simple image recognition techniques from pictures taken with the camera. How this object detection algorithm is implemented will be described in Chapter 6.

3.3 Software

The quadcopter comes with some basic but necessary functionality. As already described, several sensors and cameras are mounted. A safety mechanism for the

propellers is also included. In addition, basic maneuvers and stabilization of the drone is implemented in its firmware. Description on this implementation will not be given in detail as they will not be of direct relevance. However, a proposed structure for the overall controller system will be presented in Section 3.6.

3.3.1 Network and Communication

To control the drone from an external device or a computer, it is equipped with a WiFi network card. When the quadcopter power is turned on, it establishes a network connection with an ESSID name while an available IP address is allocated. Implemented programs for the drone can thus be run from a remote computer.

Movements of the drone will be controlled by sending a stream of commands wirelessly from the computer. These commands are normally sent 30 times per second. Similarly, the drone sends data to the computer at the same rate. These *navdata* are acquired from the accelerometer, gyroscope and ultrasonic sensors, as well as camera feed.

3.3.2 Software Development Kit

A *software development kit* (SDK) has been published at the AR. Drone community web page [5]. The kit consists of a necessary code basis to get started developing new applications for the AR. Drone. This includes firmware for the quadcopter, communication protocols, libraries for developing applications and simple code examples. A library based on this SDK will be used as a starting point in this project.

3.4 Coordinate Systems

To be able to control and navigate the AR. Drone while doing image processing, coordinate systems and reference frames must be explained and initialized. Four coordinate systems will be used:

- Geographic *North-East-Down* (NED) (n) frame, also called *world* frame (w).
- Vehicle *body-fixed* (b) frame.
- *Camera* frame (c) fixed to the camera onboard the quadcopter.
- *Image* frame (i) based on how pixels are defined in the image processing library.

The *world* frame will be applied when mapping the surface, and thus give the location of icebergs and desired waypoints. The *body* frame will be fixed to the drone and used for guidance, navigation and control. *Camera* and *image* frames are needed when designing the object detection algorithm and will be used when finding object positions. The right-hand rule is used to find directions of rotations.

3.4.1 World NED Frame

This frame is defined relative to the ground, and is here assumed to be inertial. Generally, the X-axis points towards true north, the Y-axis points towards east while the Z-axis points downwards perpendiculary to the tangent plane of the earth ellipsoid [9]. In this project, the frame will be fixed to the lab room. Figure 3.5 illustrates the world frame. Each iceberg will be given x_w and y_w coordinates. z_w will for simplicity be neglected, as one can assume the height of the object above ground level is not of critical interest.

3.4.2 Body-fixed Frame

The body-fixed frame is moving and rotating with the quadcopter. The x-axis points in the forward direction, the Y-axis to the right and the Z-axis points vertically downward relative to the orientation of the drone. Measurements from the IMU or a global positioning system relates this frame to the world frame through the Euler angles roll, pitch and yaw [9].

3.4.3 Camera Frame

To be able to transform a 2D pixel to 3D coordinates, the camera frame is used. This frame is translated according to the image frame and rotated according to the world frame. Object positions are not of interest in the camera frame, and will thus be transformed back to the world frame.

3.4.4 Image Frame

This frame is defined by the image processing library to have its origin in the top left corner of the image frame. Parameters given in the image frame defines the camera matrix to be introduced in Chapter 5.

3.5 Mathematical Modeling

To be able to control and navigate any vehicle, the system must be mathematically modeled. With six degrees of freedom (three translational and three rotational) and only four independent inputs (rotor speeds), quadcopters are severely under-actuated. Rotational and translational motion are coupled to achieve six degrees

[2]The X and Y axis are rotated by 45 *deg* in the figure to make the direction of the front camera point in the x direction. This rotation is done in the firmware. modeling is done *before* rotatating the frame

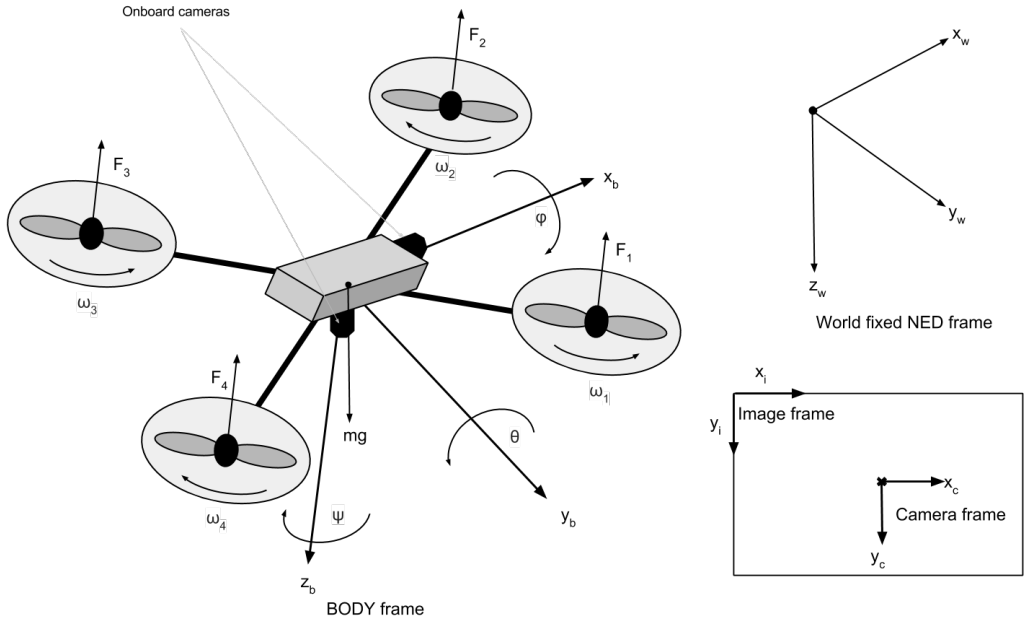

FIGURE 3.5: Illustration of coordinate axis for body frame, world NED frame, camera frame and image frame. Changing the Euler angles roll, pitch and yaw results in movements, as described in Section 3.2.1. x and y axis are initially fixed to arms connecting propellers 2 and 1 respectively[2]. Frames are rotated with respect to each other, using the right-hand rule.

of freedom. The dynamics for the quadcopter are highly nonlinear. As quadcopter stabilization is not a main focus of this thesis, a simplified model of the quadcopter dynamics and the on board controllers will be presented. Basic notation and a dynamic model of the quadcopter is given in this section, based on coordinate system and free body diagram given in Section 3.4. The following is motivated by Vik[9].

3.5.1 Notation

The desired positions of icebergs is presented in the world frame.Vectors in the world frame for the center of mass position of the drone (d) and the object (o) are

given as:

$$\mathbf{r}_d^w = \begin{bmatrix} x_{drone} \\ y_{drone} \\ z_{drone} \end{bmatrix} \quad , \quad \mathbf{r}_o^w = \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \end{bmatrix} \tag{3.3}$$

where the object position $\mathbf{r}_o^w$ is given as a function of the drone position $\mathbf{r}_d^w$. Further details will be given in Section 5.4. Vectors are written in bold with a superscript showing which frame it is decomposed into.

As stated, rotations about the coordinate axis results in drone movements. The *attitude*, i.e. the angular position, is defined in the body frame with the three Euler angles $\mathbf{\Theta}$.

$$\mathbf{\Theta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{3.4}$$

Section 3.2.1 describes resulting movements when changing the angles in $\mathbf{\Theta}$. ϕ, θ and ψ are roll, pitch and yaw, respectively.

Linear and angular drone velocities are given in the body frame as $\mathbf{v}^b$ and $\mathbf{\Omega}^b$. Velocities can also be presented in the fixed world frame coordinates ($\mathbf{v}^w$) from Figure 3.5.

$$\mathbf{v}^b = \begin{bmatrix} v_x^b \\ v_y^b \\ v_z^b \end{bmatrix} \quad , \quad \mathbf{v}^w = \begin{bmatrix} v_x^w \\ v_y^w \\ v_z^w \end{bmatrix} \quad , \quad \mathbf{\Omega}^b = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.5}$$

The vector $\mathbf{wp}^w$ contains waypoints in the world frame. This vector will be updated with new waypoints/setpoints in the world frame as the drone approaches

the wanted coordinates.

$$\mathbf{wp}^{w} = \begin{bmatrix} x_{wp} \\ y_{wp} \\ z_{wp} \end{bmatrix} \tag{3.6}$$

3.5.2 Rotations Between Frames

The body frame for the quadcopter is rotated in the world frame when it moves in space. Such rotations are of relevance when designing the controller. The other way around, transformations and rotations from body to world frames can be used to estimate velocities and position of the drone in case an external navigation system is not available. Common for all rotations between frames is that roll, pitch and yaw angles are angles to be measured and embedded into the model of the drone.

A *rotation matrix* is an orthogonal 3×3 matrix that represents rotations in a coordinate system. When multiplied with a vector, the vector is rotated from one frame to another [9]. Rotations are presented from the *body* frame to the *world* frame as well as rotations from *world* to *camera* frame. First, individual rotations are presented.

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad , \quad \mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The notation $R_{x,\phi}$, is a rotation of ϕ degrees about the x axis, and the other rotations are similar. The resulting rotation matrix rotating a vector from the *body* frame to

the *world* frame is

$$\mathbf{R}_w^b = \mathbf{R}_{z,\psi}\mathbf{R}_{y,\theta}\mathbf{R}_{x,\phi} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi s\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{3.7}$$

where sx = sin(x) and cx = cos(x). For more details on properties of rotation matrices, see [Vik, p. 9]. Rotations from the *world* frame to the *camera* frame are slightly different (coordinate system is rotated, which is the inverse of vector rotations), using the following basic rotations, and multiplying in the opposite order.

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \quad , \quad \mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Above matrices are inserted in equation 3.8.

$$\mathbf{R}_c^w = \mathbf{R}_{x,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{z,\psi} \tag{3.8}$$

3.5.3 Modeling of Quadcopter Dynamics

Properties and specifications for the AR. Drone as well as a definition of coordinate systems are given. To understand how quadcopter control operates, a mathematical model is derived. As stated earlier, the exact models applied in the AR. Drone firmware is not published. A generalized and simplified model is thus presented based on the article *Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors*[10]. Certain assumptions are made:

- The quadcopter is supposed symmetrical and rigid.
- The CoG and the body fixed origin are assumed to coincide.
- The propellers are supposed rigid.

Forces on the system are gavity in the z_w direction and forces from each rotor F_i in $-z_b$ direction. Wind forces and drag of the propellers are neglected. Newtons second law is used to derive an expression for the external forces on the quadcopter (one can also use the Lagrangian equation to derive the model). Forces $\mathbf{F}^w$ and torques (moments) $\mathbf{T}^b$ are given as

$$\mathbf{F}^w = m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - \mathbf{R}_b^w \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} F_i \end{bmatrix} \quad , \quad \mathbf{T}^b = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ T_1 - T_2 + T_3 - T_4 \end{bmatrix} \tag{3.9}$$

where $\mathbf{F}$ could also have been written in the body fame using the rotation $\mathbf{R}_w^b$ on gravitational forces. $\mathbf{T}_i$ are torques acting about the z_b-axis. Rotor forces will cause moments about x_b and y_b with arm L.

Next, the kinetics will be modeled to derive the complete dynamic model for the system. The Euler equation gives the torque balance in the body frame as

$$\mathbf{T}_b = \mathbf{I}_b\dot{\mathbf{\Omega}}_b + \mathbf{\Omega}_b \times \mathbf{I}_b\mathbf{\Omega}_b$$

which can be rewritten as

$$\mathbf{I}_b\dot{\mathbf{\Omega}}_b = \mathbf{T}_b - \mathbf{\Omega}_b \times \mathbf{I}_b\mathbf{\Omega}_b \tag{3.10}$$

where $\mathbf{I}_b \in \mathbb{R}^{3x3}$ denotes the moment of inertia. The total angular momentum of the rotors is assumed to be near zero as counter-rotating propellers make the yaw rate change close to zero. Using the assumption that the structure is symmetrical,

I can be given as

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.11}$$

The kinematic differential equations are found using that total angular velocity is the sum of velocities for each rotation. Rotations about coordinate axis that occure when changing roll, pitch and yaw are multiplied to find that $\mathbf{\Omega}^b$ is related to $\dot{\mathbf{\Theta}}$ according to

$$\mathbf{\Omega}^b = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\cos\theta\sin\theta \\ 0 & 1 & \sin\phi \\ \sin\theta & 0 & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.12}$$

Inverting equation 3.12 gives an expression for $\dot{\mathbf{\Theta}}$. Combining this equation with equation 3.10 results in 12 ODEs for the system where controllable inputs (total lift force, roll, pitch and yaw moments) of the underactuated system are given from u_1, u_2, u_3 and u_4 respectively. The moment produced on the quadcopter is opposite to the direction of rotation of the blades. Rotors 2 and 4 rotate in the clockwise direction about the z_b axis, while 1 and 3 rotate counterclockwise.

$$\begin{aligned} u_1 &= \sum_{i=1}^{4} F_i/m \\ u_2 &= L(F_2 - F_4)/I_1 \\ u_3 &= L(F_3 - F_1)/I_2 \\ u_4 &= (T_1 - T_2 + T_3 - T_4)/I_3 \end{aligned} \tag{3.13}$$

Each motor has an angular speed ω_i and produces a vertical force according to $F_i = k_f\omega_i^2$, where k_f is a constant to be parameterized when designing the model. Details on the motor model is not discussed further.

By assuming small Euler angles and small angular velocities, the equation set of 12 ODEs can be reduced to 6. This simplification is not always valid in a practical experiment, and is not shown here.

3.6 Quadcopter Control

The hardware and software of the quadcopter has been explained. Knowing which rotor to apply power to is not sufficient to make the drone fly with stabilized control. Without a feedback control system, any quadcopter would be exponentially unstable. The motor gains will not be adjusted using IMU updates, and deviations will increase drastically from equilibrium. The drone will crash shortly after takeoff.

To handle this, the AR. Drone is equipped with an in-built proportional-integral-derivative (PID) controller in its firmware to control its motion and stabilization. The exact structure of this controller is not known, as the onboard software (firmware) is closed source and not documented by the manufacturer [7]. A controller structure based on general quadcopter control is suggested in this chapter.

3.6.1 Attitude Control

Attitude control involves driving the quadcopter to a desired roll, pitch and yaw with a specified angular velocity while maintaining a constant thrust in the body-fixed frame. A simplified attitude control design is presented next.

Controllable inputs are the four rotor speeds. Desired rotor speeds can be written as

$$\begin{aligned} \omega_1^d &= \omega_h + \Delta\omega_F - \Delta\omega_\theta + \Delta\omega_\psi \\ \omega_2^d &= \omega_h + \Delta\omega_F + \Delta\omega_\phi - \Delta\omega_\psi \\ \omega_3^d &= \omega_h + \Delta\omega_F + \Delta\omega_\theta + \Delta\omega_\psi \\ \omega_1^d &= \omega_h + \Delta\omega_F - \Delta\omega_\phi - \Delta\omega_\psi \end{aligned} \tag{3.14}$$

ω_h is the rotor speed required to hover in steady state. ω_F, ω_ϕ, ω_θ and ω_ψ are deviations from hover state in net thrust forces, roll, pitch and yaw respectively.

For attitude control, a PD controller is used:

$$\begin{aligned} \Delta\omega_\phi &= K_{p,\phi}(\phi^d - \phi) + K_{d,\phi}(\dot{\phi}^d - \dot{\phi}) \\ \Delta\omega_\theta &= K_{p,\theta}(\theta^d - \theta) + K_{d,\theta}(\dot{\theta}^d - \dot{\theta}) \\ \Delta\omega_\psi &= K_{p,\psi}(\psi^d - \psi) + K_{d,\psi}(\dot{\psi}^d - \dot{\psi}) \end{aligned} \tag{3.15}$$

where $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$ can be replaced by ω_x, ω_y and ω_z according to the mapping in equation 3.12. Desired rotor speeds are found by substituting equation 3.15 and a $\Delta\omega_F$ into 3.14. Summarized, attitude control computes an angular rate setpoint from the difference between estimated and desired attitude.

Controlling world fixed position is critical for the surface mapping strategy to be disigned. Roll and picth angle rates are implicitly set by setting desired velocities in the implemented program. The onboard attitude controller drives roll and pitch to desired values. The yaw angle will not be changed during a flight, as image analysis (position estimates) are easier to perform with fixed yaw. Tests are performed to decide the performace of the onboard yaw controller. Figure 3.6 presents yaw angles from three complete test flights. Table 3.1 contains of calculated mean yaw and standard deviation for each test.

Table 3.1: Performace of onboard yaw controller, measured in degrees

Test	**Mean Yaw [deg]**	**Max Dev. [deg]**	**Std. Dev. [deg]**
1	-0.38	4	1.39
2	-0.72	5	1.39
3	-0.39	5	1.21
Mean	-0.49	4.67	1.33

Mean yaw angles for all tests are close to the reference yaw (0 *deg*) in world frame. The drone is manually placed in the defined origin of the camera frame at takeoff. Mean yaw deviations can be a result of human error when aligning the drone to coincide with the fixed frame. The onboard yaw controller has, for the

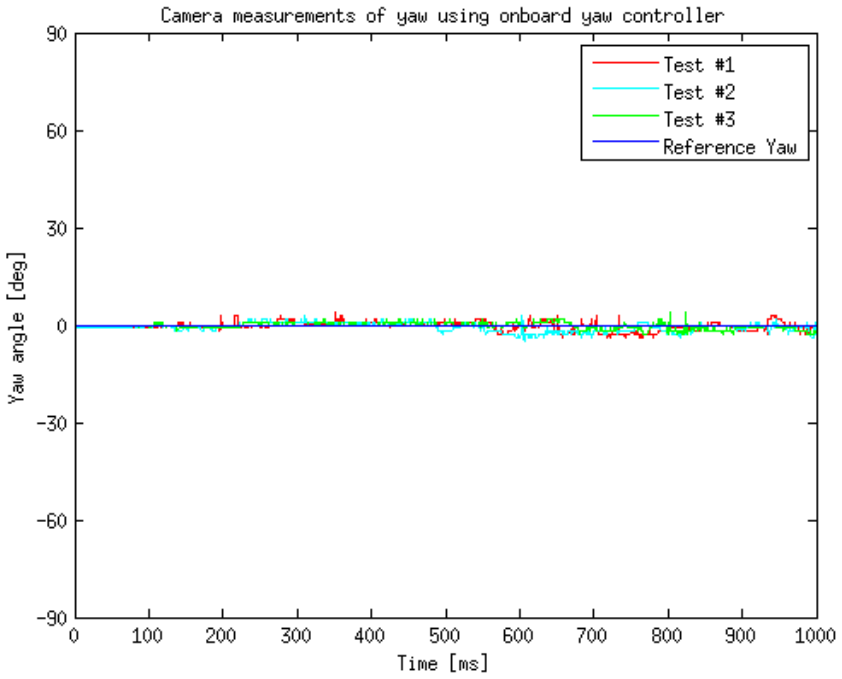

FIGURE 3.6: Measured yaw of tracker mounted on top of drone during three flights using onboard attitude controller.

purpose of this performance test, not feedback from camera yaw measurements. Initial displacements will thus be propagated through the whole flight. Maximum deviation from reference yaw is 5 *deg*, which is likely to occure at turns. Standard deviation is also low considering that OptiTrack angle measurements are defined to be accurate to 1 *deg*. Deviations from yaw reference may also be measured as pitch and roll angles are changed when the drone turns.

Considering test results, the onboard yaw controller is concluded to be able to keep a fixed yaw without significant drift. A seperate yaw controller will thus not be implemented as yaw deviations are accounted for in the positioning controller.

3.6.2 Hover and Altitude Control

Hover control is the process of reaching a desired position and yaw angle with zero linear and angular velocities. Pitch and roll angles are used as inputs to control world (XY) positions while yaw angle is controlled by $\Delta\omega_\psi$ and the altitude is

contolled by $\Delta\omega_F$. The controller structure is motivated by the article *Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors*[10], and is similar to the one implemented in the AR. Drone firmware.

