NTNU – Trondheim
Norwegian University of
Science and Technology

Experimental Monitoring of Sea Ice Using Unmanned Aerial Systems

Andreas L. Flåten

Master of Science in Cybernetics and Robotics
Submission date: June 2015
Supervisor: Lars Imsland, ITK
Co-supervisor: Anders Albert, ITK
Frederik Stendahl Leira, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Task description

Background

As offshore oil- and gas exploration and production enters arctic seas, the presence of ice becomes a substantial challenge which requires monitoring and active countermeasures. In some areas, monitoring of icebergs is expected to be an important activity, which today in many cases is done using manned planes and helicopters. The use of unmanned aerial vehicles (UAVs) could arguably give a substantial cost reduction in such tasks. AMOS is planning a trip to Svalbard in the spring of 2015, where one aim to do initial tests for a (partly, prototype) autonomous UAV system for monitoring of icebergs. This project will develop/implement/test technology for doing such experiments.

Work description

- Give a brief overview of the role of ice estimation systems in Arctic operations.
- Describe or make suggestion for the components and procedures of a UAS operation for iceberg monitoring
- Describe the platform (UAV, hardware, software, interfaces, etc.) where the monitoring system should be implemented
- Do appropriate tests, make illustrative simulations/experiments, and discuss.

Preface

This thesis is my final work of a five year Engineering Cybernetics program at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor Lars Imsland for his guidance and the opportunity to work on an interesting and challenging project. I would also like to thank co-supervisors Anders Albert and Frederik Leira for collaboration and guidance during the thesis work.

Finally I would like to thank my fellow students in GG44 for providing a great social and working environment, and Mathilde for all her love and support.

Andreas Lindahl Flåten,
Trondheim, June 2015

Abstract

The exploration of arctic seas for offshore oil- and gas resources has received increasing interest the past few years. Despite the recent dramatic fall in oil prices, estimates indicate that as much as 22% of the worlds remaining hydrocarbons are located in arctic areas. Thus it is unlikely that the arctic areas will go largely untouched the following decades. One of the main challenges of extracting hydrocarbons in arctic areas is the abundance of sea ice that can cause damaging loads on installations. An important part of oil exploration in these areas is thus the ability to manage potentially damaging sea ice. The current methods for ice management include manned helicopters and other aircraft for detection together with ships to break up or drag away dangerous ice. The main objective of this thesis is to assess the use of Unmanned Aerial Systems (UAS) to perform ice monitoring. An autonomous Unmanned Aerial System for ice detection and mapping using a thermal imaging sensor on a small fixed wing aircraft is proposed. The main contributions of this thesis is a real-time Bayesian recursive algorithm for occupancy grid map estimation representing sea ice. An expedition to Svalbard with several PhD and master students from NTNU was originally planned in April 2015, but this was canceled in March due to time constraints among the participants. The expedition was a major source of inspiration for the methods developed, and an indoor laboratory environment for on-board computer vision was developed using the Robot Operating System (ROS) software framework. The setup included a quadcopter with an on-board camera, and a motion capture system capable of tracking the pose of the quadcopter at 120 Hz. The laboratory setup was used to test much of the planned functionality for the Svalbard expedition. The developed computer vision based map estimation algorithm is capable of running in real time on an on-board computer. As a part of the preparation for the Svalbard excursion, a path planning framework developed by PhD student Anders Albert was successfully tested in the laboratory setup. The experimental results of the mapping algorithm were visually appealing, but closer investigation revealed unsatisfactory accuracy. Using on-board navigational systems alone to perform real-time mapping did not yield sufficient accuracy for practical use. Sources of error and means to improve the results in further work were investigated.

Sammendrag

Utforskning av arktiske områder i forbindelse med olje- og gassvirksomhet har vært stadig mer interessant de siste årene. Selv om oljeprisen har hatt en dramatisk nedgang den siste tiden, har det blitt gjort studier som tyder på at så mye som 22% av de resterende utvinnbare hydrokarbonene befinner seg i arktiske havområder. Det er derfor sannsynlig at aktiviteten i disse områdene bare vil fortsette å øke. En stor utfordring med utvinning av olje og gass i disse områdene er forekomsten av store mengder havis som kan gjøre stor skade på installasjoner. Håndtering av potensielt skadelig havis er derfor en viktig del av arktisk maritim virksomhet. Nåværende metoder for havishåndtering gjøres med blant annet helikopter for monitorering og bemannede havfartøyer for uskadeliggjøring av havis. Hovedformålet med denne oppgaven er å utrede bruk av et ubemannet fly med et infrarødt kamera til ismonitorering. Hovedbidraget i denne oppgaven er et Bayesisk rekursivt filter for estimering av et rutekart som representerer havis i sanntid. En ekspedisjon til Svalbard sammen med PhD- og masterstudenter fra NTNU var planlagt i april og inspirerte mye av arbeidet i denne oppgaven, men denne ble avlyst i mars på grunn av mangel på tid til forberedelser blant deltagerne. Et innendørs laboratorieoppsett ble utviklet i Robot Operating System (ROS) for å teste et prototype autonomt ismonitoreringssystem. Som del av forberedelsene til Svalbard ekspedisjonen ble kartleggingsalgoritmen testet nøye i laboratoriet. En baneplanleggingsalgoritme som skulle arbeide sammen med iskartleggingsalgoritmen utviklet av PhD student Anders Albert ble også testet. Resultatene viste at å utføre kartlegging i sanntid med kun navigasjonsutstyr som befant seg ombord i det ubemannede flyet ikke ga god nok nøyaktighet. Fremtidig arbeid bør undersøke bruk av bildedata direkte for å forbedre nøyaktigheten i kartleggingen.

Contents

Glossary

6-DOF Six Degrees Of Freedom.

AMOS Centre for Autonomous Marine Operations and Systems.

API Application Programming Interface.

BLOS Beyond Line Of Sight.

COTS Commercial Off-The-Shelf.

DCM Direction Cosine Matrix.

DUNE Dune is an Unified Navigation Environment.

ECEF Earth Centered Earth Fixed.

EKF Extended Kalman Filter.

NED North-East-Down.

NORUT Northern Research Institute.

OpenCV OpenCV - Open Source Computer Vision.

ROS Robot Operating System.

SfM Structure from Motion.

SLAM Simultaneous Localization And Mapping.

UAS Unmanned Aerial System.

UAV Unmanned Aerial Vehicle.

Chapter 1

Introduction

The main motivation for this project is to assess the use of an UAS (Unmanned Aerial System) to aid ice management. Ice management is a necessary part of carrying out operations in seas where either drifting icebergs, sea ice or ice ridges are present. The reason for ice management being a vital part of operations in these areas is that drifting ice can incur significant loads on ocean structures. These loads can lead to potentially catastrophic failures risking human life and the environment. The definition of ice management may vary according to where operations are carried out, in this project the following definition proposed in Eik 2008 will be used:

> *Ice management is the sum of all activities where the objective is to reduce or avoid actions from any kind of ice features. This will include, but is not limited to:*
>
> - *Detection, tracking and forecasting of sea ice, ice ridges and icebergs*
> - *Threat evaluation*
> - *Physical ice management such as ice breaking and iceberg towing*
> - *Procedures for disconnection of offshore structures applied in search for or production of hydrocarbons*

Even though the interest and use of UAS has increased greatly the last ten years in academia, industry and recreational use, humans are usually a central part of an operation using such systems. Thus together with the exploration of the technological capabilities of an autonomous ice monitoring system, the interaction with human decision makers and operators is an important part to consider. As time is a very limited resource when working on this thesis, it is important to narrow down the scope as much as possible. When the scope of the work has been limited, it is important to spend some time researching previous work so as to not "reinvent

the wheel" and to use existing knowledge in an efficient manner. With this in mind, it is useful to identify which challenges in a system that can be regarded as similar to other fields and how such a system best can aid ice management operations.

1.1 Detection and mapping of sea ice features

The most prominent role of UAS in ice management in modern operations could arguably be in the detection and tracking of potentially dangerous ice. This could be both in both surveying relatively remote areas far from arctic operations, and real time detection and tracking close to operations. The AMOS (Centre for Autonomous Marine Operations and Systems) had planned a trip to Svalbard during the spring in 2015, where one of the objectives was to test a prototype autonomous UAV system for ice monitoring. The main sensor planned for ice detection was an infrared camera mounted in a stabilized frame. Thermal images from previous surveys in waters with prominent sea ice features revealed that detecting these features was possible. To the authors knowledge, sea ice monitoring from aircraft has so far been done with manned fixed wing aircraft as reported in Eik 2008. There are several interesting challenges when it comes to performing ice monitoring autonomously, and many questions arise when trying to imagine such a system;

- What value can such a system add to ice management operations?
- What kind of accuracy can be expected?
- What is the overall system complexity?
- What degree of autonomy can be expected from the system?
- How robust is the system with respect to e.g. changing weather conditions?
- What kind of area coverage can be expected?

the aim of this thesis is to at least partially answer these questions by describing and implementing an experimental system to detect and map ice features with an autonomous UAS using probabilistic methods.

1.2 Previous work in tracking and mapping

The general problem of tracking multiple moving objects has been an active area of research for many years already. Usually objects are tracked on a stationary platform on a fixed and known location relative to the earth. A natural generalization to this is tracking of objects on a moving platform. Research in SLAM (Simultaneous Localization And Mapping) usually focuses on the tracking and mapping of static features relative to a moving robot, but can be extended to include moving objects for e.g. collision avoidance (Urmson et al. 2009). SLAM on small unmanned aerial platforms using visual sensors have recently been shown to perform well in outdoor applications (Forster, Pizzoli, and Scaramuzza 2014; Stephan Weiss

et al. 2013). Modern methods are often probabilistic, and in essence they try to determine the joint probability distribution of the state of all objects of interest, including the state of the sensor platform. Estimating this density function naively is however not feasible due to an exponential growth of the number of states that need to be tracked. In practice objects are usually tracked independently, introducing the challenge of associating measurements with the relevant objects. See Schulz et al. 2001 for a popular probabilistic method of associating measurements with objects of interest. A simpler problem will be considered in this thesis, it will be assumed that the state of the moving sensor platform is known perfectly, and that the objective is to estimate a world state, in particular ice density at sea level.

1.3 Computer vision on mobile robots

Using a camera as a sensor on a mobile robot has been an active area of research for roughly two decades already. In robotics cameras are used most commonly in applications like navigation and mapping. A well known problem in robotics is the SLAM problem, where cameras are often used as an important sensor. A notable and well known experimental application is the use of a stereo camera system on NASA Mars rovers for obstacle avoidance and navigation (Goldberg, Maimone, and Matthies 2002). Cameras as sensors have also made their way into commercial automotive applications for applications like partially or fully autonomous driving, see e.g. lane-following in Sadano, Kawazoe, and Shimakage 2002 or driver assistance systems in Urmson et al. 2009. A lot of the theory and methods required to use cameras as a practical sensor started development already in the 50s and 60s in the photogrammetry community, with applications like highly accurate terrestrial mapping. Modern applications of traditional photogrammetry have enabled estimation of highly accurate 3D world models used in commercial large scale surveying applications. A familiar example is the application of aerial imagery to 3D city modeling in Google Earth. A modern research topic relevant for this thesis is SfM (Structure from Motion) in the computer vision community. The aim of SfM techniques is to infer three-dimensional structures from image sequences. Successful applications include 3D object and terrain modeling, just like in photogrammetry. The theory and algorithms required in vision-based SLAM, photogrammetry and SfM problems are very similar, and there is a lot of overlap in the results and research done in the fields of vision in robotics, general computer vision and photogrammetry. It should be noted that recent advances in SLAM have often included some sort of active range sensor in addition to or instead of just a camera. Examples include LIDAR (Laser Illuminated Detection and Ranging) sensors, RADAR, ultrasound ranging sensors and so called RGB-D(epth) sensors like the Microsoft Kinect. Laser based ranging sensors have proved very effective in autonomous driving research (Levinson, Montemerlo, and Thrun 2008), where both range and intensity data can be used to navigate safely in urban environments. An interesting comparison of photogrammetric methods and the early use of airborne laser scanning in the remote sensing community can be found in Baltsavias 1999. Advantages of laser scanners include extremely high spatial resolution, depth information and illumina-

tion independence (lasers are active sensors). Advantages of using passive imaging sensors are increased spectral resolution compared to laser scanners, availability of low cost and low footprint COTS (Commercial Off-The-Shelf) sensors and to a large extent illumination independence in the case of thermal imaging sensors. It is to the knowledge of the author at the moment both cost and weight prohibitive to use laser scanners in experimental applications such as the one presented in this thesis. Many of the mentioned methods are reaching maturity, but are still mostly on a research level. Since the authors own experience in these research fields is very limited, only the simplest of methods are implemented and tested in this thesis.

1.4 Structure of this thesis

The first chapter in this thesis presents some of the most relevant theory for the project work. An algorithm for ice density estimation is presented in the same chapter. It is assumed that the reader has basic knowledge of statistics, unmanned aerial systems, control systems, linear algebra and computer vision. The following chapter presents a proposed system architecture for an ice monitoring system. This includes a description of an indoor laboratory environment that was developed to test the necessary parts of the proposed ice monitoring system. The next chapter consists of some experiments done in the indoor laboratory and on some other data relevant to this thesis. The final chapters are a discussion of the experiments done and a conclusion with some remarks regarding further work. Some code developed for this thesis is included in the appendix for illustration purposes, the rest of the implementation necessary to reproduce the results of this thesis can be found in the accompanying zip file.

Chapter 2

Theory

2.1 Notation and coordinate transformations

This section briefly describes notational conventions and coordinate transformations used in this thesis. The notation and definitions used in this thesis loosely follow those found in Egeland and Gravdahl 2002. Vectors are written in bold font and a superscript represents the reference frame which it is expressed.

$$\boldsymbol{p}^w = \begin{bmatrix} x^w \\ y^w \\ z^w \end{bmatrix} = \begin{bmatrix} p_1^w \\ p_2^w \\ p_3^w \end{bmatrix} \tag{2.1}$$

where $\boldsymbol{p}$ is a vector expressed in the w reference frame. A vector written in homogeneous coordinates is indicated by a $\sim$ symbol.

$$\widetilde{\boldsymbol{p}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

A coordinate free vector, i.e. a vector without any reference frame, is written with an arrow: $\vec{p}$. Different coordinate frames are used for guidance, navigation, control and computer vision algorithms. In motion control and robotics applications in general, it is common to have to transform vectors from one coordinate frame to another. Given two reference frames a and b, a transformation between the frames can compactly be described by the following homogeneous transformation matrix

$$\boldsymbol{T}_b^a = \begin{bmatrix} \boldsymbol{R}_b^a & \boldsymbol{r}_{ab}^a \\ \boldsymbol{0}^T & 1 \end{bmatrix} \tag{2.3}$$

Where $\boldsymbol{R}_b^a$ is the rotation matrix from frame a to frame b. The columns of $\boldsymbol{R}_b^a$ are the three unit vectors of b expressed in a. $\boldsymbol{r}_{ab}^a$ is a vector describing the translation of the origin of b relative to a, expressed in frame a. A homogeneous vector expressed

in frame b, denoted $\widetilde{\boldsymbol{x}}^b$ can now be expressed in frame a by the following matrix multiplication

$$\widetilde{\boldsymbol{x}}^a = \boldsymbol{T}_b^a \widetilde{\boldsymbol{x}}^b \tag{2.4}$$

It is often convenient to represent the rotation matrix (also called the DCM (Direction Cosine Matrix)) with fewer parameters. It is a well known fact that any rotation matrix can be expressed by three successive rotations about a fixed axis. This leads to the definition of the Euler angles. The Euler angles are minimal in the sense that they define a rotation uniquely with the fewest amount of parameters. It is however ambiguous if the rotation sequence and axes are not specified. Another disadvantage of the Euler angles is that the kinematic equations (time derivative) always have singular points. Another useful fact about rotations is that any fixed rotation can be specified as a rotation about an axis (i.e. a direction) and an angle. The angle-axis representation has four parameters; namely the angle $\theta \in \mathbb{R}$ and the axis $\boldsymbol{k} \in \mathbb{R}^3$. A rotation matrix can be computed from the angle-axis parametrization in the following way

$$\boldsymbol{R}_b^a = \cos\theta \boldsymbol{I} + \sin\theta \left(\boldsymbol{k}^a\right)_x + (1 - \cos\theta) \boldsymbol{k}^a \left(\boldsymbol{k}^a\right)^T \tag{2.5}$$

where the axis is specified in frame a and $(\cdot)_x : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ is the skew symmetric cross product matrix operator defined as:

$$\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix} \right)_x = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix} \tag{2.6}$$

Finally, the angle-axis parametrization can be related to the mathematically convenient *unit quaternion* through the following definition

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \boldsymbol{k}\sin\frac{\theta}{2} \end{bmatrix} \tag{2.7}$$

The quaternion can be interpreted as an extension of the complex numbers, where η represents the scalar real part and the vector $\boldsymbol{\epsilon}$ represents three imaginary units. The order which the parameters are represented in the quaternion is specified varies. The two most common conventions is to either have the real part as the first or last component in the quaternion. The latter is the convention used in ROS (Robot Operating System). The unit quaternion can be used to represent a rotation or a change of basis, just like the DCM. The main advantages of using quaternions are the efficiency in composing rotations, the numerical stability and the lack of singular points in the kinematic equations. Considering the quaternion as an extension of the complex numbers, an alternative way of computing the quaternion from the axis-angle parameters is through an analogue of the Euler formula:

$$\vec{q} = \exp\left(\frac{\theta}{2}\left(k_1\vec{i} + k_2\vec{j} + k_3\vec{k}\right)\right) = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\left(k_1\vec{i} + k_2\vec{j} + k_3\vec{k}\right) \tag{2.8}$$

2.2 UAV model

In this project, the goal is to use a fixed wing UAV (Unmanned Aerial Vehicle) as a vital component to perform experimental ice monitoring. An UAV can provide a mobile sensor platform that can be moved to a virtually arbitrary position in space. The platform does however not come without constraints. The constraints are both dynamic (e.g. Newtons laws via differential equations) and static (e.g. maximum bank angle, maximum thrust, minimum speed). A sufficiently detailed model of the flying UAV platform will be necessary in the development of the system. At the lowest level of abstraction, the motion of an UAV can be accurately modeled with a set of coupled nonlinear differential equations. For the purposes of this thesis however, which is the use of an UAV to position an imaging sensor in space, a model of this fidelity is not necessary. The aircraft which planned for the experimental platform to be used on Svalbard is a "Skywalkr X8" with an "Ardupilot" closed loop autopilot. At the highest level, the autopilot has the capability to follow paths in space in the form of an ordered set of waypoints autonomously. For the purposes of this thesis, this description is detailed enough.

2.3 Camera model

In the UAS intended for the prototype ice monitoring, an infrared camera is mounted in the belly of an X8 UAV in a stabilized gimbal frame. In order to relate features in captured images to the world, a model of the projection of 3D world points to a 2D camera plane is needed. A projection model usually includes several parameters grouped into extrinsic and intrinsic parameters. It is necessary to know both the intrinsic camera parameters defined by the cameras optical system and extrinsic parameters that relate the camera to the world to infer information about where a feature is located. The determination of the mapping between 3D reference coordinates and 2D camera coordinates is often called camera calibration. Procedures for the determination of this mapping are described in Zhang 2000; Heikkila and Silven 1997, and has inspired the implementation of numerical calibration methods in OpenCV (OpenCV - Open Source Computer Vision) and MATLAB. Having a good estimate of this mapping is crucial for computer vision applications relating image features to the physical world, such as ice features at sea. The projection models used in this thesis are identical to those used in OpenCV, mainly since it is used as an implementation framework and the models are fairly standard. Performing calibration of intrinsic parameters is usually done by capturing images of features that have a known structure a fixed reference frame. When using a traditional optical camera, it is common to print out a checkerboard pattern and capture images at several viewpoints. The pattern can be recognized through common computer vision algorithms, and both extrinsic and intrinsic camera parameters can be determined for the captured images. The extrinsic parameters estimated are not really useful for further work, since they only describe the camera pose when the calibration images are taken. The intrinsic parameters are however very useful, and can be assumed constant for a constant

focal length and image size. The parameters can be scaled to accommodate for changes in image size (i.e. image resolution). Since an infrared camera is used in this application, the calibration images have to be captured in a somewhat unusual way. The method used at the UAV lab is to capture images of a cooled metal plate with several evenly spaced holes. It will in this thesis be assumed that the camera intrinsic parameters can be calculated once and off-line in a laboratory environment, and that extrinsic parameters can be calculated by relating estimates of the UAV position and attitude to the camera optical frame. It is also possible to estimate extrinsic parameters from captured images during flight, i.e. using the camera as a navigation tool. Methods for camera aided navigation or more generally SLAM is not the topic of this thesis, but are discussed in Chapter 5.

2.3.1 Extrinsic parameters

Extrinsic parameters are the location and orientation of the camera in space with respect to a earth fixed reference frame as in (2.3). Given that a point in space is represented by a homogeneous vector in a world fixed coordinate frame w (e.g. a world fixed NED (North-East-Down) frame) by $\boldsymbol{p}^w$, we can express this point in a camera fixed frame c by the homogeneous transformation

$$\widetilde{\boldsymbol{p}}^c = \begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix} = \boldsymbol{T}_w^c \widetilde{\boldsymbol{p}}^w \tag{2.9}$$

It will be assumed in this thesis that the camera is rigidly mounted to the body of the UAV, such that the relative orientation of the camera and the body frame can be represented by a simple translation of the origin and a static rotation matrix. If the position and orientation of the UAV body with respect to a fixed frame is known, the camera extrinsic parameters can thus be calculated.

2.3.2 Intrinsic parameters

The cameras intrinsic parameters describe how points in a camera fixed frame are projected into the cameras optical frame. The model presented here is taken from Hartley and Zisserman 2004. Common intrinsic parameters are focal length, optical center, lens distortion coefficients and pixel scaling factors. The most simple model of image projection is the pinhole camera model. This model is a linear projection model that describes how points expressed in a camera fixed frame are projected onto the camera optical plane. Consider a point (x^c, y^c, z^c) expressed in a camera fixed 3D reference frame. Using the pinhole projection model the point can be expressed in the image plane by:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \frac{f}{z_c} \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{2.10}$$

where f is the cameras effective focal length. The camera optical frame is as usual in computer vision applications centered in the upper left corner of the image array.

The units of the coordinates (u, v) are in pixels, therefore pixel scaling factors s_u and s_v are needed to relate metric units to pixels. In addition, the optical center (u_0, v_0) must be known to arrive at the following relation between the projected point to the camera optical frame

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_u \tilde{u} \\ s_v \tilde{v} \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{2.11}$$

The pinhole model does not account for nonlinear lens distortion effects, but in some applications yields accurate enough results. The model can be extended to include distortion effects, and images can be undistorted to improve the accuracy of the pinhole model. The advantage of the pinhole model is its simplicity and the fact that it gives a linear projection model. Including the camera extrinsic parameters, we can express the pinhole projection model in homogeneous coordinates in the following way:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{A} \boldsymbol{T}_w^c \widetilde{\boldsymbol{p}}^w = \begin{bmatrix} f s_u & 0 & u_0 \\ 0 & f s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_w^c & \boldsymbol{r}_{cw}^c \end{bmatrix} \begin{bmatrix} x^w \\ y^w \\ z^w \\ 1 \end{bmatrix} \tag{2.12}$$

Note that the projection of a 3D point to the 2D image plane is only defined up to the scale factor $s \in \mathbb{R}$. This is a consequence of the camera being a *bearing only* sensor, meaning that points on a line in 3D yield the same projected point in 2D.

2.4 Image segmentation

We can determine if an image pixel represents an ice feature or not by performing image segmentation. Image segmentation assigns image pixels into two or more classes, or more generally groups them into "similar" classes. In the case of this thesis this will at the highest level be either *ice* or *non-ice* pixels. There are many ways of segmenting an image into two or more classes. There are correspondingly many ways to interpret and use the image data at hand to perform the segmentation. Examples include image intensity, detected image edges, motion, color similarity and more. The most trivial is probably through image *thresholding*. This is a simple image intensity based procedure where pixels are assigned to one class if they are above a certain threshold intensity, and to the second class if they are below it. It is also a very efficient method, since it requires only one comparison per pixel. Thresholding can work well if the image has a bimodal histogram with two distinct peaks, separating the two desired classes. Finding the most optimal threshold value might be nontrivial, and it does not need to be constant for a group of images due to e.g. variations in illumination. A simple adaptive method is called *Otsu's criterion* (Otsu 1979), where a global threshold value is selected to divide the image into two maximally homogeneous regions as a function of the image histogram. This method can naturally be extended to compute locally optimal threshold values in an image. Again, this will only work well if the image

is bimodal in its histogram and the interesting features are indeed homogeneously in either one of the histogram regions. A recent survey on thresholding techniques can be found in Sezgin and Sankur 2004. More general methods include graph based optimization methods where image intensity, motion and spatial arrangement can be used as a similarity measure as described in Shi and Malik 2000. An method based on Markov random fields using intensity data for segmenting sea ice in synthetic aperture radar (SAR) data is found in Clausi 2005. Another common method for image segmentation is to use edge detection together with morphology operations. In the laboratory environment used in this project, simple image segmentation using thresholding was sufficient. Since image data for the actual application of sea-ice monitoring is unavailable for the planned experimental platform, there is not much that can be said about which methods will work well on the true application. Images from a similar application were however obtained by courtesy of NORUT (Northern Research Institute), showing aerial imagery of dense sea-ice taken by an infrared camera. The next section shows a short analysis of these images with respect to image segmentation.