Let $\mathbf{r}_d$ be the trajectory to be tracked. ψ^d is the desired yaw angle. $\ddot{\mathbf{r}}_i^d$ are command accelerations calculated from a PID controller using the position error $e_i = (\mathbf{r}_{i,d} - \mathbf{r}_i)$.

$$\ddot{\mathbf{r}}_i^d = K_{p,i}e_i + K_{i,i}\int e_i dt + K_{d,i}(-\dot{\mathbf{r}}_i) \tag{3.16}$$

Using equation 3.9, desired accelerations (x,y,z) can be written

$$\begin{aligned}
\ddot{\mathbf{r}}_1^d &= g(\theta^d \cos\psi^d + \phi^d \sin\psi^d) \\
\ddot{\mathbf{r}}_2^d &= g(\theta^d \sin\psi^d - \phi^d \cos\psi^d) \\
\ddot{\mathbf{r}}_3^d &= K\omega_h \Delta\omega_F
\end{aligned}$$

For simplicity, K is not specified. Desired angles for roll and pitch are used as input to the attitude controller to get hover control. The equation above is inverted to get

$$\begin{aligned}
\phi^d &= \frac{\ddot{\mathbf{r}}_1^d \sin\psi^d - \ddot{\mathbf{r}}_2^d \cos\psi^d}{g} \\
\theta^d &= \frac{\ddot{\mathbf{r}}_1^d \cos\psi^d + \ddot{\mathbf{r}}_2^d \sin\psi^d}{g} \\
\Delta\omega_F &= \frac{\ddot{\mathbf{r}}_3^d}{K\omega_h}
\end{aligned} \tag{3.17}$$

Altitude control is achieved by changing the z position setpoint to the attitude controller.

To validate if the onboard altitude controller performs accurate control, tests are aquired in the Snake Robot Lab. The quadcopter is run through the generated path in space, using the designed position controller. Position measurements for the height are tracked from the camera system and plotted in a graph. Desired height set in the firmware is measured by the camera setup to be 0.87 m above ground while hovering. Figure 3.7 shows measured height through a defined path. Table 3.2 summarizes mean values and deviations.

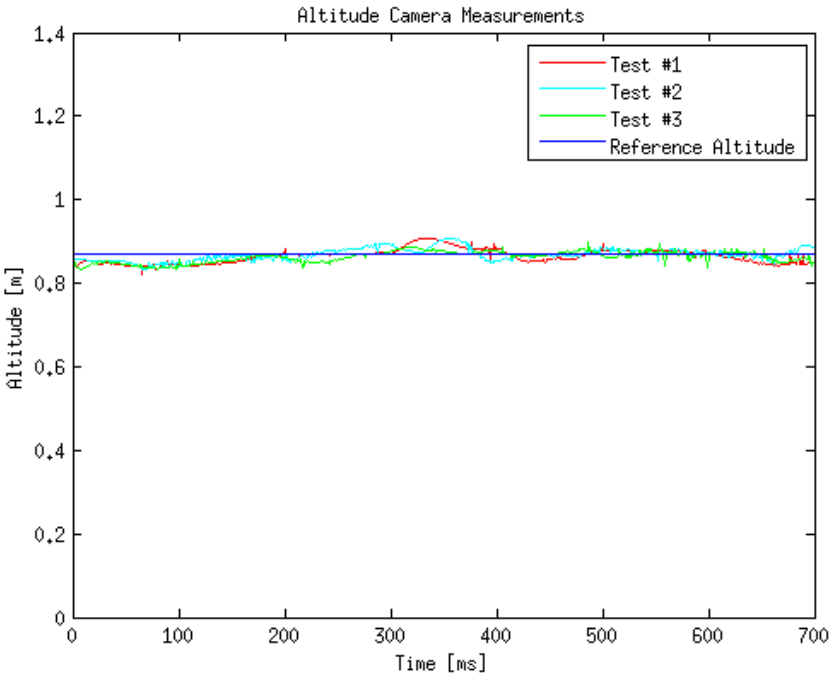

FIGURE 3.7: Measured altitude of tracker mounted on top of drone during three flights using onboard altitude controller.

TABLE 3.2: Performace of onboard altitude controller

Test	Mean Altitude [m]	Max Dev. [m]	Std. Dev. [m]
1	0.866	0.051	0.016
2	0.871	0.047	0.017
3	0.864	0.042	0.013
Mean	0.867	0.047	0.015

Mean altitudes for all tests are close to desired. Standard deviation is low, which means altitude measurements do not deviate much from the mean height. Maximum measured altitude deviation for the three tests is 5.1 *cm*, which is acceptable. Altitude is of interest when computing object positions and areas. As altitude will be fed back as an input to the object detection algorithm, minor deviations from reference altitude is not an issue.

The accuracy of the position measurements from the camera setup may add some measurement noise. When the drone is at rest, constant measurement noise is

about 3 *mm*. Bad calibration or inaccurate trackable initialization may also result in noisy and inaccurate position measurements. A new trackable is therefore defined and the orientation is initialized according to Appendix D. When moving, measurement noise will increase and position estimates at certain spots in the lab room may not have the same accuracy as the initial position due to calibration.

Evaluating mean altitude and standard deviation for flight tests, the in-built altitude controller is concluded to be sufficient for the purpose of testing in this thesis. Testing of the surface mapping strategy depends on the quadcopter altitude to determine the width of the camera field on ground as well as estimation of object sizes. As the altitude is used as input to these estimates, accurately fixed height is not required for the strategy to be accurate. If the controller was required to be more accurate, a PI controller with altitude feedback measurements from the camera setup can easily be implemented.

3.6.3 Overall Quadcopter Control Summarized

The controller architecture is devided into multiple layers. Each layer controlling different states described in previous sections. Inner layers control the stabilization of the drone, i.e. angular velocity rates and Euler angles roll, pitch and yaw represented in a $\boldsymbol{\Theta}$ vector. Angular rates are given in a $\boldsymbol{\Omega}$ vector. The control system has a cascade structure with feedback loops and integrators. A simple P controller can be used to control the motors. In this thesis, the focus will be controlling position by applying velocities in given directions.

Chapter 4

Methods and Theory for Surface Mapping and Indoor Positioning

An indoor positioning system is crusial for the designed surface mapping strategy. Measurements are used as inputs for position control of the vehicle. Background theory on methods and tools used for indoor positioning and surface mapping is presented so that it will be easier to understand the system design and implementation. A method for camera based indoor positioning is briefly introduced. Methods and theory used to perform surface mapping is the main focus of this chapter. Computer vision techniques are presented so that the reader can understand main concepts and get familiar with chosen methods and approaches.

4.1 Method for Camera-based Indoor Positioning

A positioning system that can estimate position, orientation and velocity of the quadcopter is utilized in the overall surface mapping strategy. For outdoor navigation, a Global Positioning System (GPS) is widely used, often in combination with inertial navigation. For quadcopter surface mapping experiments in the Arctic, such integrated system is likely to be a good choise. When doing experiments

on a small scale indoors, GPS is neither accurate nor available. A local positioning system is needed.

One among several alternative positioning systems is camera-based positioning. A camera system setup is combined with image processing and computer vision techniques to estimate navigation data like position, orientation and velocity for an object. The camera system can be calibrated to recognize a certain shape or object. Reflective markers with a given geometry attached to the vehicle of interest defines the trackable object. Using multiple cameras, an algorithm can be applied for converting 2D image positions into 3D position and orientation estimates. A camera-based indoor positioning system consisting of 16 cameras and software that computes position and orientation in the world frame will be interfaced for use with the quadcopter positioning controller.

4.1.1 Techniques for Marker Detection and Navigation

For a camera based navigation system to be able to track a moving robot, some unique identifier must be defined. A predefined gemotrical shape with reflective markers are attached to the drone. Because of its geometry, both position and orientation can be measured. Details on algorithms used to obtain these estimates are not revealed from the manufacturer. A marker detection algorithm is assumed to run in each camera, while the *Tracking Tools* software converts marker positions to world positions of the trackable. Triangulation is used to obtain the world position of a point of interest. The used marker is given in Figure 4.1.

4.1.2 The Camera-Based Positioning System OptiTrack

The position controller designed and implemented in this thesis relies on position and orientation measurements from a camera-based positioning system. The Sintef Snake Robot Lab is equipped with OptiTrack from NaturalPoint. NaturalPoint systems provide optical tracking of markers using one or several software releases

that interact with the cameras. The equipment from OptiTrack makes it possible to read real time tracking data like position and orientation in six degrees of freedom to be used with the Parrot AR. Drone 2.0.

16 *Flex 13* cameras are placed in the ceiling with the aim of maximizing the capture volume, i.e. the space in the room where the marker can bee seen from at least two cameras. OptiTrack software running on a local computer is used to estimate navigation data to be streamed to a remote computer controlling the Parrot AR. Drone.

FIGURE 4.1: OptiTrack marker for tracking

FIGURE 4.2: Quadcopter and camera system

The accuracy of the OptiTrack system depends on calibration and initialization. Vintervold[3] concluded that a definite measure of the accuracy is difficult to find due to severeal error sources. However, measured accuracy was less than 1 *cm* in both X and Y directions per 1 m. The promised accuracy from OptiTrack is in the sub-millimeter range for individual marker positions, and the calculation of the trackable position from marker information is not available. Thus, it is difficult to inspect. Measurement noise level is measured at the lab to be about 3 *mm*.

4.1.3 The Marker Tracking Software Tracking Tools

Tracking Tools is one of several OptiTrack programs. The cameras capture images inside the capture volume while obtaining location and size of detected markers.

Next, the information is sent to Tracking Tools, where position and orientation estimates for the trackable marker is calculated.

Before accurate tracking is possible, the system must be initialized. Initialization includes calibration, defining the ground plane, initialization of orientation and creation of a rigid body. This process is described in Appendix D.

4.2 Surface Mapping

Locating objects such as icebergs includes covering a desired area and storing the information in a map. In this specific case, surface mapping includes making the drone fly over an area of interest using its onboard camera to take pictures of the ocean. These pictures will be processed on a remote computer, and navigation data will be used to place detected ice in a map displaying the whole area covered. Using this real time map, ships can plan routes avoiding the ice. Satellite images can be combined with mapped data from the drone to consider larger areas.

4.2.1 Search Algorithm

A search algorithm is an algorithm that looks for a specific object among many based on various parameters. Search parameters are set according to the object to be detected. There are numerous different search algorithms used in computer science. A robot, in this case a drone, will be used to perform the search. This means that the predefined movement pattern of the drone will define the search pattern.

When searching for floating ice or icebergs, one can make certain assumptions and measurements. For the following example, a ship is fixed at one position using DP or is moving in a fixed direction. The following can be assumed known and be used in the complete model:

- Direction of current.
- If the ship is moving, direction of motion is known.

Ice will float with sea current. These data can be considered so that a minimized area is covered with the search algoritm. There is noe need to search for icebergs that will not be in danger of colliding with the ship within a certain time period. Details on the proposed search algorithm will be presented in Chapter 5. For now the search pattern will be restricted to a given direction and width to cover an area of interest found from above assumptions. In this thesis, a model for ice movements is not designed and will not be considered in the overall map.

Depending on the object in question, search parameters might be color, shape or size. In some cases, there will be need of matching the object with one or many reference images. In this thesis, search parameters will be color and size. This provide simplicity of the problem and may not be sufficient in an outside practical experiment (discussed in Chapter 8).

4.2.2 Image Recognition and Analysis

Image recognition will be used to detect objects at the ground. To do the implementation with images from the onboard camera, *Open Source Computer Vision* (OpenCV) will be used. This is a C++ library of programming functions mainly aimed at real-time computer vision. It is a very powerful tool containing of more than 500 algorithms. Chapters 5 and 6 will show how image recognition and analysis are carefully interfaced with GNC of the drone.

4.3 Computer Vision Methods for Object Detection

A camera onboard the quadcopter captures live images to be processed and analyzed. Image processing and analysis are performed with various computer vision

techniques. Existing algorithms in OpenCV will be applied in an overall object detection algorithm. Theory on most interesting algorithms are briefly given in this section[1].

4.3.1 Thresholding an Image

Thresholding is a segmentation method. The concept is used to separate out regions of an image corresponding to objects to be analyzed. This separation is based on the variation of intensity between background and object pixels. Each pixel intensity is compared to a threshold value or range. When pixels inside the range is separated, they are identified by a boolean value where each boolean corresponds to a color value to be shown in an image (i.e. 0 for black and 255 for white).

Binary threshold makes sure all pixels inside a range is set to one value while all other pixles are set to another value

$$dst(x,y) = \begin{cases} MaxVal, & \text{if } ThreshMin < src(x,y) < ThreshMax. \\ 0, & \text{otherwise.} \end{cases}$$

This means that if the intensity of a pixel $src(x,y)$ is within a *thresh* interval, then the new pixel intesity is set to a defined $MaxVal$. Otherwise, the pixel is set to 0. The image is now binary, with only two pixel intensities. It is easy to do several operations and analysis on the thresholded image. Section 6.7.1 shows how thresholding is implemented.

4.3.2 Finding Contours in an Image

A method for separating multiple objects in a binary image is to find the *contours* for each object. A contour is a boundary of an object, a population of pixels

[1]Theory is based on the offisial OpenCV documentation at http://docs.opencv.org/

separating an object from the background.

Every contour has a fixed starting point. All points in a contour consists of complex numbers. From the starting point of the countour, each point found are defined as complex vectors $a + ib$. a is the point offset on the X axis, while b is the offset on the Y axis, both compared to the previous point. The last vector of a contour leads to the starting point. Each vector is called *elementary vector*. The final vector contour Γ containing of elementary vectors γ is defined as

$$\Gamma = (\gamma_0, \gamma_1, ..., \gamma_{k-1})$$

4.3.3 Morphology Transformations for Noise Filtering

Mathematical morphology is a technique for analyzing and processing geometrical structures. Thresholding often result in noisy images, especially as the drone camera is moving and light reflections may be a source of noise. Binary morphological techniques called *erosion* and *dilation* can be used both for this purpose and for clear isolation of individual objects (floating ice, icebergs, ice floes). Separation of objects is important in order to find areas as well as concluding whether or not the ice pose a threat to the ship. The following theory is based on *Digital and Medical Image Processing*[1].

4.3.3.1 Erosion

Binary erosion is a process that shrinks segmented forground contours in an image. It computes a local minimum over the area of a kernel. A *structural element* S is defined as a box of a number of pixels, typically small relative to the image. This element is passed to the image to be compared with a pixel and its neighbours. If the pixel and its neighbours match the structural element, it is set to 1. If it does not match, the pixel is eroded (set to 0). Erosion $\varepsilon(X)$ of a set X by a structuring

element S is defined as

$$\varepsilon(X) = \{x \mid \forall s \in S, x + s \in X\}$$

Basically, by placing the structuring element anywhere in the image and checking if it is fully contained by a subset of the pixels, then the origin of the structuring element is part of the eroded set. The concept is illustrated in equation 4.1. Figure 4.3 and 4.4 is another example with the set X and the structuring element S. The resulting erosion $\epsilon(X)$ is given to the right.

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix} \ominus \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \rightarrow \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix} \tag{4.1}$$

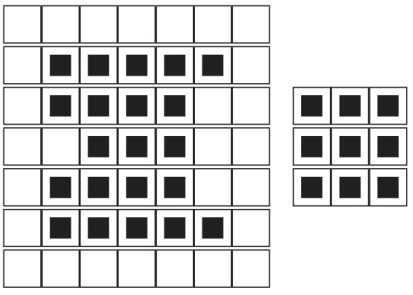

FIGURE 4.3: Set of X and structuring element S [1]

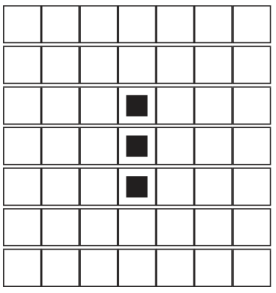

FIGURE 4.4: Erosion [1]

4.3.3.2 Dilation

Dilation is the opposite of erosion. It expands segmented areas by performing an union operation with the structural element. The dilation $\delta(X)$ of a set X by a structuring element S is defined by

$$\delta(X) = \{x + s \mid x \in X \wedge s \in S\}$$

Dilation is illustrated in equation 4.2

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix} \oplus \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \rightarrow \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix} \tag{4.2}$$

For the purpose of noise reduction, erosion is performed to remove small areas. Next, dilation is performed to restore possible information loss in the image. These operations can be run multiple times for better results.

4.3.4 Image Moments for Position and Area Calculations

An image moment is a weighted average (moment) of the image pixels' intensities, or a function of such moments, chosen to have some property or interpretation. Generally, a moment can be said to be a measure of the shape of a set of points. Calculating the area or the center of mass of an object in an image can be done by utilizing moment calculations. The object detected by finding its contours can have any shape. Moments up to third order of a polygon[2] can be calculated. Finding positions of objects is critical for the surface mapping strategy. The mass center is found by calculating spatial moments. Most interesting image moment equations are given in this section.

For a 2D continuos function $f(x, y)$, the geometric moments of order $(p + q)$ are defined as

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy$$

[2]A polygon is defined as a plane shape bounded by a finite number of straight line segments, called edges or sides.

A greyscale image with pixel intensities $I(x, y)$, spatial image moments M_{ij} are calculated by

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

where the mass center is given by

$$x = \frac{m_{10}}{m_{00}}, \quad y = \frac{m_{01}}{m_{00}} \tag{4.3}$$

and the area is given by m_{00}.

Chapter 5

System Design for Object Mapping using UAV

This chapter presents the design of the surface mapping strategy. Unified Modeling Language (UML) will be used to illustrate the communication flow and how the system operates. Path tracking is designed using a PID controller that generates reference velocities to control the drone position. Main modules are autonomous flight and surface mapping using computer vision, included network streaming of navigation data and guidance of the quadcopter.

5.1 System Design and Architecture

Commands for maneuvering are sent from a remote computer, while navigation data is sent from the drone to the computer. A second compter is used to retrieve navigation data from the camera-based indoor positioning system. These navigation data are streamed to the remote computer controlling the drone. Diagrams showing how software and hardware interacts, user interaction and states during a complete operation are worked out.

5.1.1 Overall System Architecture

An overall system architecture is given in Figure 5.1. Main modules are given as guidance, navigation, control, object detection and the physical quadcopter. The diagram is not designed to go into details on sub modules, which will be given in later sections.

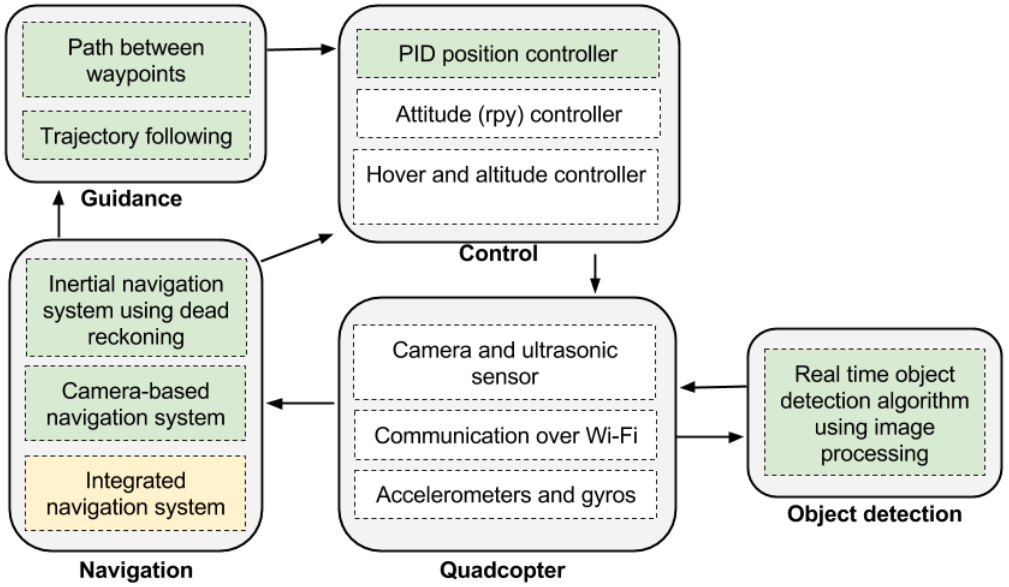

FIGURE 5.1: Overall system design architecture

Green modules are implemented in this overall system. The yellow module is proposed and discussed, but not implemented. White modules are part of the quadcopter or its firmware/software.