2.4.1 Image segmentation of NORUT data

Aerial images of dense sea ice taken with an infrared camera from a UAV was made available by NORUT. Some simple image segmentation procedures were tested on the data to indicate if infrared data can be used to reliably detect ice features. Unfortunately there is no ground-truth available for the images, so that there is no reasonable way to quantitatively evaluate the methods. The open source *scikit-image* python package was used to perform the analysis (Walt et al. 2014). A few different methods were tested, including global and local Otsu thresholding methods, segmentation by edge detection coupled with morphological operations, and segmenting by using the watershed transform. The simplest of these methods is arguably the global Otsu thresholding method. None of the mentioned methods gave any significantly better results then the others (which is hard to evaluate due to the lack of ground-truth), so only the results from the global Otsu method is presented here. The results are shown in Figure 2.1, where an additional image with morphological closing performed on the result of the global Otsu thresholded image in an attempt to reduce noise is also shown.

2.5 Ice density estimation

One of the main objectives of this thesis is to implement a method to estimate the sea ice distribution using an infrared camera. It is necessary to define how to represent and store detected ice features. It is not the aim to implement the most sophisticated methods, but rather to shed light on practical applicability of using UAS for autonomous ice detection with similar platforms. A significant part of the workload will naturally go to the implementation of the system on an experimental platform. It is nonetheless necessary to develop a theoretical framework before implementing any algorithms, both to aid the development and to increase the

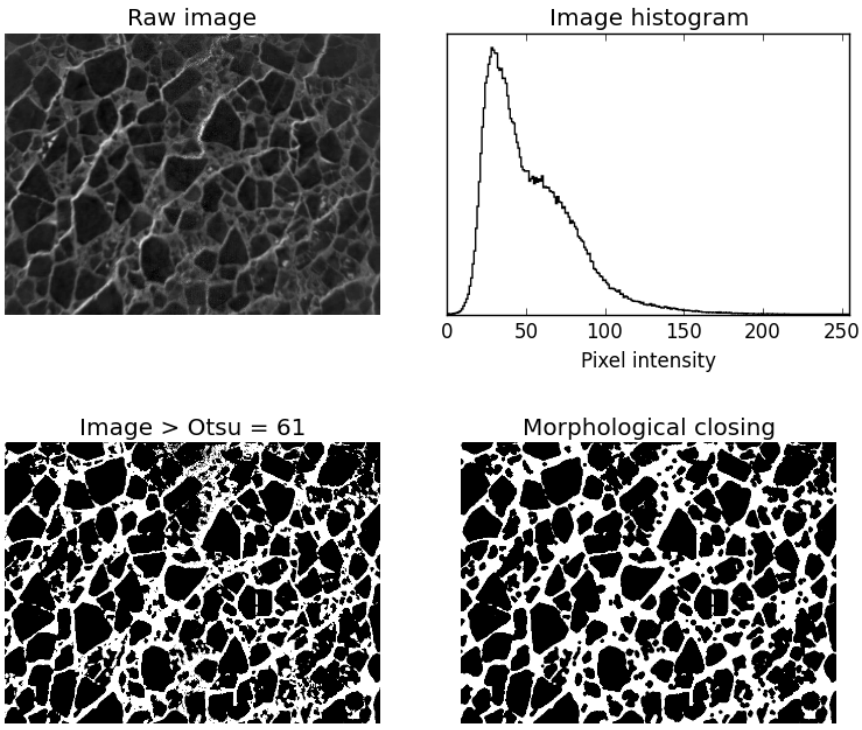


Figure 2.1: Thermal image of dense sea ice together with a histogram of grey values, a thresholded image and its morphological closure. Courtesy of NORUT

probability that the lessons learned during this project can be used in further research.

2.5.1 General Bayesian recursive estimation

A common problem in robotics is to infer the state of some physical system at a point in time by using past observations (measurements) and inputs. In tracking methods and other estimation problems in the field of robotics in general, Bayesian recursive estimation is a indispensable statistical tool. With additional assumptions, the method can be used to derive most modern recursive estimation algorithms such as the Kalman Filter and its many varieties. This section presents the method briefly, nearly identical to the presentation in Thrun 2002. For simplicity and relevance to this thesis, it is assumed that the stochastic processes which we want to infer information about has no known inputs. That is, we want to infer something about the stochastic process x_t at time t, given measurements $z_{1:t} := \{z_1, z_2, \ldots, z_t\}$. In particular, we would like to say something about the

following conditional probability distribution also called the *belief* of the state x at time t:

$$bel(x_t) := p(x_t|z_{1:t}) \tag{2.13}$$

A very important assumption about the stochastic process is that it is *Markov*. The Markov assumption essentially states that given the current state x_t, no additional information is gained by including previous states and measurements when trying to predict the next state x_{t+1}. In other words, the state x_t is *complete* in the sense that it summarizes everything that has happened in the past. This assumption is usually known to be overly optimistic in robotics applications, but yields good results in practice and computationally tractable algorithms. The Markov assumption can be summarized by the *conditional independence*:

$$p(x_t|x_{t-1}, z_{1:t-1}) = p(x_t|x_{t-1}) \tag{2.14}$$

where this particular conditional probability density plays an important role and is often called the *predictive* distribution. It will be given the following notation in further derivations:

$$\overline{bel}(x_t) := p(x_t|x_{t-1}) \tag{2.15}$$

In general the Bayesian recursive estimation algorithm, also called the Bayes Filter, tries to calculate the *posterior* distribution $bel(x_t)$ from a prior distribution $bel(x_{t-1})$ and a new measurement z_t. It does so by marginalizing over *all* previous states and using Bayes rule. These steps can be summarized by the two equations

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1})bel(x_{t-1})dx_{t-1} \tag{2.16}$$

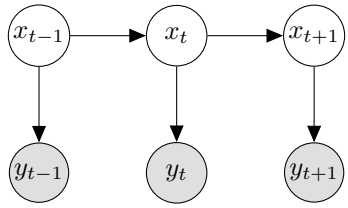
$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{2.17}$$

where η is a normalization factor such that $bel(x_t)$ is a proper probability distribution (i.e. it integrates to one). When using a Bayesian framework, we have to find the conditional probability distributions $p(z_t|x_t)$ called the *measurement model*, and $p(x_t|x_{t-1})$ called the *prediction model*. Both models are often obtained by first principle modeling together with some reasonable inclusion of uncertainty. In addition, we need to figure out how to evaluate the integral (2.16). The integral is usually not possible to evaluate explicitly, with the very important exception when we use a linear model with Gaussian noise yielding the standard Kalman Filter. Another important exception to this is when we use a finite representation of the probability distributions involved, changing the (possibly infinite) integral into a finite sum. Using finite representations has the additional advantage of being able to adapt the number of finite elements in the approximation of the probability distribution, giving the possibility to adapt to the available computational resources. The probabilistic structure of the problem the Bayes filter attempts to solve can be illustrated by the following graphical model as shown in Figure 2.2

2.5.2 Occupancy grid mapping

Estimating a map from a moving robot is a challenging task. Today there is a lot of active research in the field of SLAM, which tries to estimate the pose

Figure 2.2: A graphical model of the latent (white) and observed (grey) variables in a Markov chain



of a robot and at the same time estimate the state of the world in the form of a map. In this thesis a simplified problem will be considered, where the pose of the robot (i.e. the UAV) is known. The results of the mapping will still be highly dependent on the estimates of the robot pose, but the algorithms will be much simpler to implement. An arguably useful way of representing sea ice for ice management is by an "occupancy grid map". This is a simplified world fixed grid map that tries estimate if grid locations are occupied or free, in a probabilistic way. The algorithm presented in this section is a slight generalization of the algorithm presented in Thrun 2002 to include the possibility of a dynamic map. The Holy Grail is to estimate the posterior probability density over all possible maps given past measurements and states

$$p(m_t|z_{1:t}, x_{1:t}) \tag{2.18}$$

where m_t is the map representation, $z_{1:t}$ are measurements (in our case camera images) and $x_{1:t}$ are robot poses. Such maps have successfully been used for planar robot localization and motion planning. In this thesis, we are interested in the utility of using UAV in ice management operations consisting of ocean vessels clearing and avoiding dangerous ice. The motion of an ocean vessel is similar to planar robot movement in the sense that motion control systems usually only actively control planar position and the vessel heading. And it seems reasonable that such a map alone contains enough information for either a robot or a human to identify safe and unsafe areas for ocean vessels. A limitation with a static map in relation to ice management, is the lack of temporal information in terms of ice drift. It is possible to include ice dynamics in the estimation of the grid map as indicated by the time dependence in m_t, it might however not be strictly beneficial since it introduces additional complexity in modeling and implementation, the issue is discussed later in this section. The grid map estimation will be tackled as a recursive estimation problem, with the map defined as a 2D grid map

$$m_t = \{\boldsymbol{m}_{i,t}\} \tag{2.19}$$

where $\boldsymbol{m}_{i,t}$ are grid elements each with an associated binary occupancy value. The quantities that we wish to estimate are the probabilities $p(\boldsymbol{m}_{i,t} = \text{occupied})$ and $p(\boldsymbol{m}_{i,t} = \text{not occupied})$ for each grid cell, where *not occupied* in this context would mean that the space in grid cell i is open sea and thus safe for a vessel to traverse.

The first assumption made to be able to calculate (2.18) is to assume that the grid cell densities are independent. As an equation this reads

$$p(m_t|z_{1:t}, x_{1:t}) = \prod_i p(\boldsymbol{m}_{i,t}|z_{1:t}, x_{1:t}) \tag{2.20}$$

implying that the state of neighbouring cells do not affect each other. Intuitively this is a crude assumption, since the probability of open sea in one grid cell should tell us something about the probability of open sea in the neighbouring cell as well. There are ways to improve on this assumption, but the resulting algorithms are more computationally complex and require some additional modeling effort (as usual). The problem of estimating (2.18) thus amounts to the estimation of the state of each individual grid cell. Each of these problems is a binary estimation problem. With the additional assumption that $\boldsymbol{m}_{i,t}$ and x_t are Markov, we can estimate the grid state using a *Binary Bayes Filter*. A Bayes filter is normally implemented in two steps, the first being a predictive step using knowledge of the dynamics of the system, and the next being a corrective step using measurements to improve the estimate. Using the law of total probability and the Markov assumption, the predictive step in our case is to calculate:

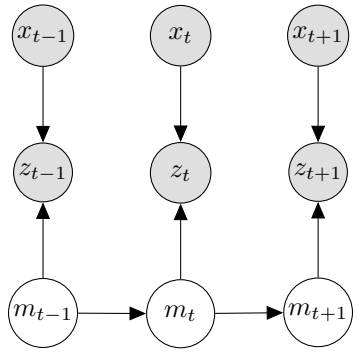
$$\begin{aligned}
\overline{bel}(\boldsymbol{m}_{i,t}) &= p(\boldsymbol{m}_{i,t}|z_{1:t-1}, x_{1:t-1}) \\
&= \sum_{\boldsymbol{m}_{i,t-1}} p(\boldsymbol{m}_{i,t}|\boldsymbol{m}_{i,t-1}) p(\boldsymbol{m}_{i,t-1}|z_{1:t-1}, x_{1:t-1}) \\
&= \sum_{\boldsymbol{m}_{i,t-1}} p(\boldsymbol{m}_{i,t}|\boldsymbol{m}_{i,t-1}) bel(\boldsymbol{m}_{i,t-1})
\end{aligned} \tag{2.21}$$

where the dynamics of the occupancy value is encoded in the transition probabilities $p(\boldsymbol{m}_{i,t}|\boldsymbol{m}_{i,t-1})$, and we have defined $bel(\boldsymbol{m}_{i,t-1}) = p(\boldsymbol{m}_{i,t-1}|z_{1:t-1}, x_{1:t-1})$. In the case of a static map, this step changes to simply set $\overline{bel}(\boldsymbol{m}_{i,t}) = bel(\boldsymbol{m}_{i,t-1})$ since the map does not change over time. The next and final step in the Bayes recursion is to integrate a new measurement z_t to improve our predictive estimate. Using Bayes rule we get

$$\begin{aligned}
bel(\boldsymbol{m}_{i,t}) &= p(\boldsymbol{m}_{i,t}|z_{1:t}, x_{1:t}) \\
&= \frac{p(z_t|\boldsymbol{m}_{i,t}, x_t) p(\boldsymbol{m}_{i,t}|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \\
&= \frac{p(z_t|\boldsymbol{m}_{i,t}, x_t) \overline{bel}(\boldsymbol{m}_{i,t})}{p(z_t|z_{1:t-1}, x_{1:t})}
\end{aligned} \tag{2.22}$$

where $p(z_t|\boldsymbol{m}_{i,t}, x_t)$ is the probability relating the state of a grid cell and the pose of the UAV to the measurement z_t. The structure of the measurements can be illustrated by Figure 2.3. This measurement density might be very hard to obtain or define, as we in essence have to describe every possible image given the pose of the UAV and the state of a grid cell. It arguably makes more sense to define the *inverse* measurement model $p(\boldsymbol{m}_{i,t}|z_t, x_t)$. Using Bayes rule again on

Figure 2.3: Graphical illustration of the map probability model. Grey nodes are observed quantities, white are unknown (estimated)



the measurement model we obtain

$$\begin{aligned} p(z_t|\boldsymbol{m}_{i,t}, x_t) &= \frac{p(\boldsymbol{m}_{i,t}|z_t, x_t)p(z_t|x_t)}{p(\boldsymbol{m}_{i,t,|}x_t)} \\ &= \frac{p(\boldsymbol{m}_{i,t}|z_t, x_t)p(z_t|x_t)}{p(\boldsymbol{m}_{i,t})} \end{aligned} \tag{2.23}$$

Inserting this into (2.22) yields

$$bel(\boldsymbol{m}_{i,t}) = \frac{p(\boldsymbol{m}_{i,t}|z_t, x_t)p(z_t|x_t)\overline{bel}(\boldsymbol{m}_{i,t})}{p(\boldsymbol{m}_{i,t})p(z_t|z_{1:t-1}, x_{1:t})} \tag{2.24}$$

The next step is to use the fact that the ith map state is binary, i.e. it can only be either occupied or free. This leads to the fact that we can calculate the probability that a grid cell it *not* occupied by

$$p(\neg\boldsymbol{m}_{i,t}) = 1 - p(\boldsymbol{m}_{i,t}) \tag{2.25}$$

Defining the *odds ratio* as the fraction between a grid cell being occupied and not, and using the fact that the negated belief yields the same result as in (2.22), we can cancel a few terms not depending on the grid state to get

$$\frac{bel(\boldsymbol{m}_{i,t})}{bel(\neg\boldsymbol{m}_{i,t})} = \frac{\overline{bel}(\boldsymbol{m}_{i,t})p(\boldsymbol{m}_{i,t}|z_t, x_t)p(\neg\boldsymbol{m}_{i,t})}{\overline{bel}(\neg\boldsymbol{m}_{i,t})p(\neg\boldsymbol{m}_{i,t}|z_t, x_t)p(\boldsymbol{m}_{i,t})} \tag{2.26}$$

Finally, defining $l_{i,t}$ to be the log of the odds ratio we get the following additive update equation

$$l_{i,t} = \log\frac{\overline{bel}(\boldsymbol{m}_{i,t})}{1 - \overline{bel}(\boldsymbol{m}_{i,t})} + \log\frac{p(\boldsymbol{m}_{i,t}|z_t, x_t)}{1 - p(\boldsymbol{m}_{i,t}|z_t, x_t)} - \log\frac{p(\boldsymbol{m}_{i,t})}{1 - p(\boldsymbol{m}_{i,t})} \tag{2.27}$$

where the first term in this update equation will simply be $l_{i,t-1}$ if we neglect map dynamics. The last term is the prior log odds ratio, and can be interpreted as a

quantity returned by the inverse measurement model when no information is added by a measurement. The map probabilities can easily be obtained by calculating

$$p(\boldsymbol{m}_{i,t}|z_{1:t}, x_{1:t}) = \frac{\exp(l_{i,t})}{1 + \exp(l_{i,t})} \tag{2.28}$$

2.5.3 Inverse measurement model

In the previous section, an algorithm for recursive estimation of an occupancy grid map was developed. Central to this algorithm is the inverse measurement model

$$p(\boldsymbol{m}_{i,t}|z_t, x_t) \tag{2.29}$$

which in our case represents the probability of the map grid cell i being occupied given an image z_t and a pose x_t. Hopefully, a raw image together with the pose of the robot contains this information. The choice of the measurement model is vital for the result of the mapping algorithm. The determination of the inverse measurement model can be regarded as a problem of finding a function

$$f : M \times Z \times X \to [0, 1] \tag{2.30}$$

where the input space consists of the map grid position (or index) denoted by M, image features Z and the robot pose X. In its simplest form the function can return a constant probability if a specific feature is observed near the grid cell, perhaps weighted by the distance to the grid cell center. The function should ideally be found in a systematic way aided by empirical data. Alternatively the function can be defined heuristically by defining sensor characteristics. Currently there is very little data available to do any meaningful systematic estimation of the function f, so a heuristic treatment is the only viable option. A possible way of defining this function is to first consider the segmentation of an observed image. If an image has been segmented into "ice" and "not-ice" pixels, the segmented image can be projected and interpolated onto the image grid directly and interpreted as detection probabilities. The quality of a measurement is then dependent on both the image segmentation and the estimation of the image perspective projection. In the inverse measurement model presented here we first define a partial measurement model by processing the image data, then account for the pose by projecting the processed image onto the mapping frame.

We first consider the image data z_t, we saw in section 2.4 that an image can be segmented into "ice" and "not-ice" classes. A straightforward thing to do is to first segment the image into these two classes. Once segmented, we can define the probabilities that a grid cell is occupied if the segmented pixel belongs to the each of these classes. The most obvious thing to do might be to simply define the probability to be equal to one if the pixel belongs to the "ice" class and to zero of it belongs to the "not-ice" class. However, this is very unlikely to reflect the true probabilities since there is significant uncertainty in the segmentation itself. Assuming that the image data is represented in the map frame, we can thus define an inverse measurement model by assigning these probabilities. Denote these

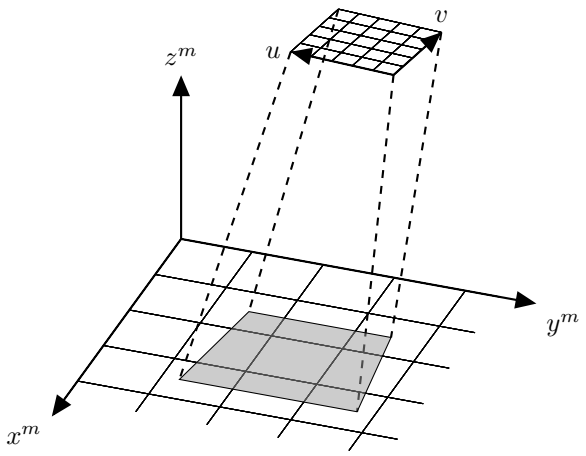


Figure 2.4: Illustration of the map grid and the perspective transformation

probabilities by P_1 and P_2 and let $s_{k,t}$ represent a segmented pixel of the image z_t. Recall that we are interested in the log-odds form of the inverse measurement model, we can assign the following values to our partial inverse measurement model:

$$n_{k,t} = \begin{cases} \log \frac{P_1}{1-P_1} & \text{if } s_{i,t} = \text{ice} \\ \log \frac{P_2}{1-P_2} & \text{if } s_{i,t} = \text{notIce} \end{cases} \tag{2.31}$$

where $n_{k,t}$ is the partial inverse measurement model. The next step is to relate this measurement to the map frame using the pose x_t.

The UAV pose x_t does not contain direct information about the state of the grid cell, but rather information about where image features observed are located relative to the grid cell i. Since the captured images are projected onto a plane we loose some information about the spatial location of the observed features in the image (i.e. the depth). We have however already constrained the grid mapping to be in a plane at sea level. With the assumption that image features (e.g. from ice features) are generated in this sea level plane, it is possible to invert a projection model like (2.12) to obtain the planar position of a feature in a world fixed reference frame. In other words, there exists a function that maps a pixel coordinate in the image plane to a grid cell in the mapping plane.

Given that we want to relate a camera measurement to the map grid, the most direct way to do this is to project and interpolate the camera measurement (i.e. the image) onto the map grid plane. An illustration of this is shown in Figure 2.4. It will now be shown that if we use the pinhole projection model (2.10) the projection a linear perspective projection, or a homography. The first parameters that need to be defined are the position and orientation of a fixed map frame relative to some common world frame shared by the map and camera frames. These parameters

can be written as the homogeneous transformation matrix:

$$\boldsymbol{T}_m^w = \begin{bmatrix} \boldsymbol{R}_m^w & \boldsymbol{r}_{wm}^w \\ \boldsymbol{0}^T & 1 \end{bmatrix} \tag{2.32}$$

An occupancy grid map is represented by a matrix in $\mathbb{R}^{N \times M}$. Let a map grid cell (a matrix element) be given by homogeneous coordinates $\widetilde{\boldsymbol{p}}^m = (u^m, v^m, 1)$, with $u^m \in \{0, \ldots, N-1\}$ and $v^m \in \{0, \ldots, M-1\}$. The map grid is defined as lying in the positive quadrant of the $x-y$ plane in the map frame. The metric location of the center of a grid cell in the map fixed reference frame can be calculated by:

$$\boldsymbol{q}^m = \begin{bmatrix} D_x & 0 & \frac{1}{2}D_x \\ 0 & D_y & \frac{1}{2}D_y \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u^m \\ v^m \\ 1 \end{bmatrix} = \boldsymbol{M}\widetilde{\boldsymbol{p}}^m \tag{2.33}$$

where D_x and D_y are metric scaling factors of a grid cell. The location of a grid cell can now be expressed in the world fixed frame by:

$$\boldsymbol{q}^w = \boldsymbol{R}_m^w \boldsymbol{M}\widetilde{\boldsymbol{p}}^m + \boldsymbol{r}_{wm}^w = \left(\boldsymbol{R}_m^w \boldsymbol{M} + [\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{r}_{wm}^w]\right)\widetilde{\boldsymbol{p}}^m := \boldsymbol{P}_m^w \widetilde{\boldsymbol{p}}^m \tag{2.34}$$

Using the pinhole projection model, and denoting an image pixel coordinate by $\widetilde{\boldsymbol{p}}^c = (u, v, 1)$ we can project the grid cell onto the image frame yielding:

$$s\widetilde{\boldsymbol{p}}^c = \boldsymbol{A}\left(\boldsymbol{R}_w^c \boldsymbol{P}_m^w \widetilde{\boldsymbol{p}}^m + \boldsymbol{r}_{cw}^c\right) = \boldsymbol{A}\left(\boldsymbol{R}_w^c \boldsymbol{P}_m^w + [\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{r}_{cw}^c]\right)\widetilde{\boldsymbol{p}}^m := \boldsymbol{X}_m^c \widetilde{\boldsymbol{p}}^m \tag{2.35}$$

where $\boldsymbol{X}_m^c \in \mathbb{R}^3$ is the perspective transformation between the map grid plane and camera pixel plane and s is some constant. If this matrix is full rank it can be inverted to yield:

$$sp^m = \left(\boldsymbol{X}_m^c\right)^{-1} p = \boldsymbol{X}_c^m p \tag{2.36}$$

where the matrix $\boldsymbol{X}_c^m$ can be used to project camera images onto the map plane. The matrix is in fact a *homography*, and it is invertible as long as the camera center does not lie in the mapping plane (Hartley and Zisserman 2004). If the camera center is indeed in the mapping plane, something is either wrong with the navigation system or the UAV has most likely crashed in the ocean.

Now that we can project images onto the mapping plane, we can complete our inverse measurement model by transforming the partial measurement model $n_{k,t}$ given in (2.31) onto the mapping plane. In OpenCV this geometric transformation can be done with the function "warpPerspective".

When the occupancy grid estimation algorithm was developed, it was assumed that the pose x_t was known perfectly. This is not the case in practice, as both the position and attitude of the UAV are estimated from sensor measurements and contain significant errors. Care should thus be taken when the inverse image projection is used to estimate the position of a feature, optionally including some simple quantification of the uncertainty in the projection like a two dimensional isotropic Gaussian distribution. To include this uncertainty would mean that we "smear" the detected features out over a larger area in the mapping plane. In practice, a simple way of including a Gaussian uncertainty model is by filtering the projected image with a Gaussian kernel. Filtering with a Gaussian kernel is included in the OpenCV image processing module.