5.1.2 State Diagram

The state diagram below presents all possible states for the drone during a complete operation, and how the drone gets from one state to another. Reading from this diagram one can get an idea how the system design and implementation is done.

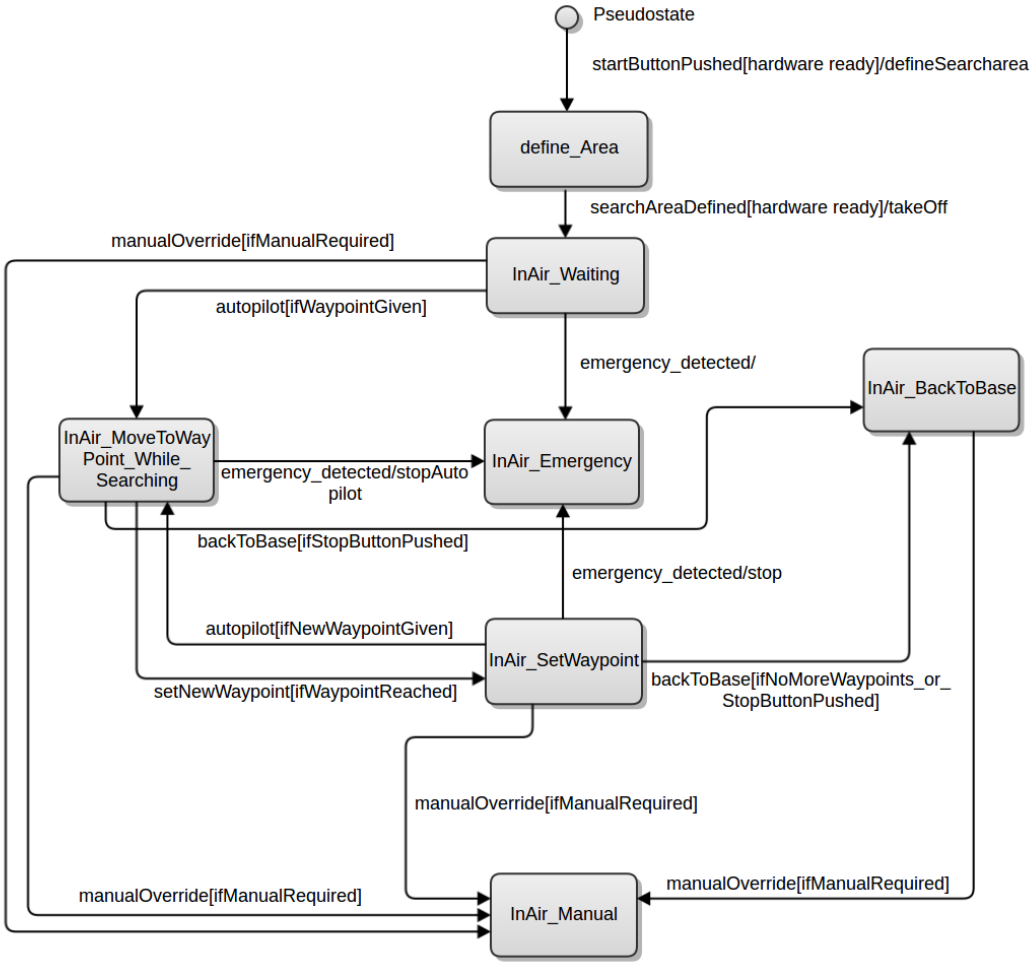

FIGURE 5.2: State diagram illustrating the possible states for the complete system

Mark that some states are not included. *takeOff* and *land* are not given due to the readability of the diagram. The arrows illustrate which states are leading to new states. The state *inAir_Emergency* is for safety reasons. All propellers shut down when the drone enters the state. An emergency is detected by the onboard sensors, and can either be an obstruction at the propellers or too large changes in roll, pitch or yaw for the drone to be able to stabilize itself.

An operator should be able to manually override the position control of the drone at all times. In cases where unwanted commands are sent from the remote computer to the drone, it might be necessary to intervene with the autonomous flight. This functionality is not implemented in the test setup, but manual control is successfully tested with a gamepad as well as the computer keyboard. The state

define_Area illustrates the program prompting the operator for desired search distances in north and east directions. In the lab setup, these distances will be given in meters.

Functionality for making the drone fly autonomously back to the initial position is not implemented for the case where the initial position (ship) is moving. If not moving, the initial position can be set as the last waypoint. A positioning system for the ship/platform is not designed. For simplicity in the lab setup, the last waypoint given as input to the controller can be the initial position given as $x = 0, y = 0$. Programming the drone to land autonomously on a moving platform is a fairly challenging problem by itself, and will not be part of this work. Vision-based quadcopter control strategies could have been used for the purpose.

5.2 Drone Position

The position of the drone can be measured or estimated in several ways. Using a Global Positioning System (GPS) tracker mounted to the drone could be one approach, preferably combined with other methods for handling short term stability issues with GPS. As the lab setup is in a small area inside a building, using GPS is not an alternative. The project work[11] written as the starting phase of this thesis uses measurements from the drone IMU to compute positions via dead reckoning. The method was concluded to drift, and not fitted for long term operations. A design for using the camera-based navigation system will be given and later compared with the dead reckoning system as well as an outdoor approach.

5.2.1 Camera-based Indoor Navigation System

Section 4.1 focuses on the methods and specifications of tools used to retrieve tracking data form the cameras mounted in the ceiling. This section will show

how the camera-based system is interfaced with the program controlling the quadcopter. Details on the implementation are given in section 6.5.

OptiTrack software can only be used on Windows computers. Interaction with Parrot AR. Drone is done from a Linux computer, which means navdata estimates must be streamed from the Windows computer to the remote Linux computer. Several NaturalPoint protocols for data streaming do exist, but cross platform streaming (Windows to Linux) is not supported by default. Work was put into adapting the existing streaming protocol *NatNet* to be able to stream data from a Windows computer to Linux. Due to implementation problems, the chosen approach was to stream data using LabView.

An existing program written for the Sintef Snake Robot Lab was used as a starting point to send UDP packages containing of position and orientation estimates. The Windows computer loads a TrackingTools project file and initiates the communication between the OptiTrack camera system and the LabView program. In LabView, tracking data are interpreted, filtered and decoded into a bit format sent over UDP to the remote Linux computer that controls the quadcopter. From the Linux computer, UDP packets are read in an UPD listener program and used as inputs to the designed path tracking controller.

5.2.2 Dead Reckoning

A dead reckoning system is a system that measures the change in position, velocity or acceleration over time. An INS is a three-dimensional dead reckoning system. To obtain the current position, the measured position is added to the previous position. The concept of an INS is described in Section 3.2.4.

The measured velocity found from the IMU together with Δt for one loop through the program is used to define a matrix $\mathbf{M}$ containing local movements

$$\boldsymbol{M} = \begin{bmatrix} v_x \Delta t \\ v_y \Delta t \\ v_z \Delta t \end{bmatrix} \tag{5.1}$$

From equation 3.7, the dead reckoning position is given as

$$\mathbf{r}^w = \mathbf{r}^w + \mathbf{R}_w^b \mathbf{M} \tag{5.2}$$

This position estmate may drift as the position is not updated by another navigation system during an operation.

5.2.3 Camera Setup/INS Integration

Generally, an inertial navigation system with dead reckoning alone is not updated by another navigation system, and will thus drift with time. Navigation system with external updates such as GPS or camera based positioning systems are needed to avoid drift. For most purposes using GPS it is preferable to integrate an inertial navigation system with GPS measurements. Chapter 8 will argue why such integrated navigation system is not implemented. However, a proposed filter is briefly presented as it will be of interest if the update source was to be GPS. Such filter could of course be implemented with OptiTrack as well, but Vintervold [3]concluded that accuracy was not a noticeable improvement for the integrated navigation system compared to OptiTrack measurments alone. Throughout this section, it is assumed that the positioning system is not as accurate as OptiTrack, or simply that GPS is to be used.

Using position estimates from the camera setup/GPS, one can create an integrated navigation system without drift. Measurements from the cameras/GPS can be sendt to an integrating filter with the INS measurements. An integrated filter

will combine the advantages of high output rate and good short-term accuracy of INS with the long-term accuracy of the camera system/GPS. Redundancy is also achieved.

Several types of integration filters exist. Among them are *uncoupled, loosely, tightly* and *deeply coupled* integration. As tight coupling increases, the performance and robustness against interference is increased. The cost is increased complexity, lack of redundance and reduced flexibility.

INS positions and velocity measurements are subtrackted from position and velocity measurements from the camera setup to form an error signal that is used as a measurement by the integration filter. A Kalman filter is a suitable state observer for the purpose.

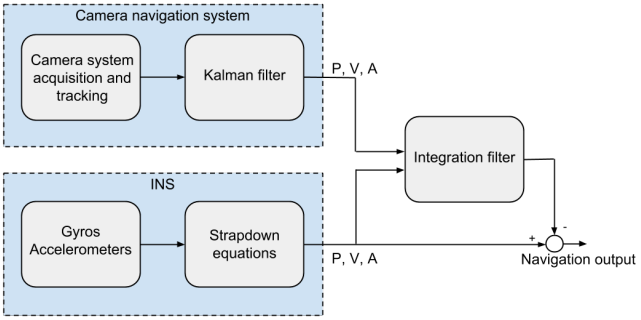

FIGURE 5.3: Uncoupled integrated system.

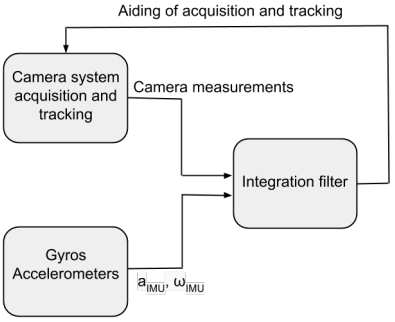

FIGURE 5.4: Tightly coupled integration.

Two proposed filters are illustrated in Figures 5.3 and 5.4. P, V and A refer to position, velocity and attitude respectively. For the tightly coupled system, feedback exist. Raw accelerometer and gyro measurements from the sensors are used instead of position, velocity and attitude data. Feedback (reset) to provide real time calibration of the INS using error measurements can be used. Outputs are position, velocity and orientation measurements to be used as inputs to a position controller.

5.3 Path Tracking Control

Autonomous flight control for UAVs has been researched for some time. Several projects have successfully simulated trajectory tracking control solved as an optimalization problem[12]. Not many of these projects actually implement the trajectory tracking algorithm on an UAV. This section will present how path tracking control has been implemented on the Parrot AR. Drone.

An UAV is initially unstable and needs a separate stabilization controller in order to be able to stay in air. Even though this is a very important and interesting subject, details will not be implemented as the AR. Drone is already stabilized from the manufacturer. As the drone is also equipped with a hover controller that makes the drone keep a fixed height, only movements in a 2D plane will be considered in the system design. For testing in the lab setup, only roll and pitch will be controlled by controlling velocities in X and Y directions respectively. Thus, the drone position in the world (NED) frame will be controlled. Details on the dynamic model and embedded controllers for the drone are given in Section 3.5.3.

The position and velocity of the drone will be controlled by a path following controller. The path is defined by a sequence of N desired waypoints $\mathbf{wp}_d^w$ and desired velocities of travel v_d, i^w in the world (NED) frame. Path segments between each waypoints has its related unit tangent vector $\mathbf{t}_i$ in the desired direction of motion on the path from waypoint i to $i+1$ and unit normal vector $\mathbf{n}_i$. Figure 5.5 illustrates the relations. The implemented controller is based on the controller design in [4] by Hoffmann et al.

Along track in each step of the path is defined to be in the direction of the $\mathbf{t}_i$ vector while *cross track* is the direction of the $\mathbf{n}_i$ vector. Given the current position measurement $\mathbf{r}^w$ and estimated velocity $\mathbf{v}^w$, the cross track error, cross track error

rate and along track error rate can be found from equation 5.3

$$\begin{aligned} e_{ct} &= (\mathbf{wp}_d^w - \mathbf{r}^w(t))\mathbf{n}_i \\ \dot{e}_{ct} &= -\mathbf{v}^w(t)\mathbf{n}_i \\ \dot{e}_{at} &= v_{d,i} - \mathbf{v}^w(t)\mathbf{t}_i \end{aligned} \tag{5.3}$$

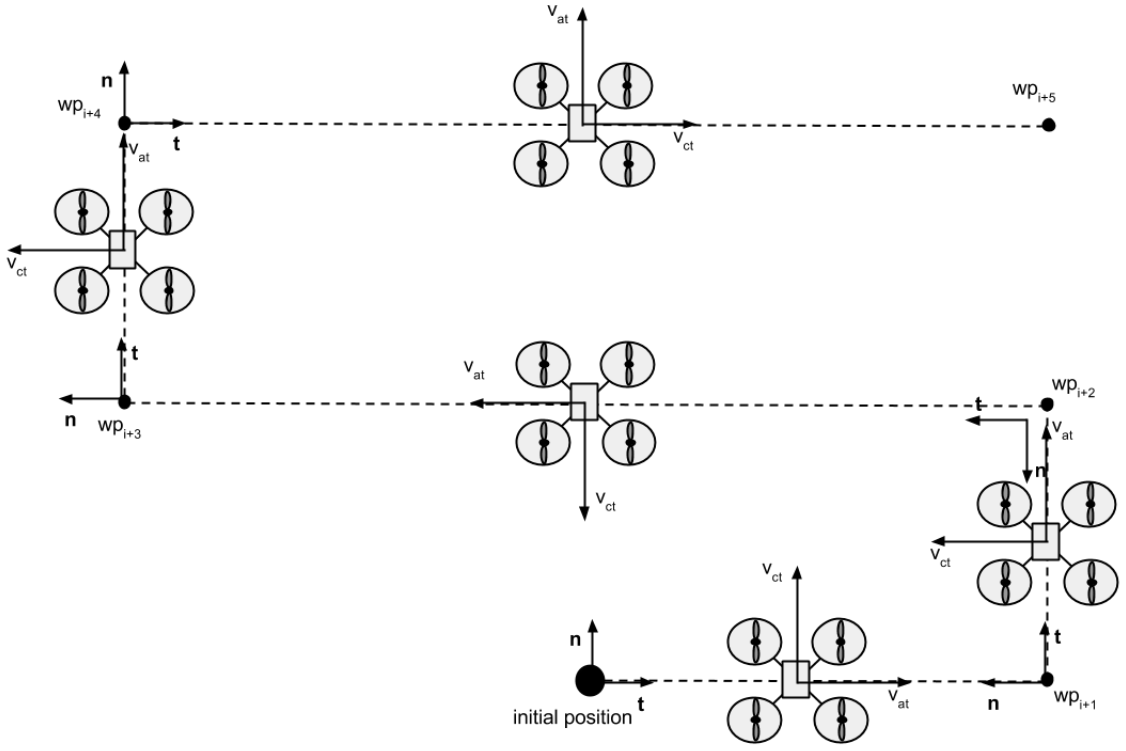

FIGURE 5.5: Cross track and along track illustrated for different drone positions. **n** and **t** vectors defines orientation and direction of motion.

Along the track of motion, only the error rate is considered, and depends only on the velocity of the drone. This means the controller does not attempt to catch up or slow down for waypoints, but proceeds along the track matching the desired velocity as closely as possible. Thus, image processing may be easier to perform. A trajectory tracking controller is designed by closing the loop on along track velocity and cross track error. The resulting controller is a PI controller in the along track direction and PID controller in the cross track direction. The state to be controlled is the velocity set in the body frame related to a desired velocity in

the world frame.

$$\begin{aligned} v_{at} &= K_{p,at}\dot{e}_{at} + K_{i,at}\int_0^t e_{at}dt \\ v_{ct} &= K_{p,ct}e_{ct} + K_{d,ct}\dot{e}_{ct} + K_{i,ct}\int_0^t e_{ct}dt \end{aligned} \tag{5.4}$$

Transitions for setting the next waypoint occurs when the drone position is within a circle of i.e. $10\,cm$ away from the current waypoint. At this transition, **n** and **t** vectors are reset in order to redefine the desired trajectories and along track/cross track directions.

5.3.1 Mapping from Desired World Frame to Body Frame

Waypoints in the world frame defines the desired path to follow. The drone orientation does not necessarily coincide with the orientation of the fixed frame, due to yaw misalignments. As desired velocities are defined in the fixed frame, the mapping in equation 5.5 can be done to convert desired velocities to body fixed gains.

$$\begin{bmatrix} v_x^b \\ v_y^b \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} v_x^w \\ v_y^w \end{bmatrix} \tag{5.5}$$

Body fixed velocities are applied by drone thrusters. ψ is the yaw angle relative to a predefined zero orientation measured by the camera setup.

5.3.2 Controller Tuning

Tuning a controller is the adjustment of its control parameters to achieve a desired control response. Stability is a basic requirement, but further requirements are defined by the system designer.

For the indoor lab setup, the controlled velocity is desired to be relatively low. This is also an advantage when doing real-time image analysis. As the AR. Drone has a built-in stabilization controller, fast velocity changes are neither necessary nor

wanted. As described in Section 5.3, desired velocities are inputs to the controller in the along track direction. One goal is to minimize the difference between actual and desired velocities. This is done by using PI and PID controllers in along track and cross track directions respectively. The most important goal is to tune the controller to be stable and follow a desired feasible trajectory. Equation 5.4 presents all parameters to be tuned.

Several tuning methods can be used. Manual tuning might be sufficient for many purposes. Proven methods like Ziegler-Nichols or Skagestads method will often result in more optimal tuning. As the implemented controller does not need to be fast and stability is not a major issue due to onboard stabilization, tuning by trial and error will be done in chapter 7.2.1.

5.3.3 UAV Waypoint Guidance

As stated, cartesian coordinate waypoints are reference inputs to the trajectory tracking controller. The algorithm makes the drone fly the shortest feasible path between waypoints, containing of straight lines and circular arcs. Controller tuning defines how quickly to adapt to the trajectory and the path smoothness.

Section 4.2.1 introduced the waypoint generation. For this strategy to be adaptive to different search areas defined by an operator, the waypoints will be generated as a function of several parameters defined in table 5.1. Waypoints are stored in a variable and used for generation of a trajectory or a path for the quadcopter to follow. Several criterias are neglected when generating waypoints. Environmental data, geographical data and possible obstacles are not part of the guidance strategy.

Table 5.1: Variables used for waypoint generation

Variable	Description
x	Search distance in X direction
y	Search distance in Y direction
n_{wp}	Number of calculated waypoints
θ, β	Bottom camera angles
h	Height above ground
d, w	Depth and width of camera search field

where the number of waypoints are calculated from (integer division)

$$n_{wp\_north} = \frac{x}{d} + 1 \tag{5.6}$$

$$n_{wp} = 2n_{wp\_north} - 1 \tag{5.7}$$

and the depth of the camera search field (d) is given by

$$d = 2h\frac{\tan\beta}{2} \tag{5.8}$$

Figure 5.6 illustrates this relation.

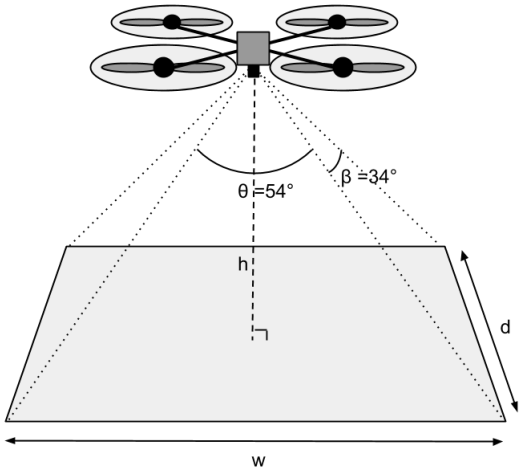

Figure 5.6: The camera mounted below the drone IMU can see an area on the ground defined by the camera angle and the height above the ground.