2.5.4 An algorithm for recursive map estimation

In this section the mathematical procedures are summarized in an algorithm which is conceptually identical to the one implemented in the experiments. The algorithm requires an initial occupancy grid map estimate m_0 in log-odds form, map parameters as defined by 2.33, intrinsic camera parameters, filter parameters P_1, P_2 as in (2.31), Gaussian blur parameter σ for localization uncertainty and sequential data in the form of images z_k and corresponding camera poses x_k, a map can be estimated recursively using the algorithm listed in Algorithm 1.

Algorithm 1 Basic occupancy grid map estimation

procedure RECURSIVEMAPESTIMATION(m_k, z_k, x_k)
  $s_k \leftarrow$ SegmentImage(z_k)
  $n_k \leftarrow$ ToLogOdds(s_k, P_1, P_2) ▷ (2.31)
  $\overline{n_k} \leftarrow$ GaussianFilter(n_k, σ)
  $X_c^m \leftarrow$ ComputeMapHomography(x_k) ▷ Use the result in (2.36)
  $N_k \leftarrow$ WarpAndInterpolate($\overline{n_k}, X_c^m$) ▷ Transform to map plane
  $m_{k+1} \leftarrow m_k + N_k$ ▷ Additive update rule (2.27)
end procedure

Chapter 3

System architecture

This chapter describes the system architecture of a proposed ice monitoring system. The design of the ice monitoring system developed in this thesis is mostly done in the software domain. It thus makes sense to describe the overall system as communicating software components. Parts of the implemented software will have to run onboard the UAS, and some parts will likely run on a ground station close to an UAS operator. This implies that a telemetry link is needed even during autonomous operation. It is however the aim to develop systems that can be operated with as much autonomy as possible, and in BLOS (Beyond Line Of Sight) operations. An experimental operation was planned in an area with very limited communication coverage (Svalbard), the expected data rates on a telemetry link will thus be quite low and the link might also experience frequent drop-outs. This leads to the requirement that the autonomy of the system should not be adversely affected by a poor communication link. A consequence of this is that real-time operation of the system will likely rely on the ability to process the large amounts of image data on-board the UAS, rather than sending the data to be further processed. With reference to Figure 3.1, the main focus has been on the development of a novel *ice mapping* component. The other components have been involved in the testing of the ice mapping component in a laboratory environment which will be described in this chapter in more detail. Figure 3.1 also illustrates which software components that run onboard the UAS in the proposed ice monitoring system, and illustrates communication links between them. Further in this chapter, the most relevant software frameworks, tools and libraries are described briefly. Finally, an experimental setup to test the proposed ice monitoring system is presented.

3.1 Software libraries and frameworks

3.1.1 DUNE

Before the Svalbard excursion was canceled, the plan was to implement the ice mapping component in a software framework called DUNE (Dune is an Unified

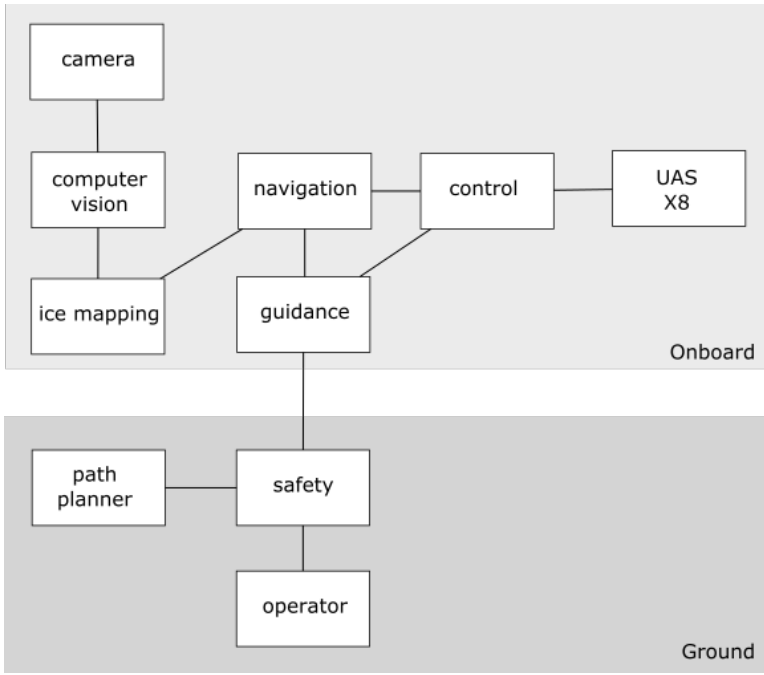


Figure 3.1: Proposed software component diagram for an ice monitoring system

Navigation Environment). The main reason for this is that the NTNU UAV lab uses this framework for UAS development and experiments, and many PhD and master students have experience with working with this framework. DUNE is a software framework designed to be used on board mobile robotic platforms mainly in the air and at sea. It has a corresponding software framework for ground station monitoring and control, called *Neptus*. The main utility of the framework is to enable a robotic software system to be developed as independent *tasks* that communicate through a well defined protocol. In DUNE the tasks are usually periodic or event driven, and the messages are sent as pre defined Intermodule Communication (IMC) messages. The power communicating with pre defined messages is that these messages can be transported over many different physical layers, including within computer memory, wired network connections and even wireless communication systems. Thus IMC messages enable both inter process communication and telemetry, command and control.

The laboratory setup developed for this thesis work was developed with DUNE and Neptus compatibility among the main goals. The idea was that an ice monitoring system could be developed in DUNE and tested on a laboratory platform and deployed on the experimental Svalbard platform with very few modifications. After the Svalbard excursion was canceled however, these frameworks became less important. The Svalbard excursion was canceled after the laboratory setup had been tested to work with both Neptus and DUNE, and it was used by master student

Recep Cetin working with validating a navigation system in some of his experiments. For this thesis it was used to test a path planning framework meant to be a part of the experimental system on Svalbard, further explained in section 4.1.2.

3.1.2 OpenCV

OpenCV is the only software dependency that the ice mapping software component has. OpenCV is an open source computer vision software library designed for high performance, real time computer vision. It was originally written in C/C++, but now has interfaces in several other languages including Python and Java. The library is released under a BSD license and is free for both academic and commercial use. Industrial leaders such as Intel and Google contribute to the development financially and through library development. OpenCV provides basic data structures for handling image data, image processing algorithms for e.g. filtering and warping of images, video analysis algorithms, multiple view geometry algorithms (Hartley and Zisserman 2004) for calibration and reconstruction, feature detectors and descriptors for tracking and detection and more.

3.1.3 ROS

In this project ROS (Quigley et al. 2009) has been used as a framework for the development of the experimental lab setup. It has been used both to test the ice mapping functionality in an isolated way, and to integrate the laboratory systems with DUNE to test path planning functionality. Like DUNE, ROS is an open-source software toolkit for robotics applications. It is not an operating system in the traditional sense, but provides a software framework suitable for robotic systems. Currently ROS only runs on top of UNIX like operating systems, like Ubuntu Linux and Mac OSX. This is a quote from the ROS documentation:

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

The main goal of ROS is to support code reuse in robotics research and development. It is designed to be as thin as possible, and has been integrated with other robotics frameworks. In this thesis it is in fact integrated with DUNE. An important note is that ROS is not designed to be a real-time framework. A central concept in ROS is that the tasks that make up a robot application can be represented by a graph. The graph nodes are the running processes and the edges are communication between the tasks. Communication between tasks can be done in several ways. Synchronous communication can be done via ROS “services”, asynchronous message passing over “topics” and storage of configuration data on the “Parameter server”. The concepts are implemented in a programming language agnostic way, such that applications can be developed in e.g. C++ or Python (the

two main implementation languages).

A very useful tool included in the ROS core libraries is the "tf" software library. It provides a standard way of keeping track of coordinate frames and performing transformations between different reference frames. A comprehensive overview of the library can be found in Foote 2013. In this project the library has been extensively used to publish and retrieve the pose of the UAV used in experimental setups. A number of static transformations have been defined as a convenience for other ROS nodes used in experiments.

3.1.4 MAVLINK

MAVLINK is an open source generic UAS communication protocol. It was first released in 2009 by Lorentz Meier under a LGPL licence, see *MAVLINK official site* 2015 for more information. It is used to communicate UAS state, parameters and commands such as desired paths. In this project it is used as a communication bridge between DUNE and ROS in a laboratory environment.

3.2 Ice map estimation

This section briefly describes the implemented ice map estimation algorithm developed in Chapter 2. A C++ module was developed with the aim of being framework agnostic. The reason for this was that even though the system was to be implemented in DUNE for the planned experiments on Svalbard, the system had to be tested in an ROS environment in the laboratory. The only dependency is OpenCV, but this is available both under ROS and DUNE build environments. The main purpose of the module is pretty much a straightforward implementation of Algorithm 1 using image processing and linear algebra functions from OpenCV. The occupancy grid map is represented as a matrix in log-odds form, and camera measurements are transformed and aggregated as described in Algorithm 1. The C++ interface for the module is included in Appendix A.2.3. The mapping algorithm is tested extensively in Chapter 4.

3.3 Experimental lab setup

Early in the project work it was assumed that testing on the Svalbard platform would be very limited. However, testing was considered very important for the success of the Svalbard experiment. Considerable time was thus spent creating a laboratory environment that would emulate the behaviour of the Svalbard platform. For the purposes of the ice monitoring experiment, the experimental Svalbard platform for ice monitoring can be thought of as an UAS with autonomous waypoint following capabilities and with a camera payload mounted in a stabilized frame. It seems plausible that the high level features of ice mapping and path planning can be reproduced in a laboratory environment. Many failures can be eliminated by good design and a lot of testing can be done by unit testing software and hardware components, but in the end complete system tests will often reveal new flaws that

cannot be found through other testing. To do outdoor flight tests the NTNU UAV-lab has to travel to Agdenes, about 80 km from NTNU in Trondheim. Test flights at Agdenes need to be carefully planned and require a licenced UAV operator. The UAV lab usually does a lot of HIL testing before doing flight tests. The HIL tests are suitable for testing guidance and control systems, but it does not really make sense to test computer vision algorithms. Ideally the test platform should be able to both guide the development of an ice monitoring system, and reveal flaws in its design at an early stage in development. The UAV lab at NTNU have access to an indoor robotics lab operated by SINTEF called the "snakelab" (the lab is used for research on snakerobots). The lab has a motion tracking system and a small quadcopter with two cameras. In the development of an indoor test platform, ROS was used extensively to glue together a working software interface emulating the Svalbard platform. A diagram illustrating the software components used and implemented is shown in Figure 3.2.

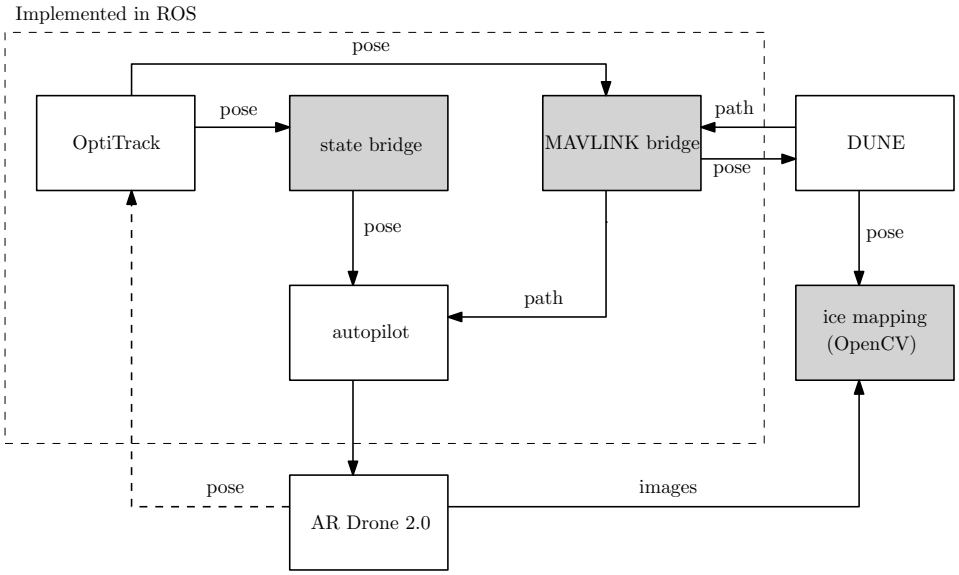


Figure 3.2: Software components in the laboratory setup. Grey boxes have been implemented in this project work. White boxes are open source software components.

3.3.1 Natural Point OptiTrack

As a part of indoor testing of computer vision and path planning algorithms, a motion capture system called "OptiTrack" from Natural Point is used for indoor navigation. The motion capture system provides millimeter accuracy 6-DOF (Six Degrees Of Freedom) state measurement of a rigid body at 120 Hz. The motion capture setup does this by emitting infrared light which is reflected by special

reflective markers mounted on the rigid body. After capturing the reflected infrared light with several cameras it is possible to calculate the position and orientation of the rigid body relative to the cameras. Calibration of the cameras is done beforehand such that the relative position of the cameras to each other and a ground plane is known. The cameras feed data to a standard desktop computer running proprietary software called "Motive" to do the necessary calculations. The calculated measurements can be visualized using the "Motive" software and further broadcast over network via UDP. The UDP feed can be parsed by an ROS device driver called "mocap-optitrack" and further publish a transformation equivalent to (2.3) presented in Chapter 2 in real time. The standard transformation published is a transformation from a world-fixed reference frame to a body fixed coordinate system fixed to the markers on the rigid body. These two reference frames both have a z-axis pointing up.

3.3.2 Parrot AR Drone

The quadcopter in the "snakelab" is a Parrot AR Drone 2, see Figure 3.3. The AR Drone is designed as a recreational multicopter platform for aerial photography. It is made of relatively cheap components, but the performance of the control system and video streams are still deemed adequate for the purposes of this thesis work. The advantages of the platform is the ease of use and the availability a relatively simple software API (Application Programming Interface). The AR Drone can be communicated with via Wi-Fi, and there exists an ROS driver called "ardrone_autonomy" to integrate the platform with other software tasks (e.g. an autopilot or image processing algorithms). An autopilot for indoor control was also necessary to implement. Fortunately a master student from the Technical University of Munich (Engel, Sturm, and Cremers 2014) had already done this with the "ardrone_autonomy" driver and released the source code. His package contains a position autopilot implemented by independent PID control loops, a state estimator and a simple GUI. The package does however not use a world fixed external navigation system, but used the on-board cameras and inertial sensors of the AR Drone to perform SLAM and data fusion of the inertial sensors with an EKF (Extended Kalman Filter). Due to the modularity of his code and the ROS framework, it was a relatively simple task to change the state estimate from the EKF with a pose measurement from the "OptiTrack" motion capture system. In principle it was only necessary to transform a state measurement from the OptiTrack system to the same format produced by the vision based EKF. The implementation of this conversion is shown in Appendix A.2.2. After applying a small patch to the autopilot, the AR Drone was capable of autonomous flight relative to the motion capture system. The patch applied consists of some sign changes in the autopilot control loops, the patch itself is included in Appendix A.2.4. I.e. it was able to follow a path defined by a sequence of poses defined in the world fixed reference frame. A pose in this case refers to a position and a heading (yaw angle). The roll and pitch of the quadcopter have to be zero for the system to be stable. The camera on the AR Drone that is pointing down was calibrated using standard methods in OpenCV, and were also validated in MATLAB using the same data.

The calibration data and results can be found in the attached zip file.

Figure 3.3: Fotograph of the AR Drone 2.0 with reflective OptiTrack markers attached

3.3.3 ROS implementation

As already mentioned, open source drivers for ROS exist for both the AR Drone and the OptiTrack system. This made ROS a suitable software framework candidate for developing an indoor test platform. The test platform should mimic the actual platform to be used on Svalbard as closely as possible, both to make the tests more realistic and to reduce the integration work needed before the Svalbard operation. By using existing open source software, it was possible to create an autopilot for the AR Drone such that it can track arbitrary sequences of poses (position+heading) in space relative to the motion capture system. After investigating the software interfaces used in the planned Svalbard platform, it was clear that choosing a MAVLINK interface would be the most natural choice in communicating with DUNE. As MAVLINK is a reasonably generic and stable communication protocol for UAS, this choice supports the future use of the test platform in other projects as well. There are also MAVLINK utilities available in ROS, simplifying the implementation greatly. A ROS package including all the implementations necessary to perform experiments was made. A description of how to install and use the basic features of the software package is included in Appendix A.3.

Conversion to a global frame

The autopilot interfaced from DUNE assumes that paths are represented in a global reference frame, specifically the WGS84 ellipsoidal global reference system. Since positions relative to the motion capture system are represented in a local Cartesian frame, it is relatively straightforward to convert to and from the WGS84 representation (latitude, longitude, height above the ellipsoid). This conversion had to be done in real-time when receiving desired paths from DUNE or sending current positions to DUNE. The algorithms described by Fossen 2011, p. 34-39 to convert to/from an ECEF (Earth Centered Earth Fixed) frame to a WGS84 representation were implemented in C++ to perform the conversions. In addition, a static transformation from the world fixed frame defined by the motion capture system to the ECEF frame was defined in ROS. It should be noted that the actual global position reported by this system is not very accurate. The error lies in the transformation from the world fixed frame to the ECEF frame, including the specification of "north" relative to the world fixed system. This transformation was more or less guessed by the author by recording the approximate latitude, longitude, height above the ellipsoid and heading at the location of the lab. This is not an issue in the applications presented in this thesis however since only relative accuracy is important, which is guaranteed by the motion capture system down to millimeters.

Static local transformations

Aerospace and navigation conventions suggest having an earth fixed NED frame (hereby denoted *ned*) and a body fixed frame with the x-axis pointing forward and the z-axis pointing down (denoted *aero*). This is the case for the autopilot interfaces used by DUNE. This implies that positions and poses have to be transformed into proper frames for communication with DUNE. Since the motion capture system publishes the pose of the AR Drone with respect to frames with the z-axis pointing up, two static transformations are published to be able to use standard aeronautical conventions. The world fixed motion capture system is simply denoted as *world*, and the body fixed as *body*. In practice the transformations between these frames are described by a simple rotation of the body frame, and a simple rotation of the world fixed frame. To make things a bit more concrete, consider the body fixed frame published by the motion capture system. This can be related to a standard aeronautical frame with the x-axis pointing forward and z-axis pointing down by a rotation about the x-axis with a magnitude of π radians. Using the extended Euler formula (2.8), it can be computed by:

$$q_{ned}^{world} = \cos\frac{\pi}{2} + \vec{i}\sin\frac{\pi}{2} = \vec{i} \tag{3.1}$$

and since the body frame is related to the aeronautical frame by the same rotation, we have:

$$q_{aero}^{body} = \vec{i} \tag{3.2}$$

Like in the definitions of the transformations between the local frames and a global frame, it should not be assumed that the NED frame is indeed pointing towards a

true north. These simple transformations are used to transform information from the motion capture system to DUNE and the ice mapping algorithm using the “tf” ROS tools.

Chapter 4

Experiments

4.1 Indoor Testing

To increase the probability of success on Svalbard, indoor tests of some of the system components were performed. Many failures can be eliminated by good design and a lot of testing can be done by unit testing software and hardware components, but in the end complete system tests will often reveal new flaws that cannot be found through other testing. More details on the experimental lab setup can be found in section 3.3.

4.1.1 Test of occupancy grid mapping

Several white paper patches were placed on the floor in the indoor test lab to test the map grid estimation algorithm, see Figure 4.1. The position of the patches relative to the motion capture system fixed world coordinate system was recorded by placing a set of markers on each patch and manually recording the location. The location of the patches and the (coarse) maximal radius of an inscribed circle in the paper patch can be seen in Table 4.1.

Patch	**x** [mm]	**y** [mm]	**r** [cm]
1	-1068	2615	3
2	-800	1824	3
3	-1316	1593	4
4	101	1316	4
5	-332	1026	4.5
6	-22	2069	8
7	-557	2399	6.5
8	-1129	626	5

Table 4.1: Location and relative size of paper patches

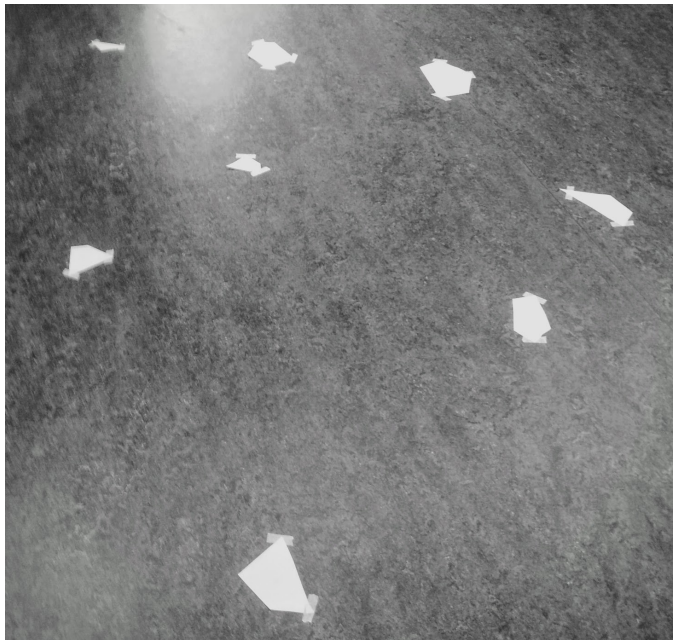


Figure 4.1: Photograph of the paper patches placed on the ground.

The patches had varying geometry but were all approximately simple polygons like triangles and rectangles. The AR Drone was flown manually over the patched area, and both the pose of the drone and the image stream was recorded with the "rosbag" ROS tool. A visualization of one of the paths flown is shown in Figure 4.2. Image segmentation together with the computation of the homography between the image plane and the mapping plane has been discussed earlier as an integral part of the mapping algorithm. Examples of these procedures for the relevant datasets are illustrated in Figure 4.3. Three datasets were recorded and are presented in this section. The paths flown seen from above and the map estimates for the corresponding datasets can be seen in Figure 4.5.

The map estimates are annotated with numbers corresponding to the patches listed in Table 4.1. The numbers are located approximately at the locations of the patches. The exact locations of the patches is not that important, since the estimates show pretty large errors in position anyway. There has not been done any effort to quantify the error since the results were not very promising by visual inspection.

4.1.2 Test of path planning framework

One of the main reasons for spending time on implementing a laboratory set up before the Svalbard excursion, was to test a path planning framework developed by PhD candidate Anders Albert. His framework is developed in MATLAB, and

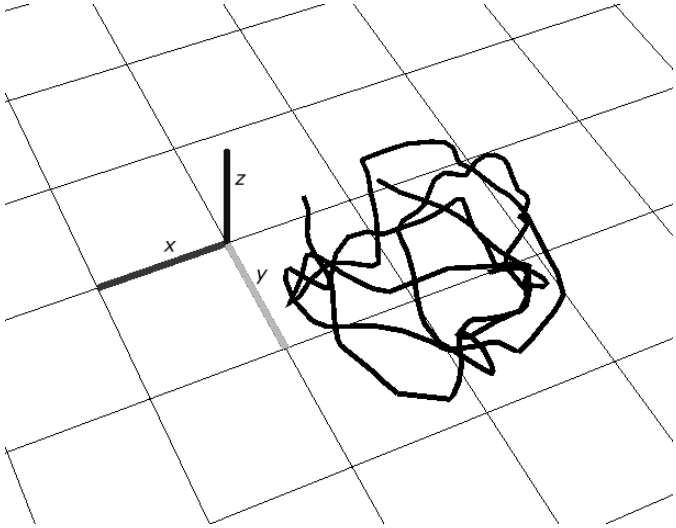


Figure 4.2: Visualization of a path flown for the occupancy grid mapping test created with the "rviz" tool. Grid squares are 1 meter wide. Axes show the origin and orientation of the world fixed frame.

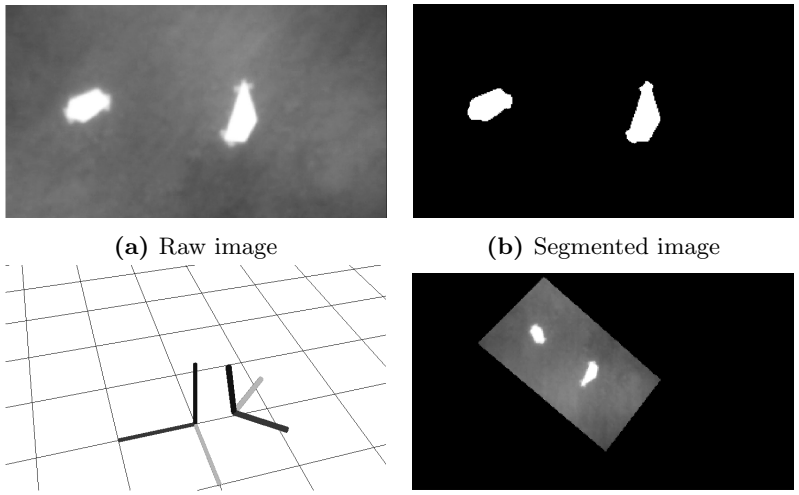
with the aid of PhD candidate Frederik Leira it was interfaced with DUNE via TCP. Once the ROS laboratory setup had been interfaced with DUNE, the path planning framework was tested in real-time in the lab.