Using a simple search pattern, an overall motion path and camera FOV is given in figure 5.7.

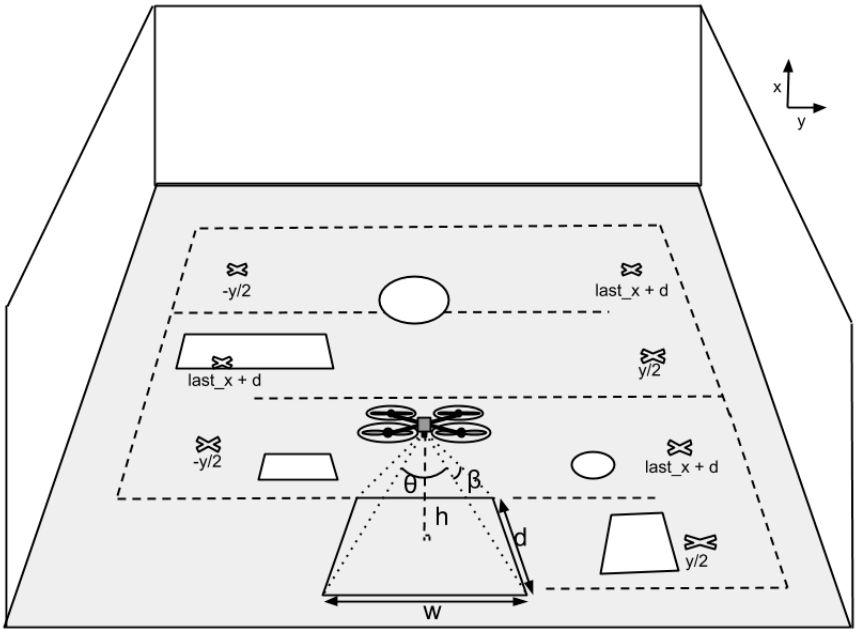

FIGURE 5.7: Drone used for surface mapping. Objects on the ground will be spotted by the camera, and position and size will be estimated.

Waypoints are marked as "x" in Figure 5.7. One can see that the first waypoint is set to be $\frac{y}{2}$. Next, if the waypoint number is even, the waypoint in the X direction is given as $\mathbf{wp}^{w}_{x,next} = \mathbf{wp}^{w}_{x,before} + d$, while the waypoints in Y and z directions are unchanged. When the waypoint number is odd, the waypoints are unchanged in X and Z direction, while $\mathbf{wp}^{w}_{y,next} = \pm\frac{y}{2}$. With these waypoints in the y direction, the actual distance covered by the camera will be $\frac{y}{2} + \frac{w}{2}$. This overlap may compensate for positioning inaccuracies. Overlap in the X direction can also be set in a similar manner to avoid possible blind spots due to deviations from desired trajectory. With other words, every second waypoint is a change in X direction, while every other second waypoint is a change in Y direction to obtain the search pattern in the figure above. Only changes in waypoints are displayed in the figure. When the last defined waypoint is reached, the drone moves back to the initial position and lands.

5.4 Object Position

Positions of detected objects (and areas) are of great interest. The object detected by the search algorithm is not at the same (X, Y) position as the quadcopter. Two approaches for finding the object position are implemented. Method number one neglects orientation changes, while the second method computes the back-projection of a 2D pixel point to world coordinates. Several assumptions are stated for both approaches.

- Objects are assumed to lie in a 2D plane with zero height.
- The 2D plane (ocean) coincides with the defined world coordinates for X and Y.
- Possible timing errors between image frames and position/attitude information are neglected.
- The camera position is assumed to coincide with the measured position of the drone.

5.4.1 Neglecting Orientation to Find Object Positions

The first method is further simplified and makes more assumptions.

- Assume that the camera is an ideal digital frame type with an ideal perspective type lens. Therefore, camera and lens distortions are neglected.
- Orientation of the camera is assumed to be fixed, i.e. no tilt (roll or pitch) for the camera. Possible yaw deviations are fed back as a rotation about the z axis to the position estimates.

Assuming a fixed camera coordinate system adds a source of error, and will affect accuracy of results. This neglection is made as the drone operates at low velocities,

and thus has small angle changes. In addition, search areas are defined so that waypoint turns are made in a region outside the area of critical interest. A method that includes roll and pitch measurements in position estimates are proposed in Section 5.4.2. The iceberg position can be calculated as follows

$$\mathbf{r}_o^w = \mathbf{r}_d^w + \mathbf{r}_o^i \tag{5.9}$$

where $\mathbf{r}_o^i = \begin{bmatrix} r_x^i, r_y^i \end{bmatrix}^T$ is the object position in the image frame in meters. The height (h) will be retrieved from OptiTrack measurements. Calculating the object position in the (XY) plane can be done by using that one point in the image is defined as the origin and measuring the distance covered in one image based on the height from the sea as well as the camera frame angle. That is, converting image pixels to width in meters using equation 5.8. The vertical camera resolution of $w_{pix} = 640$, $d_{pix} = 360$ pixels gives

$$\begin{aligned} w &= 2h\frac{\tan\theta}{2}[m] \Rightarrow 640[pix], \quad \theta = 54\,\text{deg} \\ d &= 2h\frac{\tan\beta}{2}[m] \Rightarrow 360[pix], \quad \beta = 34\,\text{deg} \end{aligned} \tag{5.10}$$

The center of the image captured by the drone is defined as the pixel position $x = 0, y = 0$, as shown in Figure 5.8.

By defining

$$\mathbf{r}_o^i = \begin{bmatrix} r_x^{pix} \\ r_y^{pix} \end{bmatrix}$$

as the pixel position of the centre of the object in the image frame, the position can be found in meters from

$$r_x^i[m] = \frac{\mathbf{r}_x^{pix}[pix]}{360[pix]}d, \quad r_y^i[m] = \frac{\mathbf{r}_y^{pix}[pix]}{640[pix]}w$$

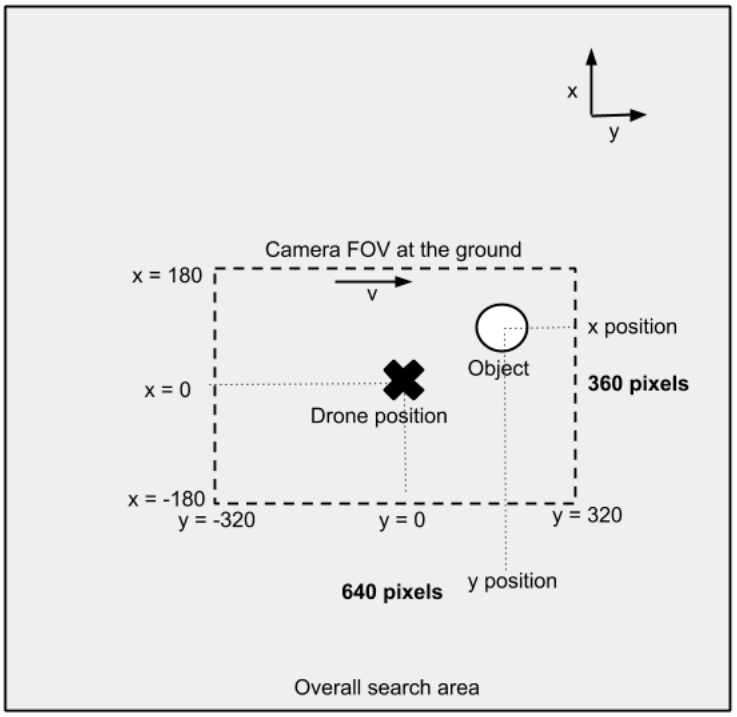

FIGURE 5.8: The image from the drone is defined in pixels. Using that the center of the detected object is a given pixel position in the frame the pixel position can be converted to meters as above.

where the x and y direction is defined as in Figure 5.8. Inserting these positions in the vector $\mathbf{r}_o^i$ results in the total object position $\mathbf{r}_o^w$ in equation 5.9. This approach does not consider the camera orientation. Next section will describe an algorithm that does just that.

5.4.2 Back-projection of a 2D Pixel Point to World Coordinates

Projections between dimentions in images and the real world is a major field of study in computer vision. Projecting a 3D point onto a 2D image is necessary functionality for image processing, but will not be described in this thesis. The inverse operation is of great interest when mapping objects in world coordinates with a camera attached to a drone. While manouvering, the drone will normally have changes in roll, pitch and yaw. Object positions detected by the image processing algorithm will have to be mapped to 3D world coordinates to provide

accurate results. Upcoming theory is motivated by the book *Multiple View Geometry in Computer Vision* (ch. 8) [13] and the Ph.D. Thesis *Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video* (ch. 2) [14].

Chapters 7 and 8 will indicate why such mapping is useful, as well as discussing assumptions made in the surface mapping strategy. Section 5.4 introduces assumptions and simplifications.

Finding the relation between a 2D pixel position and the position on the ground in meters is a mixed problem. Among factors that are involved are rotations between frames, camera position and a *camera calibration matrix* $\mathbf{K}$ (3×3) on the form of equation 5.11.

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.11}$$

Where f is the focal length and $p_x,\ p_y$ are pixel coordinates of the translation to the center of the image. This matrix is individual for each camera, and is found by calibration. The values for this matrix for the AR. Drone 2.0 vertical camera are written in Appendix B.

In order to find the back projection from a 2D point to world coordinates, the projection of a world point onto a 2D image plane is first found. The pixel position of an object $\mathbf{r}_o^i = (x, y, 1)^\intercal$ of a world coordinate point $\mathbf{r}_o^w$ is found by translating the camera position $\mathbf{C} = \mathbf{r}_d^w$ to the world coordinate origin before rotating. Unlike the defined pixel frame in figure 5.8, pixels are now counted from the top left corner like the defined image frame in Figure 3.5. Using homogeneous coordinates for the object position $r_o^w = (x, y, z, 1)^\intercal$, an expression for the pixel can be found position before inverting it.

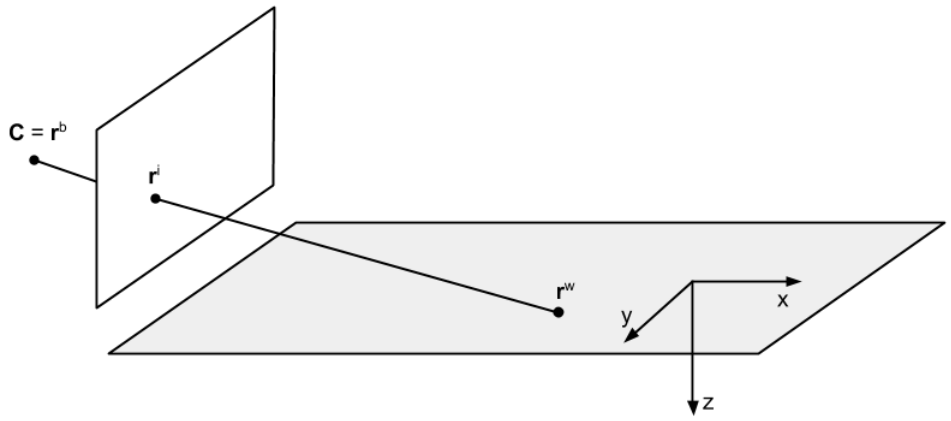

FIGURE 5.9: Illustration of perspective image of points on a plane. The world XY-plane is aligned with the plane in the figure. Points in the image are related to points on the plane through projective transformation [13].

An expression relating the camera matrix $\mathbf{K}$ to the object position in the camera frame and the image frame can be derived as

$$\mathbf{r}_o^i = \mathbf{K}\left[\mathbf{I} \mid \mathbf{0}_3\right]\mathbf{r}_o^c \tag{5.12}$$

The camera frame and world frame are related via translation and rotation. This is presented in the following expression:

$$\begin{aligned}
\mathbf{r}_o^c &= \mathbf{R}_c^w(\mathbf{r}_o^w - \mathbf{C}^w) \\
&= \begin{bmatrix} \mathbf{R}_c^w & -\mathbf{R}_c^w\mathbf{C}_w \\ \mathbf{0}_3^\intercal & 1 \end{bmatrix} \begin{bmatrix} x^w \\ y^w \\ z^w \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{R}_c^w & -\mathbf{R}_c^w\mathbf{C}_w \\ \mathbf{0}_3^\intercal & 1 \end{bmatrix} \mathbf{r}_o^w
\end{aligned}$$

where $\mathbf{R}_c^w$ contains of rotations between the world frame and camera frame, included a rotation of $\frac{\pi}{2}$ about the Z axis according to Figure 3.5. The second line gives the expression in homogeneous coordinates. Equation 5.12 is thus on the

form

$$\mathbf{r}_o^i = \mathbf{K}\mathbf{R}_c^w \left[\mathbf{I} \mid -\mathbf{C}^w\right] \mathbf{r}_o^w \tag{5.13}$$

which is a general mapping given by a pinhole camera model. Defining the matrix $\mathbf{P} = \mathbf{K}\mathbf{R}_c^w \left[\mathbf{I} \mid -\mathbf{C}^w\right]$, one can set

$$\mathbf{r}_o^i = \mathbf{P}\mathbf{r}_o^w \tag{5.14}$$

Using homogeneous coordinates and assuming $z_w = 0$ to redefine $\mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_4 \end{bmatrix}$, the world position is given by equation 5.15.

$$\mathbf{r}_o^w = \mathbf{P}^{-1}\mathbf{r}_o^i \tag{5.15}$$

This is written on homogeneous form. Normalizing is done by deviding $\mathbf{r}_o^w$ by its third element to get the inhomogen form.

5.5 Object Area

The area of the objects detected will be calculated in pixels from moment calculations. This area is converted to an understandable parameter, i.e. m^2. The total area in pixels of the image is found from $a_{total}^{pix} = w^{pix} d^{pix}$. Next, the total area seen on the ground is found from $a_{total}^{m} = wd$, where w and d are found from equation 5.10. When using the back-projection method, positions of all four corners of the image can be used to find correct values for w and d to compute the area. This is not implemented, as estimated area differences are negligible at small orientation changes and required accuracy is not as high as position estimates.

Using these relations, the area of the object in meters can be found. First, the algorithm calculates the area of the object in pixels. At last, the object area is found from

$$a_{object}^{m} = \frac{a_{object}^{pix}}{wd} a_{total}^{m} \tag{5.16}$$

When the area is calculated from pixels within a contour, any shape can be detected.

5.6 Object Identification

Doing object detection involves identification of different objects, as more than one object can be detected. Position and size will be identifiers in this project. The polygon shape is also an attribute that can be utilized.

The size of each detected object will be stored together with the position calculated as shown in Section 5.4. For the algorithm to decide if an object is detected before, it will investigate its position. The following assumptions and requirements are made.

- If the estimated position of an object overlaps a previously detected object in another iteration, the same object has been detected before and will not be added.
- An object is not categorized as an iceberg until the whole object is within the camera frame and the estimated area is above a minimum value.
- Objects that are not yet fully inside the frame are considered as *part of iceberg*.
- Detected objects are categorized according to its size relative to predefined limits. Small objects are displayed as *ice floe*, medium objects are assumed to be *icebergs* while very large objects are assumed to be *solid ice*. Mark that these are assumptions, and should be investigated further.

Using the requirement that object positions cannot overlap means the distance d between the objects must be larger than the sum of the radius for each object.

$$d > r_1 + r_2 \tag{5.17}$$

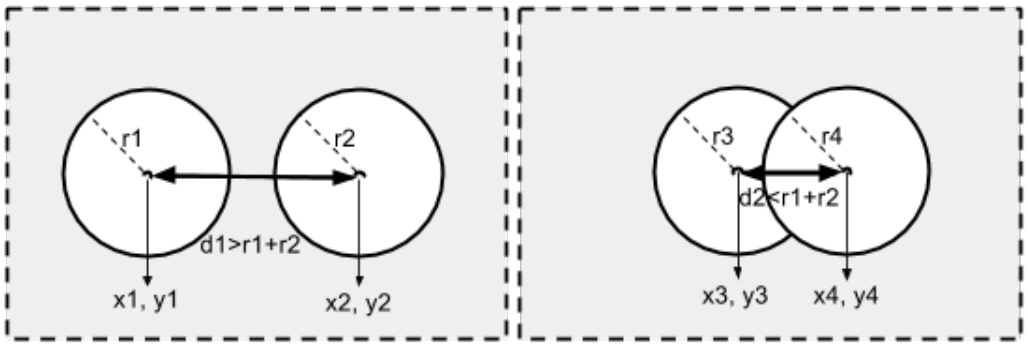

FIGURE 5.10: Examples illustrating if the identification algorithm will register the detected object or not. The left frame illustrates two objects detected by the camera, while the objects in the right frame is defined as one single object.

Each loop through the algorithm will check if the position has changed with more than the sum of the largest radius for each detected object. The reason for choosing this approach is that position estimates might be inaccurate and that the estimated position of an object that is not entirely in the image frame will change when larger areas are present in later loops through the program. Identification is crusial to avoid spotting the same object over and over while in the image frame. The identification algorithm does not handle cases where objects are moving at high velocities.

5.7 Object Tracking

Ice is moving with currents and winds. Mapping the location of an iceberg at a given time might be helpful, but it is even more interesting to know where it is in a given time. There are several approaches for this problem.

The drone could track an iceberg and keep it in the camera frame for a sufficient time period to register movements and calculate the direction of motion. This approach has two problems.

1. If there are many icebergs, the strategy would be very time consuming

2. If many icebergs are located in one image frame, it might be challenging to decide how many to physically track

One option might be designing an algorithm tracking multiple objects at the same time, but keeping all objects within the frame would be challenging. Another strategy could be to visit (and map) all icebergs, give each iceberg an individual ID and later return to each iceberg to determine how far it has moved. Main challenges for this approach is how to locate and recognize individual icebergs when doing the second visit. Object motion tracking is not designed and implemented in this project, but can be considered in future work.

Chapter 6

Implementation

As the design for an autonomous object mapping system has been presented in Chapter 5, this chapter will focus on how the design is implemented using the Parrot AR. Drone. First, the chosen programming language, platform and use of external libraries is presented. Next, guidance, navigation and control for AR. Drone is implemented to obtain autonomous path following. The implementation of the position PID controller is explained, including sending of tracking data over UDP. At last, the implementation of the image recognition strategy is presented. Only submodules of the implementation is shown, due to the readability as well as the relevance for project goals. For the complete implementation, see attached code files from DVD. The code structure is presented in Appendix A.

6.1 Choosing Programming Language and Platform

A brief study will be presented to argue the chosen programming language for the project. Multiple programming languages have been used to implement programs for the AR. Drone. Python and Java are examples of used languages. The interested reader can have a look at *python-ardrone*[15] and *Javadrone*[16]. Matlab has also been used in several relevant projects.

Java is an object oriented programming language, and the developed *Application programming interface* (API) mentioned above could have been used as a code basis. This alternative proved to have one disadvantage. The library to be used for image recognition, OpenCV, is not natively compatible with Java. Even though there exist a connection between Java and OpenCV, *JavaCV*, it introduces another dependency in the system. Another disadvantage is the lack of support for the API. Java was thus concluded not to be used.

Python is a high level programming language with simple syntax compared to many low level programming languages. The code developed with python is converted to code understood by the computer while running. The fact that the code is not compiled, results in the program in most cases takes more time to complete. As a result of the lowered performance, real time image processing using OpenCV was briefly tested to be too demanding for the experiment. Especially when handeling many frames per second.

Matlab is widely used for similar purposes. Challenges arise when doing real time image processing. Raaen concluded in his work[2] that this challenge in addition to the disadvantage of an extra dependency when using OpenCV results in the suggestion not to use Matlab.

The Parrot AR. Drone is initially programmed using C and C++. C++ is a intermediate-level object oriented programming language. It is basically the same as the C programming languahe, except that C++ involves enhancements like classes. A rusult of the fact that the AR. Drone SDK is implemented in C, finding documentation online is fairly easy. As OpenCV is devoloped in C++, the chosen programming language is C++.

The development will be done on a Linux platform (Ubuntu 12.04), as it is generally a good combination with C programming as well as communicating with hardware.

6.2 Software Development Kit

The SDK for Parrot AR. Drone is briefly presented in section 3.3.2. It is used as a basis for the library *cvdrone*[17]. The cvdrone software is used as a basis in this project, as most of the code for communication methods and low level interaction with hardware is modified from C (in the SDK) to C++. It is not a direct translation, but has the same main modules.

Controlling the AR.Drone is done through 3 main communication services.

- Commands: *AT** commands are sent from a client over UDP to control and configure the drone. These commands are sent constantly, 30 times per second.
- Navdata: Information from sensors onboard the drone is called *navdata*. Navdata may i.e. be angles, engine rotation speed and battery status, and are sent from the drone to the client over UDP.
- Video stream: A video stream is constantly sent from the drone on a third UDP port. These images can be processed on the client computer and be used for several purposes.

AT* commands are organized in management threads which collect commands sent by all other threads, and send them with correct sequence numbers to avoid the drone from processing old commands. A navdata management tread and a video management thread automatically receives the navdata and video stream, decodes it and provides the client with data that is ready to be used as inputs to any control system. The control threads handle requests from other threads for sending reliable commands and checks for drone acknowledgements [5].

Next, some of the most important commands and functions are given and explained. The following is an example of an AT* command that is used to move the drone:

```
AT*PCMD = seq, flags, phi, theta, gaz, yaw
```

where

- seq: Sequence number to make sure the drone is not processing old commands
- flags: Flag enabling the use of progressive commands. Passing the argument 0 to this flag makes the drone enter a *hovering* mode
- phi (ϕ): Left/right angle $\in [-1.0; +1.0]$
- theta (θ): Front/back angle $\in [-1.0; +1.0]$
- gaz: Vertical speed $\in [-1.0; +1.0]$
- yaw (ψ): Angular speed $\in [-1.0; +1.0]$

The command is used in the *cvdrone* library function

```
move3D(double vx, double vy, double vz, double vr;)
```

which is used to move the drone in 3D space. Given arguments are velocities in m/s. The function converts velocities in m/s to motor thrusts which results in changes in θ, ϕ and ψ.

Similar AT* commands are used to select which navdata to send.

6.3 Module Architecture

Figure 6.1 illustrates main modules of the implementation. Not all modules of the complete system are given, but GNC for the drone and important tools are presented. This architecture is intended for the testbed, and may thus not be

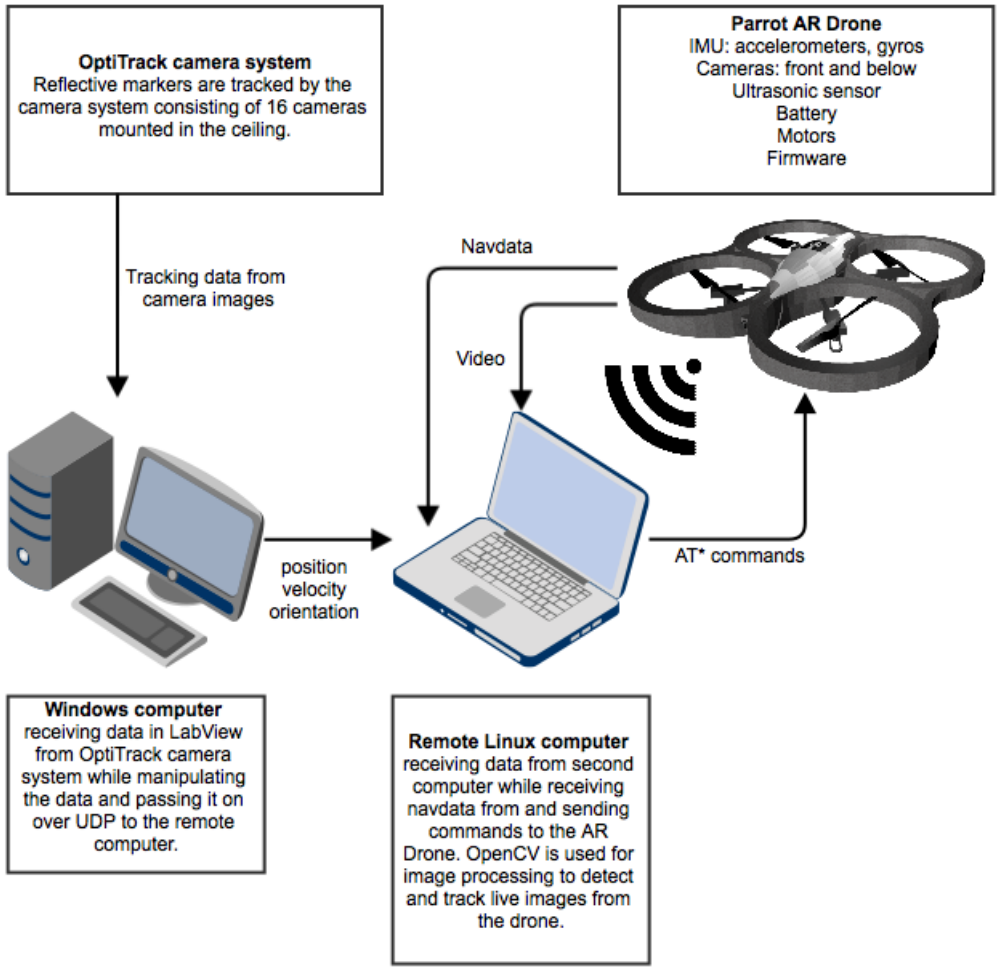

FIGURE 6.1: Basic system architecture illustrating communication between measurement systems, remote computer and the drone, as well as main tools used in the implementation.

applicable in a real experiment of ice mapping at the ocean surface. Arguments for this hypothesis will be discussed in Chapter 8.

The rest of this chapter will focus on the implemented program containing of a PID controller for path following, and how this controller can be used to perform a live object mapping strategy for ice management.

6.4 Sending Camera Setup Navdata from LabView Streaming Program

Position, velocity and orientation is acquired from the OptiTrack system in LabView. An initialized Tracking Tools project file is linked to read mesurements from a calibrated project. Methods for writing measurements to file and live plots are also implemented. The main module and functionality of this program is passing navigation data over UDP from a Windows computer which receives tracking data. OptiTrack is not available for Linux, which makes this second computer and operating system an extra dependency in an overall surface mapping strategy. Appendix D shows how to use the program. Figures D.1 and D.2 are LabView block diagrams. Implementation and functionality of the main modules are briefly presented in the following paragraph.

An infinite loop that sends UDP data contains of a method that sends data packets at a constant rate. Local and remote ports are defined, along with a remote IP address which is assosiated with an IPv4 address for the computer connecting to a network cable at the lab. Position, velocity and orientation are stored in variables and converted to bytes. All bytes included one start byte and one stop byte are added to an unsigned byte array of 20 bytes before converted to a string that is sent over UDP. Next, this string will be decoded at the receiving side.

6.5 Receiving Camera Setup Navdata from LabView Streaming Program

An User Datagram Protocol (UDP) listener program is written to retrieve the data streamed by the LabView program. UDP is not a three-way handshake protocol like TCP, and can thus not guarantee that data actually arrive at the receiver. For an application like this, UDP will be faster and provide more rapid measurements

which generates a level of redundancy. In case several measurements are lost, the program will still function and the missed packets are not relevant as the drone has moved.

Listening for UPD packets at a defined port is done in a separate thread within the main C++ program. The called function is decalred as

```
void* udp_listener(void *threadInfo);
```

In the main program, a thread variable is defined and the thread is created using Pthread.

```
pthread_t udp_listener_thread;
pthread_create(&udp_listener_thread, NULL, udp_listener, NULL);
```

A struct with the data to be recieved is declared and initialized included mutex. A mutex lock makes sure to give the code exclusive access, preventing other threads from executing concurrently and access the same memory locations. The mutex is locked before every time a variable in the struct is used.

```
typedef struct UDP_INFO {
        double posCam[3];
        double orientation[3];
        double velocity[3];
        pthread_mutex_t udp_info_mutex;
} UDP_INFO;

UDP_INFO udp_xyz_rpy;
```

Each variable is presented in two bytes, meaning position, velocity and orientation consist of 18 bytes in total. In addition, a START and a STOP byte are used to define the beginning and end of a package. Thus, a total of 20 bytes are received over UDP.

6.5.1 UDP Packet Listener

The UDP packet listener function was declared above. Some complementary details are presented here.

The *udp_listener(void)* method first initializes the mutex before running the *getaddrinfo()* method that converts human-readable text strings representing hostnames or IP addresses into a dynamically allocated linked list of *struct addrinfo* structures. Next, the result is bound to a socket. If the initialization process is successful and data is ready to be received on a socket, a *while* loop receives the data and adds it to a buffer. At last, a mutex lock is applied when reading from the buffer. Storing i.e. the *x* coordinate is done with the following code.

```
pthread_mutex_lock(&udp_xyz_rpy.udp_info_mutex);
    udp_xyz_rpy.posCam[0] = getShort(buf[2],buf[1])/1000;
pthread_mutex_unlock(&udp_xyz_rpy.udp_info_mutex);
```

Knowing that one byte consist of 8 bits, the method *getShort()* is used to read the correct bits.

```
double getShort( unsigned char high, unsigned char low ) {
        short shrt = (((unsigned short)high) << 8 | ((unsigned short)low));
        return (double)shrt;
}
```

6.6 Guidance, Navigation and Control

Strategies for guidance, navigation and control (GNC) of the drone is implemented in C++. The GNC system includes positioning of the drone, waypoint generation within a wanted area as well as a PID controller for waypoint following.

6.6.1 Positioning using Dead Reckoning

The basic mathematical models and expressions needed to implement dead reckoning is presented in section 3.5, while the design of the method is given in section 5.2.2. Dead reckoning is not considered in an overall system design, but rather evaluated as a positioning system that may be used in an integrated navigation system or to provide redundancy in measurements. The implementation of the strategy is thus not a focus here. For implementation details, see the project work[11] or complete attached code.

6.6.2 Positioning using Camera-based Indoor Navigation System

Position, velocity and orientation are streamed to the C++ program over UDP according to 6.5. To read the variable of question in the main program, i.e. the estimated x position of the tracked object, *getPosX()* is called

```
double getPosX()        {
        double val;
        pthread_mutex_lock(&udp_xyz_rpy.udp_info_mutex);
        val = udp_xyz_rpy.posCam[0];
        pthread_mutex_unlock(&udp_xyz_rpy.udp_info_mutex);
        return val;
}
```

Similar methods are used to read measured velocity and orientation.

6.6.3 Waypoint Generation

Waypoints are automatically generated by a function *setWaypoint*. The system operator is asked for desired length and width of the search field at startup (*distanceNorth* and *destanceEast*).

The method is declared as

```
double * setWaypoint(double distanceNorth, double distanceEast,
    int & numberOfWaypointsReached, double waypoint[]);
```

Listing 6.1: Waypoint generation function

The function returns a pointer to an array with waypoints in the world frame. This array is updated each time a waypoint is reached, so that new waypoints are given as long as they exist. *waypoint[]* is the array to be updated. The camera opening angle is globally defined and used in calculations to find where the drone must move in order to cover uncovered areas. *nWaypoints* is returned from the next method computing how many waypoints are needed to cover a desired area with the camera.

```
int setNumberOfWaypoints(double searchDistanceNorth);
```

6.6.4 PID Controller for Waypoint Following

The path tracking controller is based on measurements of position, velocity and orientation. It is implemented with the following arguments

```
autoPilot(int & nWaypointsReached, int numberOfWaypoints, double pos[],
double velocity[], double waypoint[]);
```

Listing 6.2: Autopilot method that takes the drone between waypoints

Along track and cross track errors are calculated according to equation 5.3. Errors are used in calculations of controller outputs in cross track and along track directions. Not all details are given, i.e. how velocities *vx* and *vy* are set using PID contributions and how the *move3D()* function is called.

```
double e_ct_temp[2], d_e_ct_temp[2], d_e_at_temp[2];

d_e_at = 0;
e_ct = 0;
d_e_ct = 0;

// Compute errors for controller
for(int i=0;i<2;i++) {
    e_ct_temp[i] = (waypoint[i]-pos[i])*n_vector[i];
    e_ct += e_ct_temp[i];

    d_e_ct_temp[i] = -velocity[i] * n_vector[i];
    d_e_ct = d_e_ct + d_e_ct_temp[i];

    d_e_at_temp[i] = v_des - (velocity[i]*t_vector[i]);
    d_e_at = d_e_at + d_e_at_temp[i];
}

// Integral
i_at = (d_e_at + i_at) * dt;
i_ct = (d_e_ct + i_ct) * dt;

// PID contribution for along track and cross track
double PID_at= (K_p_at * d_e_at) + (K_i_at * i_at);
double PID_ct = (K_p_ct * e_ct) + (K_d_ct * d_e_ct) + (K_i_ct * i_ct);
}
```

LISTING 6.3: PID controller implementation

6.7 OpenCV for Object Detection

Algorithms from OpenCV are used to perform object detection while the drone is in air. Important aspects from the implemented object detection algorithm will be presented. The section covers how an image from the camera is manipulated before several algorithms for detection are used. The implementation of back-projection to world positions is also briefly presented.

6.7.1 Manipulate Image from the Quadcopter

Before the detection algorithm can be applied, the real time images must be converted and noise will be filtered.

```
// Get an image from the drone
Mat image = ardrone.getImage();

// Binalize/threshold image
Mat binalized;
Scalar lower(minH, minS, minV);
Scalar upper(maxH, maxS, maxV);
inRange(image, lower, upper, binalized);

// Morphology Transformation to remove noise
morphOps(binalized);

// Show resulting image on the screen
imshow("Binalized image", binalized);

// Detect objects
detectObject(binalized, image, sizes, posCam);

// Show live image for realtime tracking and detection
imshow("Live tracking image", image);
```

LISTING 6.4: Get image feed from the drone and convert the image to black & white

Images from the AR. Drone vertical camera is stored in a *Mat* which makes it possible for OpenCV methods to be called directly on the image. Next, all pixels are converted to 0 or 1, which is black or white. The function *detectObjects* is the implemented algorithm described in the next section.

6.7.2 Object Detection Algorithm

In the lab experiment, objects are detected based on color and size. Calibration is done at startup according to Appendix B to make the program register only specific

color spectrums, i.e. white. Pixels are set to 1 (white) within object contours. Main modules and functionality of the implemented detection algorithm is illustrated in Figure 6.2.

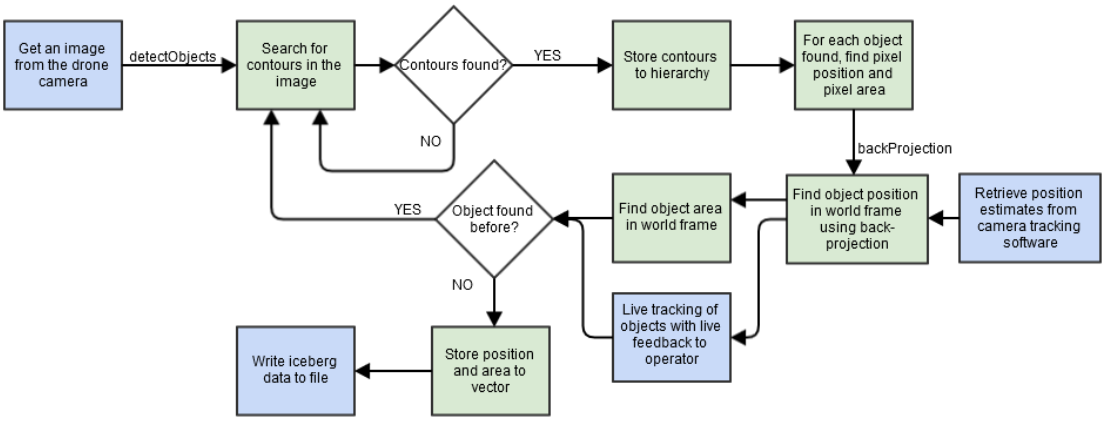

FIGURE 6.2: Flowchart for object detection algorithm. Blue boxes represnt external dataflow while green boxes are dataflow inside the algorithm/method.

The most interesting function used from OpenCV in this algorithm is *findContours()* which is called with the following parameters

```
findContours(clonedBinalized, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
```

LISTING 6.5: Function call used in algorithm for object detection

A cloned version of the binalized (thresholded) image is sent as input parameter, as this image is changed by *findContours()*. Before the function call, contours and hierarchy is declared as

```
vector<vector<cv::Point> > contours;
vector<Vec4i> hierarchy;
```

LISTING 6.6: Vectors for storing of contour points and hierarchy

where *contours* contains of a detected contours in the image. Each contour is stored as a vector of points. *hierarchy* is a vector containing of information about the image topology, i.e. number of contours. *cv::RETR_CCOMP* is a mode paramter

that retrieves all of the contours and organizes them into a two-level hierarchy, i.e. both external and internal contours. *cv::CHAIN_APPROX_SIMPLE* is a method parameter that compresses horizontal, vertical, and diagonal segments and leaves only their end points.

6.7.3 Back-projection of a Pixel Point to World Coordinates

Back-projection calculations are described in section 5.4.2. For testing, these calculations were implemented in Matlab. Orientations and positions were set manually. The C++ implementation applies the OpenCV *cv::Mat()* container for matrix calculations which is basically a class containing of two data parts: the matrix header (matrix size and how it is stored) and a pointer to the matrix values. Generally, this variable is used for image analysis, but also provide standard matrix behaviour. The implemented back-projection algorithm is illustrated in figure 6.3.

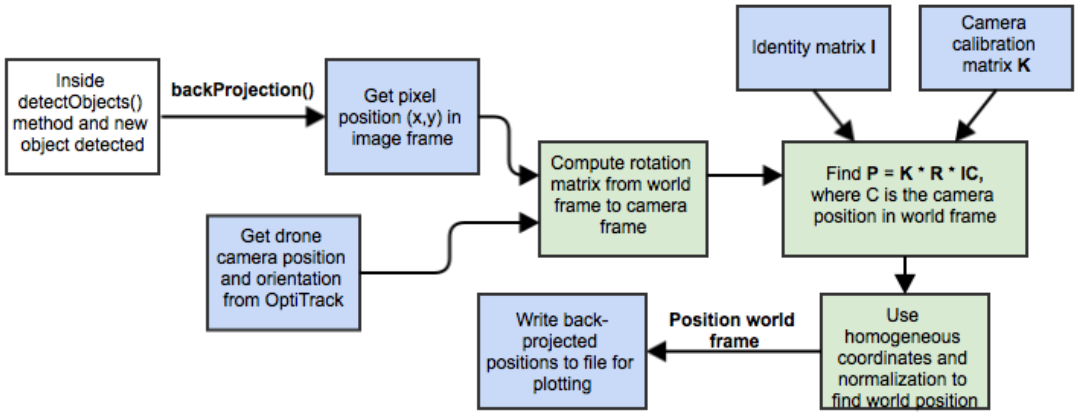

FIGURE 6.3: Flowchart illustrating back-projection algorithm. Blue boxes are external dataflow while green boxes are dataflow or computations inside the algorithm.

Object positions, areas and polygons are written to text files before imported in Matlab for plotting. A real time map with drone position and object position is also available when running the program.

Chapter 7

Simulations, Testing and Results

In order to determine the functionality of the overall system, it is tested in the indoor lab setup. Submodules will be simulated and compared with drone and camera system measurements. Autonomous flight control and object detection are main modules to be tested. *The reader is strongly recommended to watch the attached video for live examples.*

7.1 Simple Quadcopter Model in Simulink

A simplified model of the quadcopter and the positioning algorithm is implemented in Matlab Simulink. The goal for the simulation is to generate plots of the position of the drone in the XY-plane using PID control. The designed waypoint generator is also tested. A simplified block diagram of the Simulink implementation is presented in Figure 7.1. Further details can be found in Appendix C. The simulation does not contain of a complete model of drone dynamics, as attitude control is implemented in the firmware.