4.2 Svalbard

There was originally a planned excursion to Svalbard in April 2015 to test some of the methods presented in this paper. However, the excursion was canceled due to time constraints among the participants. The experiments would have been conducted together with NORUT in a limited airspace in Kongsfjorden. Kongsfjorden is a small fjord (ocean inlet) on the west coast of Spitsbergen. The fjord houses two glaciers, "Kronebreen" and "Kongsvegen", and it was expected that Kongsfjorden would have high ice densities in the period of the excursion due to seasonal melting of the ice glaciers. The fjord is approximately 10 km wide and 20 km long.

4.3 Feature detection in Eggemoen data

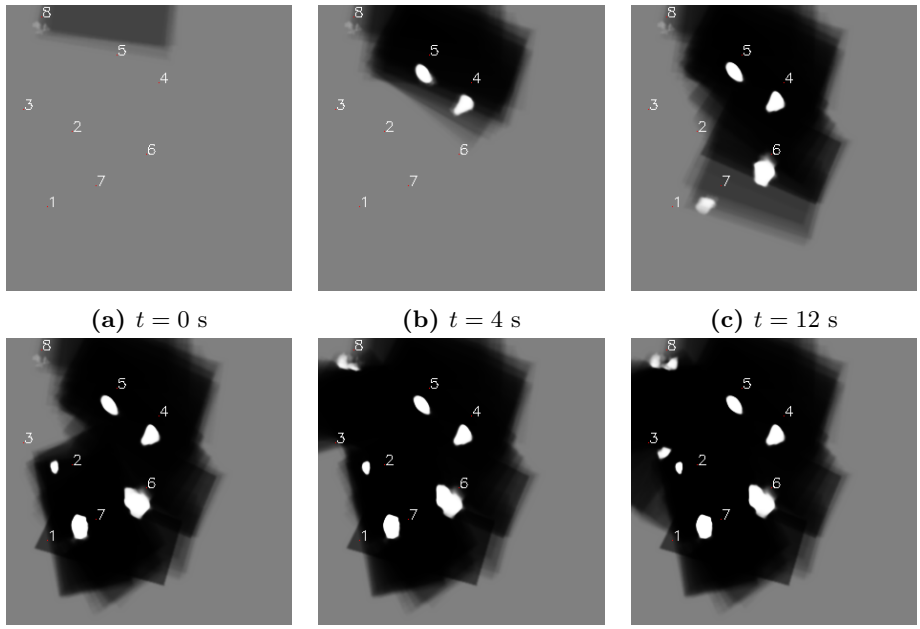
A flight test was performed at Eggemoen air base in February 2015 by PhD student Frederik Leira to evaluate the X8 experimental platform with an infrared camera. Video recorded on this test flight was used to evaluate if it would be possible to use the infrared camera to improve image Geo referencing by using feature detection and matching in OpenCV. Several feature detection methods implemented



(a) Raw image **(b)** Segmented image

(c) Pose **(d)** Perspective transform

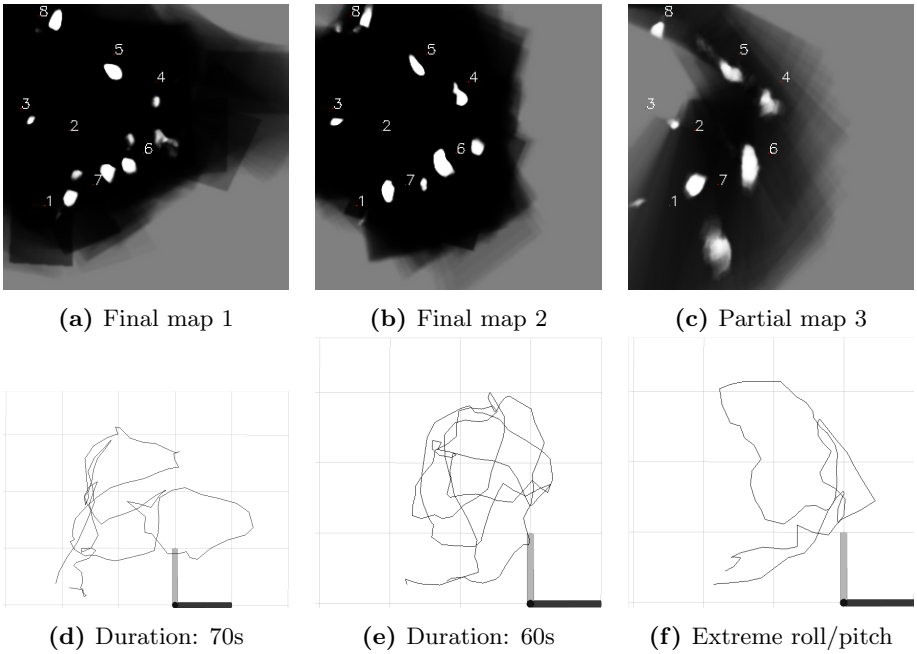
Figure 4.3: Showcase of essential parts of the mapping pipeline



(a) $t = 0$ s **(b)** $t = 4$ s **(c)** $t = 12$ s

(d) $t = 22$ s **(e)** $t = 28$ s **(f)** $t = 33$ s

Figure 4.4: Sequential estimation of the occupancy grid map corresponding to the dataset in Figure 4.5e



(a) Final map 1 **(b)** Final map 2 **(c)** Partial map 3

(d) Duration: 70s **(e)** Duration: 60s **(f)** Extreme roll/pitch

Figure 4.5: Paths flown in the three datasets and corresponding map estimates

in OpenCV were tested, and some metrics were evaluated to compare the performance of the different feature detectors. Since most feature detectors rely on textured environments with salient features present, it is not expected that the choice of feature detector will be critical in using these types of methods for localization. The evaluation will however shed some light on the case of using the image data from the X8 platform for localization. The video file used in the following experiments is a video recorded near the Eggemoen air field with a pilot manually flying the X8. Features that can be observed visually include trees, bushes, roads, buildings and large rocky formations. There is significant motion blur in many of the frames, and many frames seem to be dropped. The frame dropping is likely due to an unreliable data link between the video capture and recording equipment.

4.3.1 Video preprocessing

A minimal amount of video preprocessing was done in OpenCV before analyzing feature extraction and matching in the image sequence. It was noticed by visual inspection that the video file contained a lot of duplicate frames, which could lead to highly biased results later on in the analysis. It is not known why the video contains duplicates, but a possible reason is transmission latency in the video capture pipeline. To detect duplicate images, a few different methods were tested. Ideally it should be possible to discriminate duplicates and non-duplicates easily. First, a standard 2-norm was computed for each consecutive image pair. The

value of the image difference norm was manually inspected for a small part of the video. It was observed that even though images were clearly identical, the image difference norm was nonzero. Intuitively if images are in fact identical, the norm of the difference should be zero. Compression techniques may however introduce some noise in the image sequence. An alternative method was tested by computing the image histograms of consecutive image pairs and comparing them. Histograms can be compared using several different metrics in OpenCV. The method that yielded the best result was the Hellinger distance. The Hellinger distance is a metric to compare two probability distributions. If H_1 and H_2 are two image histograms, the Hellinger distance is computed with the following formula in OpenCV:

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_{I=1}^{N} \sqrt{H_1(I) H_2(I)}} \tag{4.1}$$

where $\bar{H}$ is the mean of the histogram bins and N is the total number of bins in the histogram. Both the image difference norms and Hellinger distance were computed for a sequence of 7000 consecutive image pairs from the Eggemoen video. The resulting histograms of image norms and Hellinger distances is shown in Figure 4.6. Ideally there should be some threshold value of the norm where duplicates and non-duplicates are easily separated. Even though the histogram of distance norms shows a clear peak where most of the duplicates are likely located, it is not very clear where this threshold should be located. The Hellinger distance was chosen as a better alternative since the histogram shows a more clear separation. A threshold distance of $h = 0.03$ was used to classify an image as a duplicate and is skipped in further processing. Visual inspection of some of the video sequence showed clear separation by Hellinger distance when the images were not duplicates.

4.3.2 Feature detection

Some of the implemented feature detectors in OpenCV were tested on the video sequence. For each detector some metrics were calculated to evaluate their performance. The most important part of the feature detection process is the ability to reliably detect and match features to corresponding features in another image. It is desirable to evaluate the possibility of performing image alignment with a minimal number of assumptions. Since features observed from the air are relatively planar with respect to each other, one method to evaluate if there is a successful set of correspondences between two images is to attempt to estimate a homography between the images. The following procedure was used for each image in the sequence to estimate a homography between the current and the previous image:

1. Check if image is duplicate as defined in the previous section, skip if duplicate criteria is met
2. Detect keypoints
3. Extract descriptors for keypoints

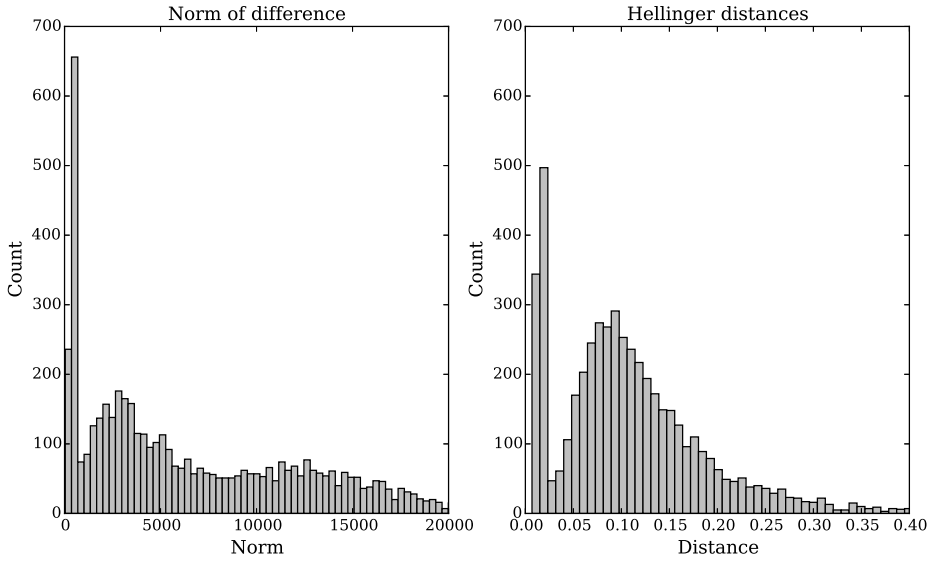


Figure 4.6: Histograms showing separation of duplicate images

4. Match keypoints to keypoints from the previous frame using a brute force search

5. Remove matches that have a distance higher than $2*minDist$ where $minDist$ is the minimal distance among the matches

6. If the number of matches is at least 4, estimate a homograpy robustly using a RANSAC method (Fischler and Bolles 1981)

To keep things simple it is assumed that if a homography cannot be estimated with sufficient accuracy (i.e. low enough reprojection error (Hartley and Zisserman 2004)), then the correspondence problem is not solved. A correspondence problem is assumed solved (i.e. a successful feature match) if the average reprojection error is less than 5 pixels. An example of an image pair with detected features and robustly estimated correspondences is shown in Figure 4.8. Standard methods in OpenCV are used to detect features, extract feature descriptors and match features. To perform matching a brute force search is used for simplicity and since computational speed is only interesting as a relative measure between the different detectors. Unless otherwise noted, standard or recommended parameters for feature descriptors, extractors and matchers are used. A summarized table of results for a sequence of 4119 images taken from the Eggemoen video is shown in Table 4.2. In addition, a graphical comparison of the feature matching and homography estimation results is shown in Figure 4.7. The computations were done on a standard desktop computer with an Intel i5 3.3 GHz processor with 8 GB RAM and an AMD Radeon HD 6450 GPU using OpenCV version 2.4.10. The

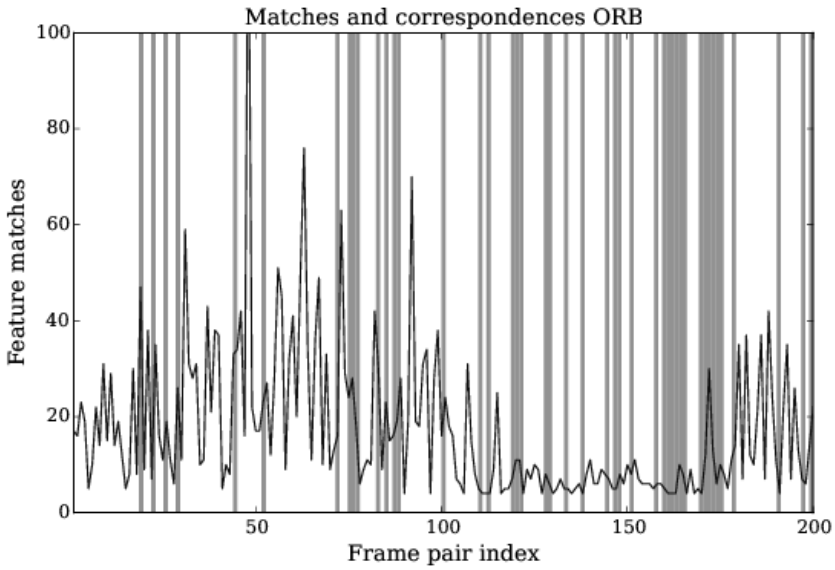


Figure 4.7: Plot showing number of feature matches and assumed successful feature matches for a subset of the Eggemoen video. A successful feature match is indicated by grey vertical bars.

results of the experiment show that SURF performs best out of the three detectors, but also has the longest mean computation time. It is encouraging to see that the recently developed ORB detector is about 10 times faster than the traditional SIFT and SURF detectors, and it performs almost as good as the SURF detector when it comes to homography estimation.