The controller block contains of Matlab code implementing the PID controller as described in Section 6.6.4. As shown in Figure 7.1, waypoints are given as

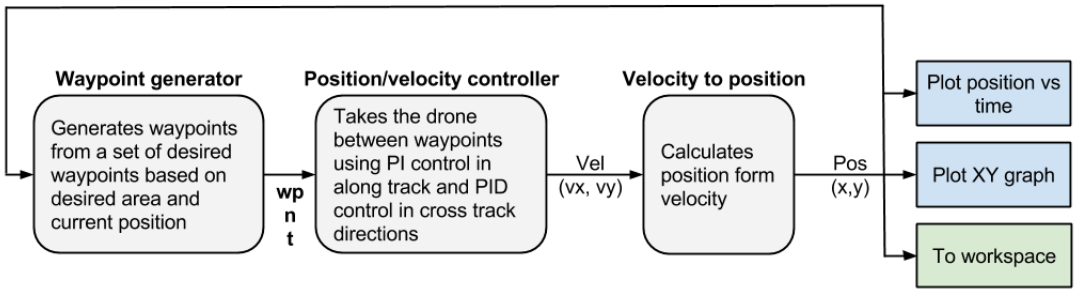

FIGURE 7.1: Simple Simulink model for testing the waypoint generator and position controller.

references to the controller. These waypoints are automatically generated and updated as waypoints are reached.

Waypoints in the world frame are sent along with its respective **n** (normal) and **t** (tangential) vectors from the waypoint generator. These vectors are used in the controller to generate velocity gains to control the drone position. In simulation, velocity is integrated to obtain position displacements. The positions are plotted both with respect to time and as a XY-graph. A position vector is sent to workspace for plotting together with measuements.

7.2 Testing of Autonomous Flight Controller

The path following controller is tested by experiments in the Snake Robot Lab. A trackable marker is attached to the vehicle so that position and orientation can be tracked in 3D space. As the altitude controller was evaluated to perform well in Section 3.6.2, additional altitude control is not implemented. Testing of the position controller is therefore done in 2D. Position control in 3D can easily be implemented as an extesion of the current implementation.

7.2.1 Controller Tuning

Manual tuning of the PID controller is done to provide desired behaviour. Drone positions for a tuned set of controller parameters will be presented. A step response is applied by placing the drone at a distance from the desired trajectory in both cross track and along track directions. A typical desired behaviour is that the drone will quickly reach the trajectory in cross track direction while moving slowly towards the waypoint in along track direction. Position differences may be caused by manual placement of the drone.

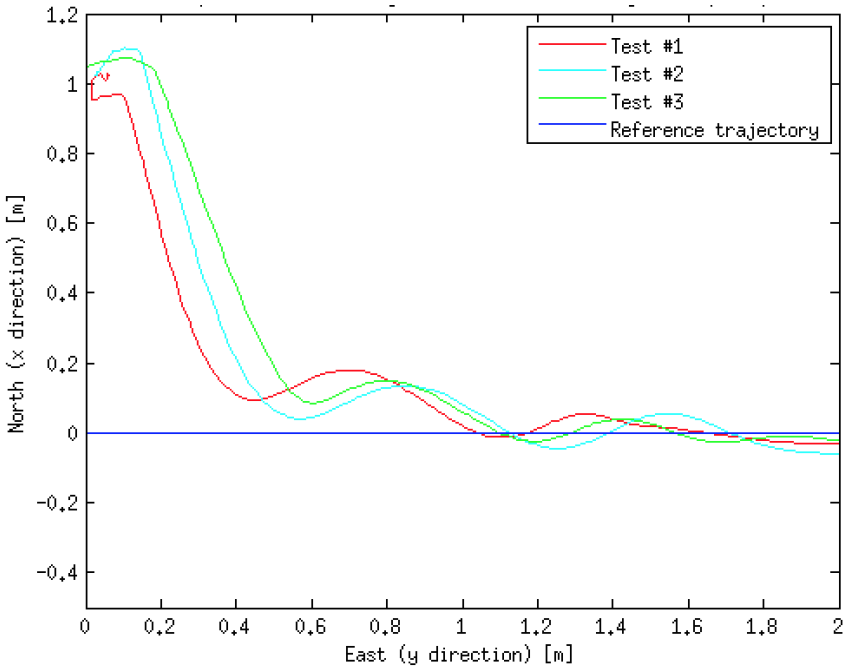

FIGURE 7.2: Position plot of path when a step response is applied. The drone is placed at about 1 meter from the desired trajectory in x direction. The waypoint is set to $x = 0,\ y = 3$ while tuning the controller to get a desired response.

Cross track controller parameters are found manually by first changing $K_{p,ct}$ until the quadcopter oscillates in cross track direction. When this value is found, the parameter is set to about half. Next, $K_{i,ct}$ is found so that position offsets are corrected within a desired time. At last, $K_{d,ct}$ is increased so that the drone reaches its reference in short time without decisive overshoot.

TABLE 7.1: PID tuning parameters

Parameter	**Tuned value**
$K_{p,at}$	0.6
$K_{i,at}$	0.2
$K_{p,ct}$	1.8
$K_{d,ct}$	1.3
$K_{i,ct}$	2.2

7.2.2 Path Following

Using tuned variables found in the previous section, position measurements through a complete test path is plotted in Figure 7.3. Each coloured line represents individual flights, where the desired path is a simple search pattern. Origin is defined in the camera system[1].

Minor deviations at takeoff are expected. Through the path, the drone reaches all waypoints and position sticks to the reference trajectory with acceptable deviations. The larges deviations seem to occure when turning at waypoints, where along track and cross track directions are changed.

Next, inflight camera measurements are compared to dead reckoning measurements used in the project work[11] calculated according to sections 5.2.2 and 6.6.1. Figure 7.4 illustartes measured positions.

As suspected, Figure 7.4 indicates that dead reckoning estimates drift with time, and result in significant position errors. The pattern shape is fairly identical though, but magnitudes deviate with up to about 30 %.

[1]Camera system origin is fixed in the calibration process. This point is also chosen to be the takeoff position, but could just as well have been in $x = 0$, $y = 2$ or any other position. In later tests, the world frame origin will be redefined to a convenient position in the lab room.

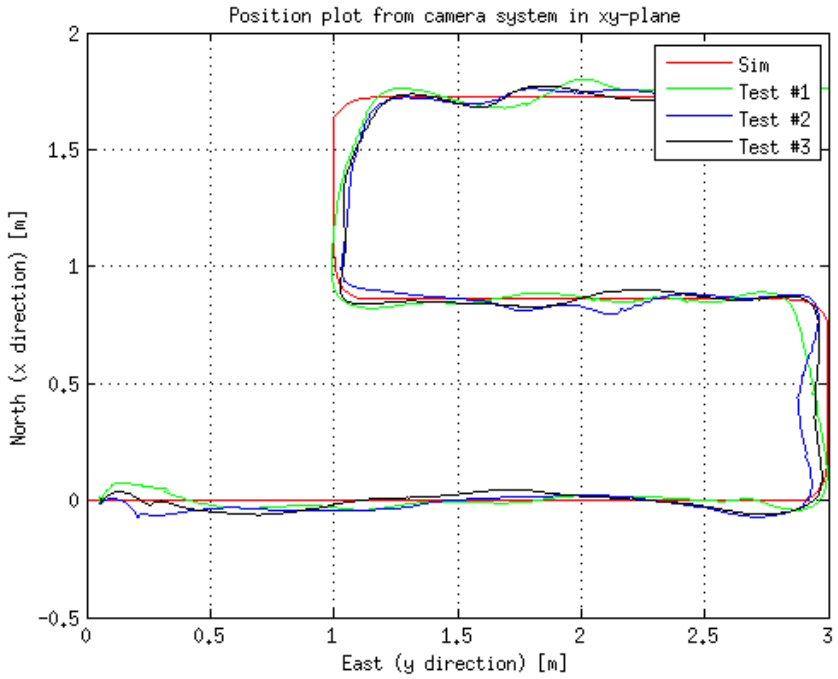

FIGURE 7.3: Simulated comapared with measured position of the drone in meters with takeoff in camera system origin.

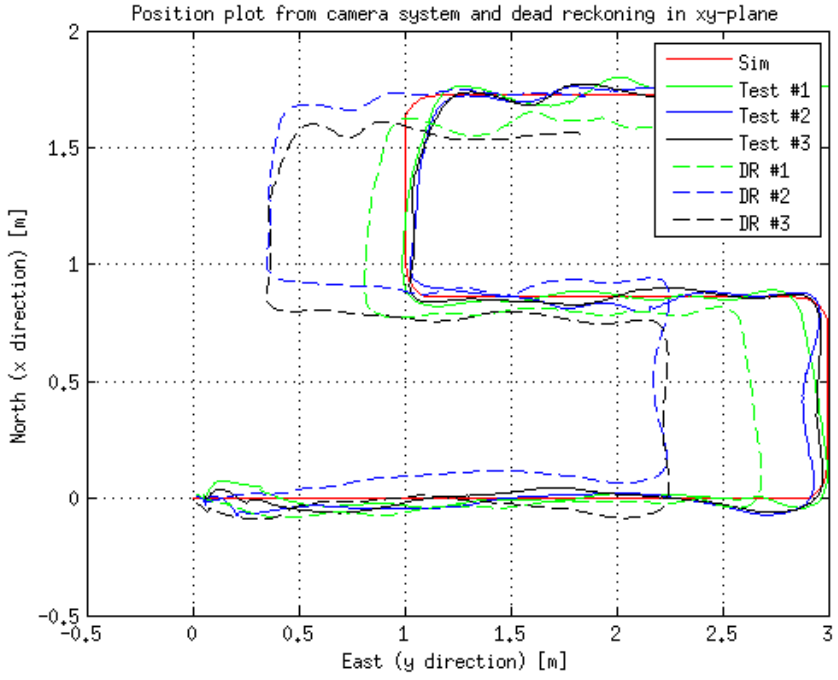

FIGURE 7.4: Simulated comapared with measured position of the drone in meters. Dead reckoning estimates for respective tests are presented with dotted lines.

7.2.3 Velocity Profile

The PID controller is tuned to adapt quickly to the desired trajectory. Along track velocity is close to constant, while cross track velocities will change with changing deviations. This is illustrated in an experiment, which is also used to briefly evaluate dead reckoning velocity estimates. The quadcopter velocity estimates are plotted for a part of the path in Figure 7.5.

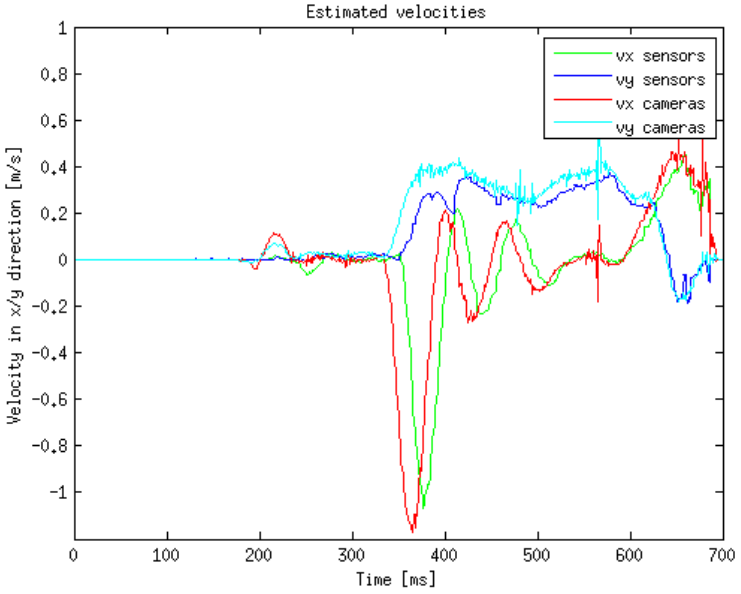

FIGURE 7.5: Velocity profile of test 1 from figure 7.2. Cross track velocities (vx in plot) are higher than along track velocities.

Dead reckoning velocity estimates are close to camera measurements in magnitude, but seem to be delayed by several milliseconds. However, estimates are acceptable and may be fitted as the main input source to the controller if external velocity estimates were not available. This is not an issue while using the camera system though, which provides real time velocity estimates.

7.3 Testing of Back-Projection from a 2D Point to World Frame

Back-projection of a 2D pixel point to world coordinates has been presented and implemented. A simple test is performed with the drone camera using measurements from OptiTrack to evaluate its performance. To be able to test that the algorithm compensates for orientation misalignments, a fixed orientation and position is used, i.e. the quadcopter is not moving. Orientation and camera position are set according to

$$\mathbf{\Theta}^w = \begin{bmatrix} \phi^w \\ \theta^w \\ \psi^w \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\pi}{4} \\ 0 \end{bmatrix} \quad , \quad \mathbf{\Theta}^c = \begin{bmatrix} \phi^c \\ \theta^c \\ \psi^c \end{bmatrix} = \begin{bmatrix} \frac{\pi}{4} \\ 0 \\ 0 \end{bmatrix} \quad , \quad \mathbf{C}^w = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Screenshots are taken from the live object tracking program, and can be seen in Figures 7.6 and 7.7. Three objects are inside FOV. Tables 7.2 and 7.3 presents estimated object positions and deviations. The same experiment has been done for both the simplified method assuming no changes in roll, pitch or yaw and using back-projection.

FIGURE 7.6: Test of brack-projection using fixed orientation and camera position.

FIGURE 7.7: Test of simplified method using fixed orientation and camera position.

TABLE 7.2: Simplified method to world coordinates

Measurement source	**Object 1**	**Object 2**	**Object 3**
Position (X,Y) simplified [m]	(0.19, -0.24)	(-0.10, 0.15)	(0.23, 0.21)
Position (X,Y) measured [m]	(1.23, -0.38)	(0.72, 0.24)	(1.42, 0.42)
Deviation [m]	(1.04, -0.14)	(0.82, 0.09)	(1.19, 0.21)

TABLE 7.3: Back-projection to world coordinates

Measurement source	**Object 1**	**Object 2**	**Object 3**
Position (X,Y) back-projection [m]	(1.27, -0.36)	(0.76, 0.24)	(1.47, 0.42)
Position (X,Y) measured [m]	(1.23, -0.38)	(0.72, 0.24)	(1.42, 0.42)
Deviation [m]	(0.04, 0.02)	(0.04, 0.0)	(0.05, 0.0)

Various experiments are performed using different orientations with similar results. Back-projection measurements prove to give good position estimates as long as the assumed flat ground plane (ocean) holds. Sources of error are mainly human errors when placing the camera (quadcopter) with fixed position and orientation. Calibration of the camera matrix may as well introduce minor misalignments. Roll, pitch or yaw are never as large as 45 deg in air. Measured maximum values from several test flights are about: $\phi = 10{-}15\ deg$, $\theta = 10{-}15\ deg$ and $\psi = 4{-}5\ deg$.

However, results from the simplified method have significant estimation errors in both X and Y directions as seen in table 7.2. Even though this experiment takes orientation arguments that are not representable for measurements in air, back-projection provide far better results which is important for position accuracy.

7.4 Testing of Object Detection and Mapping

To test the overall surface mapping strategy, white paper objects with different areas and shapes are placed randomly at the floor. The drone is placed at an initial position and the search area is defined to make the drone cover a desired area.

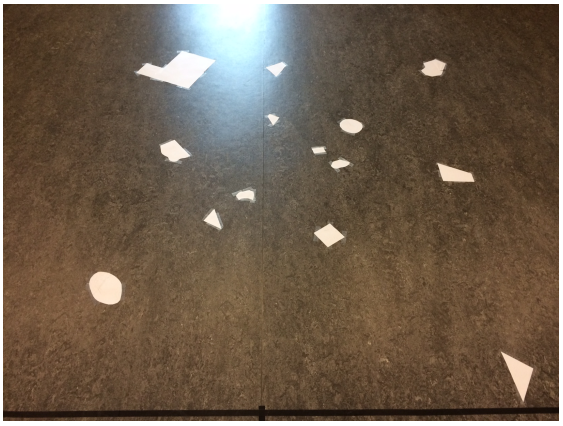

FIGURE 7.8: Objects at the ground.

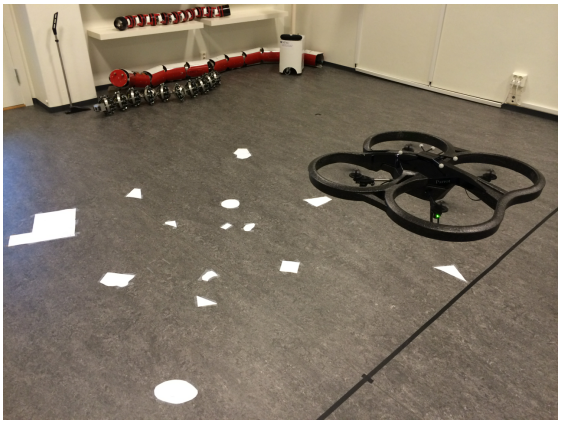

FIGURE 7.9: Drone in air.

The surface mapping strategy is designed with two ways of finding object positions. Both approaches will be tested and discussed.

Positions (m), areas (m^2) and the polygon points of each detected object are written to files and loaded in Matlab. The quadcopter program is started, which makes the drone fly autonomously between waypoints while detecting objects. Object data of interest are plotted in world coordinates in Figures 7.10 and 7.11. World coordinate origin is defined by the OptiTrack camera calibration. At startup, the program is asked to cover the area within 2×2 meters. Waypoints are automatically set by the program as described in Section 5.3.3 with a goal of avoiding blind spots. Two tests are performed. To make the plot readable, only objects that are completely in

the frame are plotted. However, partial objects are also detected and written to a separate file. A FOV overlap is set in the waypoint generation method, to increase the probability that all objects will be totally inside the frame. Partial objects can be shown in a seperate plot if it is found important, and are also displayed in live tracking. In Figures 7.10 and 7.11, the smallest objects are filered as ice floes, and thus not plotted. Two tests are presented.

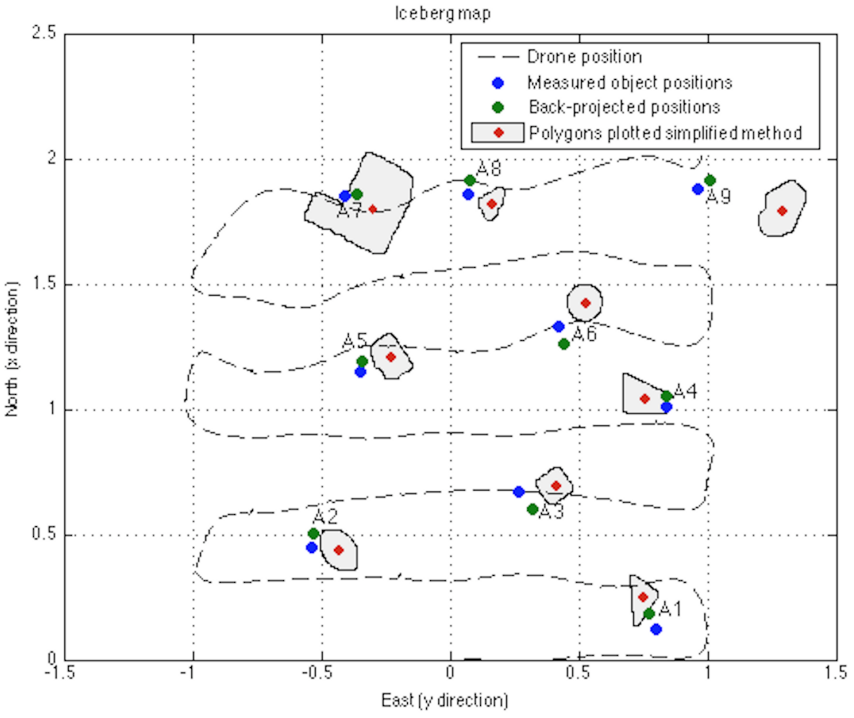

FIGURE 7.10: Plot of iceberg map from test number 1. Stored polygons are plotted for the simplified method only, . CoG positions from back-projected and simplified method are compared.

Areas of detected objects are measured in OpenCV and written to file along with position estimates. Areas for icebergs *A1*, *A2*, *An* are given in table 7.4. Camparing these estimates, one can see that area measurements are close to identical for both tests. Actual areas are not computed for all shapes. Object areas that are easy to measure are controlled to deviate with less than 10%.

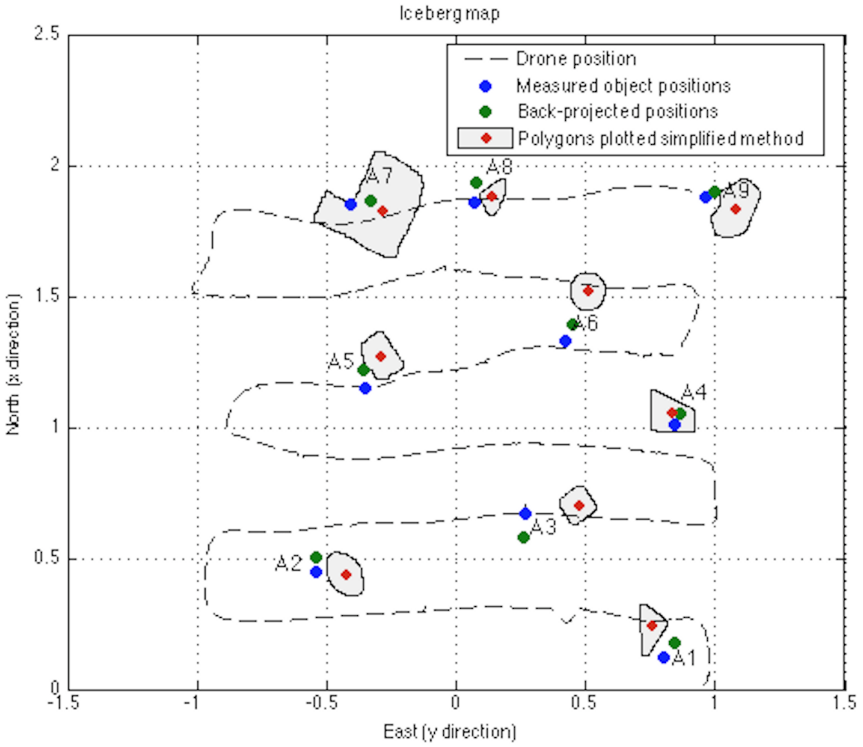

FIGURE 7.11: Plot of iceberg map from test number 2. Stored polygons are plotted for the simplified method only. CoG positions from back-projected and simplified method are compared.

TABLE 7.4: Estimated iceberg areas from OpenCV

Iceberg	**Area Test 1** [m^2]	**Area Test 2** [m^2]
A1	0.0115	0.0117
A2	0.0162	0.0163
A3	0.0118	0.0119
A4	0.0184	0.0185
A5	0.0167	0.0169
A6	0.0136	0.0138
A7	0.0826	0.0883
A8	0.0076	0.0053
A9	0.0240	0.0234

Figures 7.10 and 7.11 are results of the surface mapping strategy. Object shapes are

plotted from stored polygon data. Corners are slightly rounded due to morphology transformations described in Section 4.3.3. Both object detection approaches are tested and plotted to be compared. Polygons are only plotted for the simplified method for finding object positions, which makes the figure readable[2]. Blue dots represent measured CoG positions of objects, green dots are estimated positions using back-projection while red dots inside polygons are positions estimated using the simplified method. Polygon shapes are not back-projected, as they are not considered to have significant shape changes at small orientation changes. Position deviations for both designs are presented in table 7.5.

TABLE 7.5: Deviations for position estimates for simplified method and back-projection using data from test number 1

Iceberg	**Position (x,y)** [m]	**Dev. simpl.** [m]	**Dev. back-proj.** [m]
A1	$(0.12, 0.80)$	$(-0.13, 0.05)$	$(-0.06, 0.03)$
A2	$(0.45, -0.54)$	$(0.03, -0.15)$	$(-0.05, -0.01)$
A3	$(0.67, 0.27)$	$(-0.02, -0.13)$	$(0.07, -0.05)$
A4	$(1.01, 0.84)$	$(-0.02, 0.09)$	$(-0.03, 0.00)$
A5	$(1.15, -0.35)$	$(-0.06, -0.11)$	$(-0.04, -0.01)$
A6	$(1.33, 0.42)$	$(-0.08, -0.14)$	$(0.07 - 0.02)$
A7	$(1.85, -0.41)$	$(0.07, -0.12)$	$(-0.01, -0.05)$
A8	$(1.86, 0.07)$	$(0.06, -0.10)$	$(-0.06 - 0.01)$
A9	$(1.88, 0.96)$	$(0.09, -0.33)$	$(-0.03 - 0.05)$
Mean dev. (abs)		$(0.062, 0.136)$	$(0.046, 0.025)$

Table 7.5 givs a mean absolute value deviation of $x = 6.2\ cm$, $y = 13.6\ cm$ for the simplified method and $x = 4.6\ cm$, $y = 2.5\ cm$ when using back-projection. Back-projection provides a significant improvement in Y-direction.

[2]Another reason for not plotting polygons for the back-projected method is to avoid implemention time for back-projection of each polygon point in Matlab. Back-projection of CoG is done in C++.

CoG estimates for the simplified method are displaced in the direction of motion. Due to pitch and roll changes, the camera is tilted so that a pixel is projected in the backward direction with regard to the motion. When neglecting the tilt angle, objects are estimated to be further away than they actually are. Roll, pitch and yaw angles are low during the whole operation, but the polygon of object $A9$ in Figure 7.10 is a good example. At the time this object was detected in the frame, values for roll and pitch were relatively large in contrast to values at other object locations. Measurements are made while in the process of turning at a waypoint, which demonstrates how much influence the camera orientation has on position estimates.

Chapter 8

Discussion

As the whole system has been tested, several findings will be discussed. Emphasis of this discussion will focus on the concept that is implemented, and not how such a system will work in a real experiment with outdoor conditions. Section 8.5 will briefly discuss some of the challenges that will arise in the real world.

8.1 Control Method and Waypoint Guidance

Controlling the position of the drone using PID control has been tested, and proved to follow a desired trajectory without large position deviations, shown in Figure 7.3. Tuning the controller in both along track and cross track directions made it easy to control velocities. Minimizing or eliminating blind spots while at the same time minimizing travelling distance and number of waypoints, requires the controller to stick to the trajectory in cross track direction. The controller was thus tuned to be aggressive in cross-track direction.

A downside to the implemented controller is that statical **n** and **t** vectors are used. For a more comprehensive approach, these vectors should be computed online. Thus, paths with any shape can be followed using dynamically updated waypoints, which will provide an approach to make the drone follow a moving object

or paths with any feasible shape. For such controller, a time optimal feasible trajectory generator can be designed. Such method will compute a speed profile that traverses a sequence of waypoints in minimum time while satisfying acceleration and speed constraints. Time optimality is not considered important when doing lab experiments, where it is favourable to keep low velocities for optimal image capturing and processing with minimum noise. When doing surface mapping at a whole different scale above the ocean, high velocities may be preferable in order to cover as large area as possible within a given time. Time optimal speed and acceleration plans are often solved as optimization problems. Posing trajectory planning as an optimization problem has one major limitation. Using a quadcopter outside to cover large areas, one can assume most real time processing has to be done on a microcontroller onboard the drone. Thus, a microcontroller that is capable of solving a nonlinear optimization problem in real time is needed.

A PID controller was chosen for position control due to its design and tuning simplicity. Alternative controllers may provide advantageous behaviour. Various projects focusing on i.e. acrobatic maneuvers benefit from nonlinear methods such as input/output lenearization. Differential flatness guarantees that control inputs or trajectories are given using an output trajectory [18]. Backstepping controller design has been used for systems with extreme performance requirements [19][20], where flight tests were successfully performed outside. Other projects are based on LQR control [21][22], where UAVs are able to track trajectories with acceptable accuracy. MPC is another commonly used controller method that is not looked into in this thesis. Generally, these mentioned controller designs are more optimal and fitted for extreme performance requirements. However, PID control provides a good balance between simplicity an optimality for the application in this project.

The defined search algorithm is simple, but effective. An operator defining the search direction towards X (north) and Y (east) makes it possible to avoid searching in unwanted areas. These are fixed to the world frame. Rotations must be performed in order to define the search direction according to a wanted direction of motion for the ship and the direction of currents. Other search patterns may be

convenient in some use cases. Moving in a spiral to map a circle around the ship may be an extension of interest.

8.2 Navigation System

Figure 7.4 proves the assumption that inertial navigation using dead reckoning will drift with time. Quadcopter positions are found from integration of speeds, which are not precise. The yaw is measured by integrating a gyro signal, which will also drift on a large time scale. Using the OptiTrack marker tracking system results in accurate position measurements that are of great importance both for quadcopter position and velocity control and object mapping. With high accuracy for position and orientation estimates, OptiTrack is fitted for indoor lab experiments.

As suggested, an integrated navigation system using GPS and inertial navigation will be of use when doing outdoor experiments. GPS accuracy of about $1-10\ m$ is not sufficient as the only measurement source. An integrated solution could have been used with OptiTrack as well, to avoid possible erroneous measurements and provide redundancy. Section 5.2.3 propose such system, while Vik[9] goes into detail on GPS/INS integration.

8.3 Object Detection and Mapping

The object detection algorithm used for surface mapping is based on finding contours in an image filtered by color and size. As well as providing accurate position and area estimates, live object tracking is implemented. Multiple objects can be detected and tracked independently in one image frame, which provide numerous possibilities for gathering data. In this thesis, world frame positions, areas and polygon shapes are collected. Comparing object shapes (Section 9.1.1) and further analysis may be performed.

Figures 7.10 and 7.11 indicates that back-projected position estimates are more accurate than the simplified method. This result is expected, as the simplified method does not consider orientation changes. Most, if not all, estimated positions show this tendency. With other words, roll/pitch/yaw were unlike zero at all iceberg positions. However, positions estimated with back-projection also suffer from minor misalignments. The drone camera matrix may be inaccurate due to bad calibration, which may be an error source. In future work on the project, this matrix should be recalibrated.

A disadvantage for the back-projection method is that it relies on orientation measurements. In case these measurements are not correct, position estimates will be wrong. In the early testing phase, one object could be detected multiple times at different positions. Erroneous orientation measurements (unwanted spikes) were the error source. The issue was handeled using low pass filtering of orientation measurements combined with a mechanism checking for unlikely measurements. The approach is not flawless, as fast changes may occure in air. A integrated navigation system would be useful to provide redundant measurements and design a mechanism that guarantees accurate orientation input to the algorithm. GPS would not provide redundancy in orientation measurements (for outside experiments), which requires inertial measurements to be accurate. The implemented back-projection method assumes projection is done on a horisontal plane ($z = 0$). For object mapping in the ocean, this assumption holds. It is also assumed that the horizon is never in the image frame. Singularities at orientations of 90 *deg* is a problem when calculating rotation matrices. Due to the above assumption and that the drone will never fly at this orientation, singularities will not occure in normal experiments. A mechanism to make sure singularity never occures could be designed so that if 90 *deg* measurements are made, the last valid measurement will be used instead.

In cases where the whole object is not inside the image frame, partial objects can be plotted. One condition may be that only the largest detected part of an object at a given position is plotted. Detecting objects based on color and size may

not be sufficient for more complex setups or if the ground has similar colors as the objects, i.e. ice compared to reflections, waves etc. For normal conditions in the lab setup, light reflections are filtered as noise. Such filtering may result in actual objects being removed. However, this was not an issue while testing. By contrast, similar experiments in a real world may perform bad filtering due to numerous pixel intensities close to a thresholded interval. This hypothesis will not be discussed further, as it is not a focus in the thesis at hand.

8.4 Overall Approach

The AR. Drone is well fitted as a testbed for object mapping, as it is equipped with various sensors. Using the library *cvdrone* made it possible to focus on the main goal of the project, and not how to retrieve navdata or send commands to the drone. Combining OpenCV with the position and velocity controller and writing the complete program in C++ is a good choise, as the programming language is fast and well fitted for real-time applications. Developing on a Linux platform has several advantages considering dependencies, compared with i.e. Windows. OptiTrack provides accurate measurements which are decisive for the overall performance of both GNC and object mapping.

8.5 Lab Setup vs. Real World

As indicated, there is a large difference between the lab setup and experiments in the real world. First and foremost, the hardware of the drone is not fitted for outside use under heavy wind and weather conditions. The weight of only 420 g is one reason. If the weight is to be increased, motor power and battery capacity will also have to be increased. With current hardware, the battery capacity is about 16 minutes in air, which is not sufficent for long-term operations.

Data and commands are sent over WiFi. The range of normal Wifi is limited. Another communication technique have to be used in a real world experiment. Safety for the drone, other equipment or personnel is not a main focus in the design. Safe return to the ship must be designed, and extensive surveillance of onboard conditions must be taken into account to prevent the drone from getting lost in the sea. Using a camera for detection does have its restrictions in different weather conditions. Other sensors, or possibly combinations, have to be used in a real experiment. A study on fitted sensors and hardware is suggested as future work on the subject in Section 9.1.5.

Chapter 9

Conclusion and Recommendations

A surface mapping strategy has been successfully designed, implemented and tested. Two main modules are combined to make the drone fly autonomously between waypoints while searching for objects at the ground. To verify the modules and overall system performance, the position controller was implemented and simulated in Matlab Simulink. The object detection algorithm has been tested with both a method neglecting drone orientation and a method using back-projection of a 2D pixel point to world coordinates.

Testing of implemented strategies for waypoint guidance and position control using navigation data from OptiTrack resulted in low position deviations from the desired trajectory. High accuracy and update rate for position, velocity and orientation measurements provided the overall GNC system with satisfactory inputs.

Surface mapping using a camera also proved to provide good results. Combining image analysis with orientation data, the designed back-projection method has overall good performance. The larges measured deviation between estimated and real object CoG was 35 cm for the simplified method and 7 cm using back-projection. Mean deviations were $x = 6.2\ cm,\ y = 13.6\ cm$ and $x = 4.6\ cm,\ y =$

2.5 *cm* respectively. As orientation measurements increases, the simplified method error increases exponentially while back-projection error is unchanged.

Object detection is a wide and complex subject. In order to do time analysis of real icebergs and floating ice, heavy and complx algorithms are required. For the lab setup, the simplified detection algorithm is sufficient. If such system is to be applied for detecting real ice, the design would have to be changed. Several reasons are worth mentioning. Using a regular camera would be problematic at night, or in bad weather conditions. Different sensors would have to be used, either in combination with a camera or alone.

Main goals were to implement a GNC system for the quadcopter to be used in a surface mapping strategy. Images captured from air were supposed to build an ”iceberg map”. The assigned goals of the thesis were achieved. An overall strategy combining the AR. Drone, OptiTrack, OpenCV and the C++ library *cvdrone* is a good testbed for surface mapping.

9.1 Future Work

9.1.1 Contour Recognition

Find a way to compare the shape of two contours that are detected at distinct positions and orientations (yaw) in the image frame. Such functionality may be interesting if returning to the estimated position of an iceberg to look for an object with a specific contour. To avoid that the desired iceberg is detected at this position, a unique identifier (not only the area) may be necessary.

9.1.2 Dynamical Waypoint Generation

Object tracking that dynamically sets new waypoints in CoG of detected objects may be of interest. The drone can follow/track the iceberg for some time, measuring its movements. When enough data is collected about potential movements, a state estimator (i.e. a Kalman Filter) may be used to estimate future positions and movement speeds while the drone progress on its search pattern. For each visit, the Kalman Filter is updated with new measurements. In a practical experiment, such functionality and data may be combined with measured currents, wind directions and satellite images to make fairly accurate assumptions for a larger area.

Another field of application for dynamical waypoint generation can be based on moving the drone through the complete path like the current implementation, before generating a new set of waypoints containing of CoG for previously spotted objects. By contrast, such solution suffers from several limitations. In case an iceberg has moved far, a new search must be initialized. If no data about direction of motion is known, a spiral search may be suitable. Typically, such approach is time consuming and complex.

9.1.3 Path Following for Curved Paths

Time optimal path following for any feasible trajectory is briefly discussed in Section 8.1. For distance minimization and possible obstacle avoidance, such time optimal speed and acceleration plans can be preferable.

9.1.4 Quadcopter Swarm Technology

The quadcopter surface mapping strategy designed in this thesis would be more effective if more than one drone was used. A proposal for future work is to look

into quadcopter swarm technology and implement a prototype using two AR. Drones that maps an area effectively.

9.1.5 Hardware and Platform for Outside Conditions

The first prototype for a surface mapping system is designed. If such system is to be tested in appropriate environments outside, suitable hardware and platform choises is necessary. A literature study on applicable quadcopter configurations, communication systems and safety mechanisms is suggested.

Appendix A

Code Structure

The complete code basis for the implementation is attached to the thesis. As stated, the implementation of the autonomous object mapping strategy is build on the project *cvdrone*, which is a modified version of the official SDK [5]. Simulations are implemented in Matlab Simulink. Organization of the code structure is given in this appendix to make it easier to navigate.

A.1 Surface Mapping Strategy Implemented in C++

The root directiory includes two directories: *C++ implementation* and *Matlab simulation, testing and plotting*. The first includes the following directories of relevance:

- **src**: This is where all the scource code is located.
- **build**: The *make* file that needs to be build is located here. The executable output file will be generated when the make file is successfully run.
- **opencv-2.4.6.1**: This is a complete installation of the C++ library OpenCV. It includes a large number of algorithms used in image recognition applications.

- **licenses**: Includes various license files for used libraries.
- **ffmpeg**: Cross-platform solution to record, convert and stream audio and video.

The organization of the *src* directory is explained next:

- **main.cpp**: File containig most functions implemented in this project. The *main()* function is used to initialize the system and call the other functions used by the program.
- **iceberg.cpp**: Contains class to set data about detected objects for live tracking.
- **listener.cpp**: A UPD listener program to receive data from Labview.
- **.hpp files**: Header files for each .cpp file.

The *src/ardrone* directory contains the following .cpp files with respective .hpp files:

- **ardrone.cpp**: Initialize and finalize the AR. Drone
- **command.cpp**: Functions for initializing AT commands, send takeoff/land commands, activate emergency stop and functions for moving the drone
- **video.cpp**: Initialize video stream, get images from drone camera and finalize video
- **udp.cpp**: Initialize socket, send and receive data, and finalize the socket
- **navdata.cpp**: Initialize navdata, create thread, get various navigation data and finalize navdata
- **config.cpp**: Get configuration of AR. Done and parse a configuration string

A.2 Simulations Implemented in Matlab Simulink

Several modules and functions are tested and simulated in Matlab or Simulink. Fils worth mentioning are

- *Path_following.mdl* is the implemented (simplified) position and velocity controller that takes waypoints as references and outputs positions. Both waypoint generation and the position controller are tested.
- *positionPlot.m* plots measured positions together with a reference.
- *icebergPlot.m* plots the icemerg map, includning contours and drone position.
- *backprojection.m* is written for testing of the back-projection method.
- Several other files are used for plotting and testing. See attached code for details.

icebergPlot and *positionPlot* are scripts for plotting data from the drone together with simulations. Text files are loaded into the Matlab session as vectors, and used for plotting.

Appendix B

Drone Camera Calibration

The bottom camera onboard the AR. Drone must be calibrated in order to be able to detect desired objects. The object detection in this project is based on size and color. The color detection will have to be calibrated, and the procedure is given below.

1. Track bars are placed so that maximum ranges for *hue*, *saturation* and *value* are initialised to the range 0-255. The image is now completely white.
2. Adjust *H min* and *H max* so that the gap between them is as small as possible, and the color that is calibrated is still displayed.
3. Make the same adjustments for the S and V part as well. All other colors should now be filtered out, and thus displayed as black pixles.

In addition, if one wants to add roll, pitch and yaw changes in the model for back projection of a 2D pixel point to world coordinates, the camera matrix must be found. Running a calibration sample using a printed chess board, the following

values were found for K.

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 8.0618 * 10^2 & 0 & 3.1936 * 10^2 \\ 0 & 8.0304 * 10^2 & 1.8492 * 10^2 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

Appendix C

Simulink Model

The position controller and waypoint generator is implemented in Matlab Simulink. Figure C.1 shows the block diagram. See attached code for details on each module.

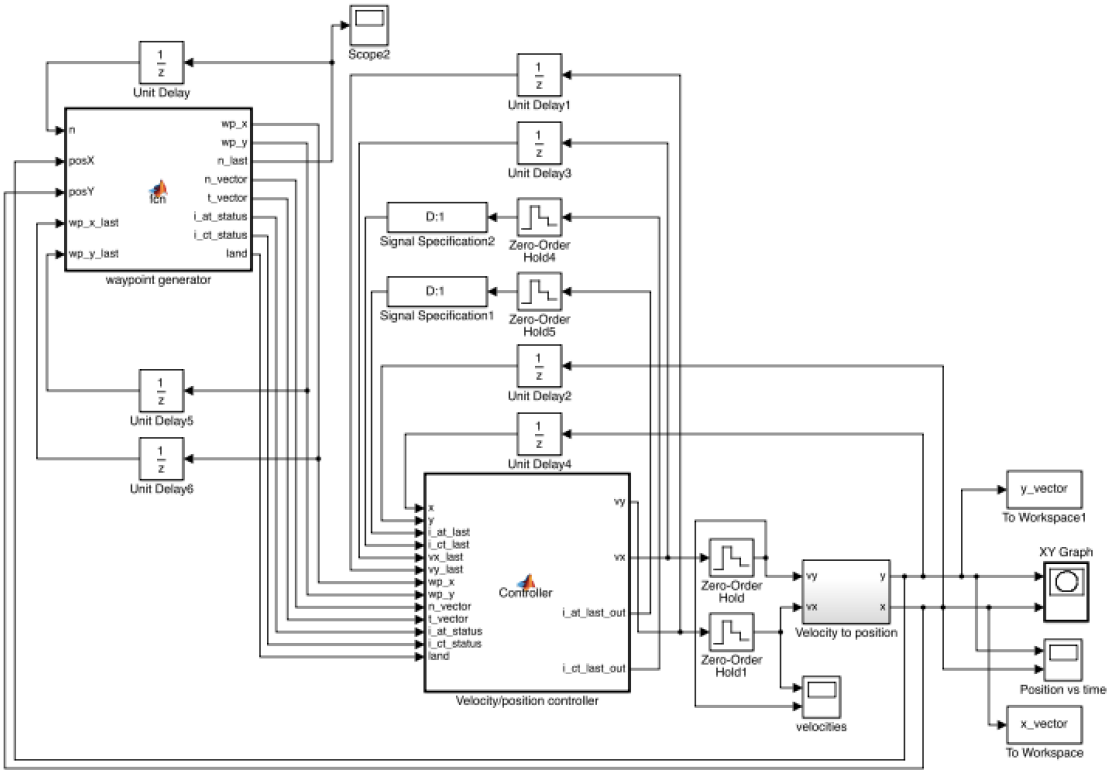

FIGURE C.1: Simulink model for testing of GNC.

Zero-order hold blocks are added to the model to make the measurements and controller gains be sampled at the same rate as the navdata is sent from the drone

and the *AT commands* are sent to the drone. This rate is at $30Hz$. Simulated X and Y positions are stored in vectors to be plotted together with measurements.

Appendix D

Position, Velocity and Orientation from Camera Setup in Snake Lab

The camera setup in the Sintef Snake Lab is compatible with OptiTrack and its related software. Position and orientation are estimated in the Tracking Tools software running on a Windows computer, but to access the data in the Linux C++ program, desired data must be streamed from one computer to another. This Appendix goes into details on the process of setting up the motion capture system and sendning estimated navigation data over UDP from a LabView streaming software. This software utilizes defined trackables in the OptiTrack camera system. Program files are not attached to the thesis at hand, but can be made available from the Department of Engineering Cybernetics at NTNU. The following is written as a guide to get started using the motion capture system for the purpose of reading navigation data of any robot in the lab.

D.1 Camera Calibration in Tracking Tools Software

Tracking Tools is a 3D marker and six degrees of freedom real time motion tracking software for rigid bodies. The 16 cameras mounted in the ceiling streams

video to Tracking Tools for image analysis. Images from several or all cameras are combined with complex algorithms into a 3D space for a trackable object detected by markers. To be able to create such capture volume, the cameras need to be calibrated according to eachother. Details on Tracking Tools and camera calibration can be found in the User's Guide [23].

1. Make sure you have the *OptiWand* with three reflective markers and a trackable marker that can be attached to the vehicle.
2. Launch Tracking Tools Software and turn the cameras on with the remote power button.
3. Go to *3-Marker Wand Calibration*.
4. Go through all five main sections in the calibration pane settings. Details on each section can be found in [23].
5. Start wanding. Make sure to cover as much space as possible for optimal calibration results. Remove any marker or object within the capture volume. Move the wand smoothly for about 5-15 minutes.
6. When sufficient data has been collected, press *calculate*. 3D triangulation and optimization algorithms are applied to the data.
7. When calculations are finished, press *Apply Result*.
8. For optimal quality, press *Apply and Refine* when prompted.
9. Save the file, put your ground plane to your desired origin and click *Set Ground Plane*.
10. Your calibration result can now be used in the LabView streaming program.

Note that the capture volume range depends on the camera calibration. If the cameras are set up for capturing movement on the floor they might not be able to verify position of an object close to the ceiling.

D.2 Initialize Trackable Object in Tracking Tools

When the camera system is successfully calibrated, a fixed (world frame) coordinate system is defined. Next, a trackable object must be defined and its orientation must be initialized.

1. Launch the Tracking Tools Software.
2. Open an existing Tracking Tools project. This is the calibration project generated at the calibration process. **Please do not update the Tracking Tools software, as a new calibration will be required**.
3. Place a trackable marker somewhere inside the capture volume. This marker should be visible in the frame.
4. If you want to see real time coordinates, select all the points of the object detected by the cameras, right click on the object and choose properties. In the *Trackables* panel you can now see properties of the object. By choosing *Real-Time Info* you can see the position and orientation.
5. Next, you probably want to define (and initialize) a new trackable object. Select all the visible points, go to the *Trackables* panel and press *Create From Visible*. To change default values for orientation, go to the *Orientation* tab.
6. Now, trackables, calibration and the project file must be saved. Save the files in the following order:
   (a) Save Trackables
   (b) Save Timeline Recording
   (c) Save Camera Calibration
   (d) Save Project

The project file can be opened in a LabView program that will be used to retrieve and stream navigation data.

D.3 Stream Navigation Data over UDP from LabView

For the navigation data to be used as input in another program, it will be streamed over network. LabView is used to stream data over UDP.

1. Run the *Position Tracker* LabView project.
2. Open *Position tracker.vi*.
3. Specify the path to the generated Tracking Tools project file.
4. Connect remote computer that will receive streamed data to a wired network in the lab.
5. Go to *Window - Show Block Diagram*. The structure of the program is presented. See figure D.1.
6. Double click on *Send position data to remote computer* and define the IPv4 address of the remote computer. Also specify the remote UDP port (the port to be read in the remote program) and the local port to be used.
7. In case you want to write measurements to a .txt-file, open *Write measurement data to file* to set the path for the output file.
8. To display real time measurements, click *Plot measurement*.
9. Sending a data package over UDP is the main module of the program, which is presented in figure D.2.
10. When ready to stream real-time measurements, click *Run* in the top left corner of the main window. Displaying plots of measurements while streaming may cause an extra delay in the package sending loop.
11. To stop measurements press the *Stop* button in the main project window. Aborting execution by using LabView GUI may cause errors.

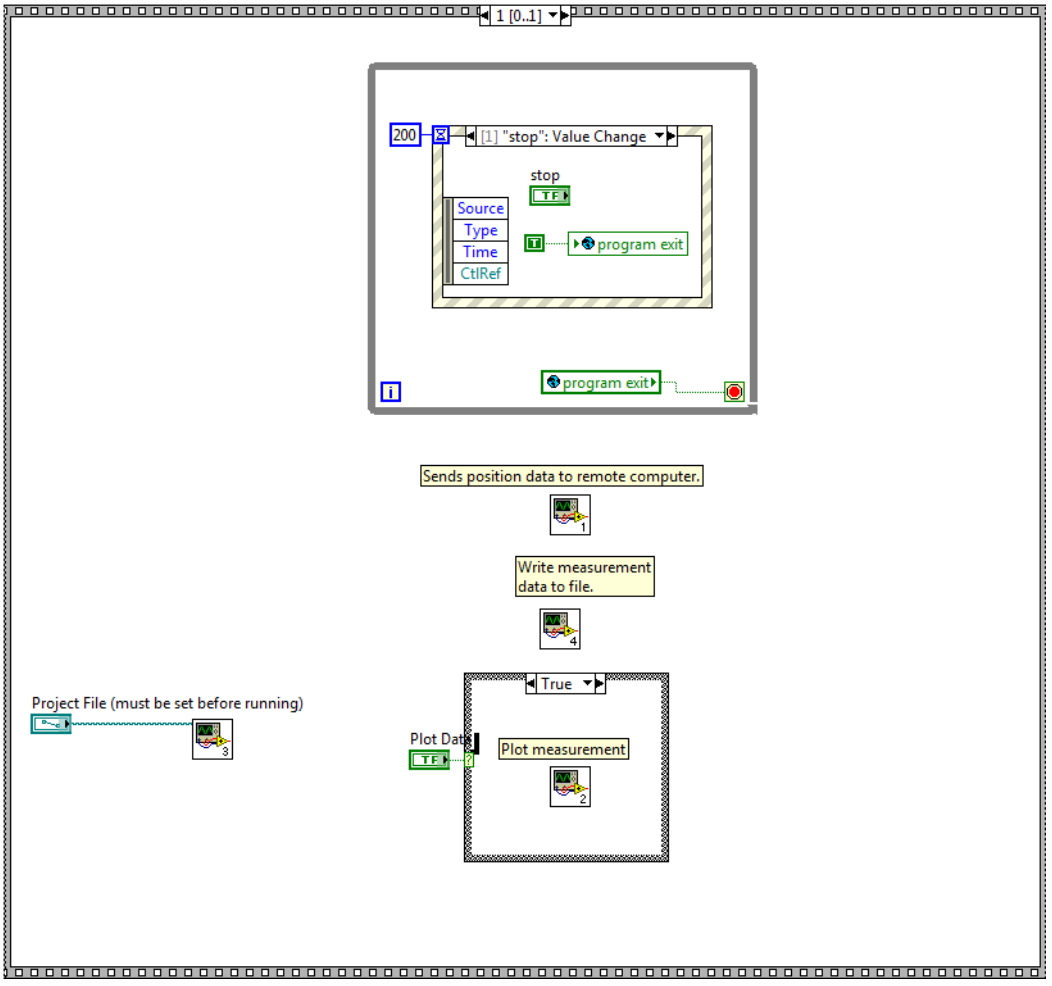

FIGURE D.1: Overall block diagram for LabView streaming program.

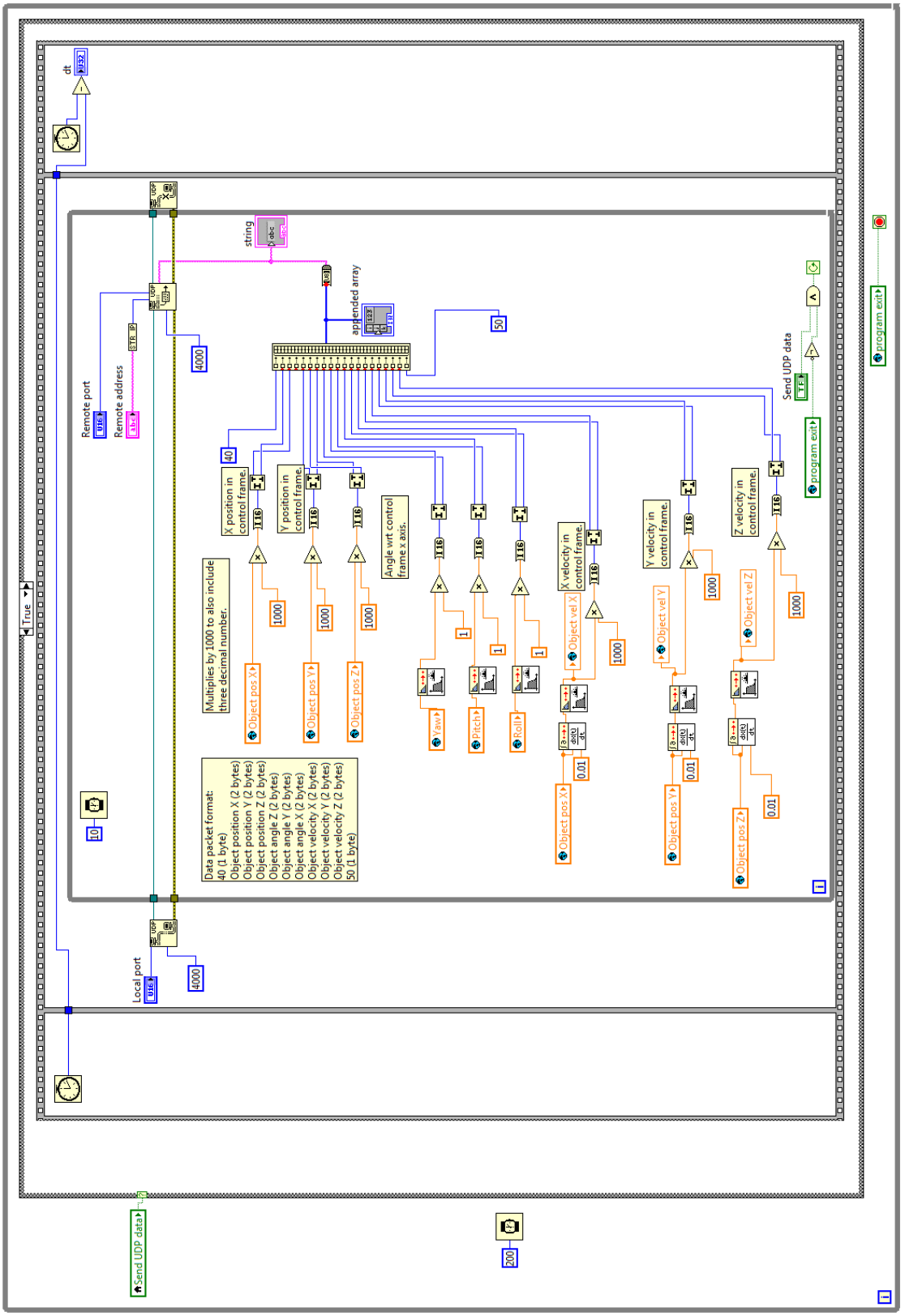

FIGURE D.2: Block diagram of LabView UDP streaming module. A total of 20 bytes are sent in one package. Each data value given in 2 bytes.

Appendix E

System Setup on a Linux Computer

To get the code for the overall surface mapping strategy and position controller up and running on a camputer, a couple of pre requirements must be set up. A working version of the system has the following specifications and requirements:

- AR. Drone version 2.0.
- Optitrack camera system in Sintef Snake Lab.
- Windows computer running LabView tracking software for the OptiTrack system.
- A remote computer (laptop) running Ubuntu 12.04 LTS 64 bit.
- g++ (C++ compiler) and OpenCV installed.
- The remote computer has the latest version of the code basis written for this thesis.

Details on setup of the surface mapping strategy code basis are given below to make the reader get the qaudcopter in air without too much trial and error.

E.1 Install g++

The C++ code basis is written and compiled for a Linux system. Step number one is to install a C++ compiler:

```
$ sudo apt-get install g++
```

This compiler can be used from the terminal, i.e. compiling the *makefile* for the project.

E.2 Install OpenCV and Related Requirements

OpenCV is the computer vision library used in this project. It has to be installed on the remote computer running the program in order for used OpenCV algorithms to run. First, make sure GCC is installed:

```
$ sudo apt-get install build-essential
```

Next, we will install Git (version control system)

```
$ sudo apt-get install git
```

Clone the existing Git repository into your desired folder

```
$ cd <your_working_directory>
$ git clone https://github.com/Itseez/opencv.git
```

Install the following requirements

```
$ sudo apt-get install libopencv-dev
$ sudo apt-get install libhighgui-dev
$ sudo apt-get install libcv-dev
$ sudo apt-get install ffmpeg
```

Please have a look at the OpenCV documentation[1] for more details, or if you need to build OpenCV from source.

E.3 Get the Quadcopter Code

The code is written with the help of the *cvdrone* library[17]. All rights are reserved to the author according to the license[2]. The complete code basis included surface mapping and path following is attached to the thesis at hand.

Make sure all requirements are installed before compiling:

```
$ cd <your_path>/ardrone/cvdrone/build/linux
$ make
```

If everything is set up correctly, included linking to OpenCV files, the file is successfully compiled and you can run the test by typing

```
$ ./test.a
```

Please have a look at the attached video for live examples.

[1] http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html
[2] https://github.com/puku0x/cvdrone

Bibliography

[1] Twan Maintz. *Digital and Medical Image Processing*. Universiteit Utrecht, 2005.

[2] Håvard Håkon Raaen. Software Infrastructure for a UAV Testbed For Ice Management Guidance and Estimation Algorithms. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2013.

[3] Ylva Stokke Vintervold. Camera-Based Integrated Indoor Positioning. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2013.

[4] Gabriel M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Quadrotor Helicopter Trajectory Tracking Control. Stanford University, California, 2011.

[5] ArDrone projects. Ardrone developer guide. `https://projects.ardrone.org/attachments/download/365/ARDrone_SDK_1_7_Developer_Guide.pdf`, October 2013.

[6] ARDrone-flyers. Ar.drone specs. `http://www.ardrone-flyers.com/ar-drone-specs.html`, October 2013.

[7] Nick Dijkshoorn. Simultaneous localization and mapping with the ar.drone. Master's thesis, Universitet van Amsterdam, 2012.

[8] David Vissière Nicolas Petit Pierre-Jean Bristeau, François Callou. The Navigation and Control technology inside the AR. Drone micro UAV. 2011.

[9] Bjørnar Vik. *Integrated Satellite and Inertial Navigation Systems*. Department of Engineering Cybernetics, NTNU, 2012. ch. 1, 2, 5.

[10] Nathan Michael Daniel Mellinger and Vijay Kumar. Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors. Sage on behalf of Multimedia Archives, 2012. doi: 10.1177/0278364911434236.

[11] Kenneth Eide Haugen. Surface Mapping using Quadcopter. Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2013.

[12] Markus Hehn and Raffaello D'Andrea. Quadrocopter trajectory generation and control. In *IFAC World Congress*, volume 18, pages 1485–1491, 2011.

[13] Richard Hartley & Andrew Zisserman. *Multiple View Geometry in computer vision*. Cambridge University Press, 2003. cha. 6, 8.

[14] Yannick Morvan. Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video. Eindhoven University of Technology, 2009.

[15] Bastian Venthur. python-ardrone. `https://github.com/venthur/python-ardrone`, November 2013.

[16] Javadrone. Ar.drone java api. `http://code.google.com/p/javadrone/`, November 2013.

[17] puku0x Github. cvdrone. `https://github.com/puku0x/cvdrone`, January 2014.

[18] T. John Koo and Shankar Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *IN PROCEEDINGS OF THE 37TH CONFERENCE ON DECISION AND CONTROL*, pages 3635–3640, 1998.

[19] E. Frazzoli, M.A. Dahleh, and E. Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American*

Control Conference, 2000. Proceedings of the 2000, volume 6, pages 4102–4107 vol.6, 2000. doi: 10.1109/ACC.2000.876993.

[20] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron. Flight test and simulation results for an autonomous aerobatic helicopter. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 2, pages 8C3–1–8C3–6 vol.2, 2002. doi: 10.1109/DASC.2002.1052943.

[21] Zhe Jiang, Jianda Han, Yuechao Wang, and Qi Song. Enhanced lqr control for unmanned helicopter in hover. In *Systems and Control in Aerospace and Astronautics, 2006. ISSCAA 2006. 1st International Symposium on*, pages 6 pp.–1443, Jan 2006. doi: 10.1109/ISSCAA.2006.1627508.

[22] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. A prototype of an autonomous controller for a quadrotor uav. In *European Control Conference*, pages 1–8, 2007.

[23] NaturalPoint Corporation. Tracking tools 2.4.0 user's guide. NaturalPoint Inc. DBA OptiTrack, 2012.