4.4 ODROID-XU3/U3

To enable on-board real time computer vision, two potential computing platforms were purchased early in the project work. The platforms were chosen after consulting PhD student Frederik Leira and measuring the available space in the UAV payload compartment. The ODROID-XU3 has eight available ARM processor cores and an OpenCL 1.1 "Full Profile" enabled Mali GPU from ARM. The ODROID-U3 is slightly less powerful, with a four available ARM cores and an older Mali GPU without OpenCL support. The platforms were tested on arrival to check its applicability for doing on-board computer vision. The platform arrived with a pre-built Linux image running Ubuntu 14.04 on a 32 GB MMC card. OpenCl drivers were included in this image. While the U3 is smaller and lighter, it is significantly less powerful. The U3 may still be more than powerful enough for the planned operation

[1] An image is considered matched if at least one feature from the previous image is matched.

	Num. images	Matched[1]	Homographies	Mean comp. time
ORB	4119	2091	387	1.75×10^{-2} s
SIFT	4119	1926	268	1.75×10^{-1} s
SURF	4119	3431	434	1.95×10^{-1} s

Table 4.2: Feature match and homography estimation comparison for an image sequence of length 4119

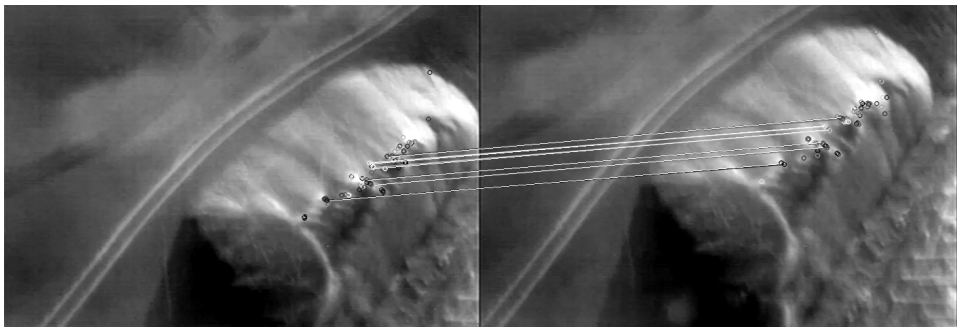


Figure 4.8: Visual example of detected features (circles) and feature matches (lines) between two images in sequence

on Svalbard, and have a lower power consumption. On arrival the platforms were tested to see if the required software for the planned operation could run on the platforms. Both platforms arrived with a pre built Ubuntu 14.04 Linux image, this was not changed before running any tests. The two most important software frameworks that are necessary for computer vision applications in the NTNU UAV lab are DUNE and OpenCV. DUNE was successfully compiled by getting the newest source code from *DUNE* 2015 and built on the targets. For the OpenCV initial testing it would be interesting to see if OpenCL acceleration would work "out of the box" on the ODROID XU3. To test this, the newest release of OpenCV was fetched from *OpenCV* 2015 (version 3.0.0-dev). In this version of OpenCV, many algorithms are implemented using both OpenCL and native CPU in a transparent API where OpenCL is chosen and used at run-time if it is available. Unfortunately, after a lot of testing it seemed like the OpenCL drivers produced a lot of errors in the official OpenCV performance tests. After some investigation a bug report revealed that there might be insufficient OpenCL support at the moment for the Mali T628 GPU *Mali OpenCl bug* 2015, leading to the conclusion that OpenCL acceleration is not available at the moment of writing this thesis. This gives the XU3 a slightly smaller theoretical advantage over the U3.

	Dim	**Mass**	**CPU(s)**	**GPU**	**Memory**	**Price**
XU3	70x90x18 mm	> 78g	Octacore	Mali T628	2 GB	179 USD
U3	83x48x? mm	48g	Quadcore	Mali 400	2 GB	65 USD

Table 4.3: Purchased platforms for on-board image processing, data from *Hardkernel* 2014

Chapter 5

Discussion

5.1 Occupancy grid mapping

The presented algorithm for occupancy grid mapping was tested in a relatively realistic scenario for the originally planned Svalbard excursion. While overall results are intuitively appealing, the accuracy and robustness is not convincing. The main source of error in the laboratory setup is in the unknown and varying delay of the image stream relative to the position data, leading to a potentially gross violation of the assumption that the pose is perfectly known together with the image measurement. This is problem since even if the accuracy of the pose measurement from the motion capture system is high, the time-alignment between position and image measurements is critical. The main reason this is a problem is that the image data is streamed over an unreliable wireless connection. The problem will perhaps be smaller on the actual intended platform for experiments on Svalbard, where the camera system is connected to the onboard computer through en Ethernet cable. The synchronization problem would also be a much smaller issue if the images were taken a known time, since buffering either images or pose estimates and delaying map estimates a few seconds is not really an issue. Controlling the image shutter is thus a possible solution to the problem, but is dependent on the camera equipment used.

Another assumption that was made was that camera fixed coordinate frame was related to the motion capture reference frame via a simple rotation about the x-axis. While the true transformation is close to this, there is a significant translation and probably also errors in the rotational alignment. This could be improved by careful calibration, but that would not be a viable solution in a practical system. A realistic way of coping with this in a practical system is to try to estimate this transformation from sensor data as a calibration procedure. A method for estimating the transformation between an IMU and a camera system in a Kalman Filter framework, known as inter sensor calibration is described in the PhD thesis of SM Weiss 2012. A similar method may be required to achieve satisfactory results.

5.2 Using image data for localization

Experiments suggested that using only the on-board navigation system for mapping might not yield accurate enough results for practical use. There is also an issue with aligning state measurements with the images captured, since the camera shutter is not necessarily controlled manually. Using the image data to improve state estimates, or at least to perform image alignment, might be an option to improve the mapping accuracy. This leads to the well known problems of vision-based monocular SLAM and the related SfM. This is the main motivation for including a short analysis of relevant methods for camera based navigation in Chapter 4. Using image data for navigation and alignement of image data relative to a global reference frame is an active and relatively mature research topic. Groundbreaking work in monocular SLAM was done by Andrew J Davison et al. 2007, enabling highly dynamic robust navigation and augmented reality applications in small environments in real time ($\sim$ 30Hz). The work in Andrew J Davison et al. 2007 is essentially an EKF using a camera motion model to aid feature matching in a new image frame. An EKF is recursive by nature, only using the newest measurement to improve the current state estimate. Experiments done in Strasdat, Montiel, and Andrew J. Davison 2010 imply that using a bundle adjustment type algorithm instead of a recursive Kalman Filter is both more accurate and less computationally intensive. An implementation of a real-time SfM can be found in Mouragnon et al. 2009. A similar approach motivated by augmented reality applications is shown in the much cited paper Klein and Murray 2007, they call their approach PTAM (Parallell Tracking and Mapping). There are a few open source PTAM implementations available, including by the original authors under a GPL licence. The recently announced *Project Tango* by Google states that it solves its bundle adjustment and SLAM problems using optimization methods similar to those in Mouragnon et al. 2009; Klein and Murray 2007, i.e. not recursively. Another interesting publication similar to the PTAM approach is shown in Forster, Pizzoli, and Scaramuzza 2014, which is also available as an open source implementation and has been tested on an embedded ARM platform similar to the one purchased for the experimental platform developed in this thesis. Some of the methods presented are suitable for large scale outdoor applications, and make no assumptions about scene geometry. It has already been noted that the scenes viewed in this application will approximately be planar. Image features in sequences are thus related by a homography, and this fact could perhaps be used to simplify or robustify SLAM approaches. Possible issues with using images for localization in this application include that detected keypoints are inherently non stationary since they are located at sea, and that images will often have very little texture when the scene viewed is highly homogeneous (e.g. pure ice or pure sea water). Then again, homogeneous regions are not that interesting and require less accuracy in mapping for the purposes of an ice monitoring system.

5.3 Potential challenges using an infrared camera

In Eik 2008 it is mentioned that one of the main challenges using traditional fixed wing aircraft for ice observation is the amount of fog experienced in the areas where sea ice can be a threat. Fog reduces visibility significantly and can challenge the use of optical sensors such as cameras. In this project experiments on Svalbard using an UAS equipped with an infrared camera were planned. The infrared camera may also be sensitive to weather effects such as fog. Since no experiments were carried out outdoors, it is hard to tell whether this would be a large problem.

Chapter 6

Conclusions and further work

In this thesis an algorithm for recursive ice density map estimation has been developed. The map has been represented as an occupancy grid map, and a camera together with a pose sensor has served as the input to the map estimation algorithm. To test the map estimation algorithm and to prepare for an excursion to Svalbard, an indoor laboratory environment was developed in ROS. The laboratory environment consists of an AR Drone 2 quadcopter with a camera facing towards the ground. The position and orientation of the quadcopter is recorded with a motion capture system. An autopilot was developed to enable the quadcopter to fly autonomously relative to the motion capture system. The same laboratory environment was used to test a path planning framework intended to be used for ice berg monitoring developed by Anders Albert. The path planning framework is implemented in MATLAB, and interfaced to the autopilot via DUNE and MAVLINK. Experiments done to test the mapping algorithm showed relatively poor estimation quality. A key assumption made in the mapping algorithm is that the pose is perfectly known, while this was thought to be the case in the lab it is certainly not the case on-board an UAS. The main issue in the lab was to time-align the image sequence with the pose estimates, and to calibrate the camera frame with the pose estimate frame. In March the Svalbard excursion was canceled, prompting a change in the direction of the thesis work. This made it possible to explore some methods to improve the map estimation algorithm. It is believed that a promising way to improve the estimates is to use structural information in the image sequence obtained during flight. Many modern SfM or SLAM methods track keypoints in images to perform image alignment and navigation tasks. Some time was thus spent to analyze image sequences from the proposed Svalbard platform to see if it was possible to perform keypoint extraction and matching with a thermal camera. Qualitatively the video sequence that was analyzed was not very good, it contained significant motion blur and at times very little texture. Even so, there were significant parts of the video sequence that allowed reliable feature detection

and matching of pairs of images.

Future work in similar real time ice density mapping should include inter sensor calibration and the possible use of image data for localization of ice features. The assumption that occupancy grid cells are independent is without doubt wrong in practice. This deserves some attention, and could improve the amount of information that one can retrieve from the image data.

Appendix A

Appendix

A.1 Zip file

The appendix contains mostly information on software implementations that were done during the thesis work. In addition to this appendix, ther is an attached zip file to this master thesis. The zip file contains all code necessary to reproduce the results in this thesis. The contents of the zip file is divided into the folders listed in Table A.1.

Folder	Description
ros_optitrack_mavlink	The main ROS package., see Appendix A.3
optitrack	Calibration file for the motion capture system
featurematch	Code to produce the results in section 4.3
norut_segment	Code to do the image segmentation analysis in section 2.4
odroid	Code to reproduce the results in section 4.4
cam_calib_ardrone	Data and code to calibrate the camera for the AR Drone

Table A.1: The contents of the attached zip file

A.2 Code

Note: some code has been trimmed for readability (e.g. removed header guards). For the original source, see the accompanying zip file.

A.2.1 MAVLINK bridge class definition

```
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <tf/transform_listener.h>
#include <tf/transform_datatypes.h>
#include <cmath>
#include <string>
#include <sstream>
#include <chrono>
#define MAVLINK_DIALECT ardupilotmega
#include <mavconn/interface.h>

class MavlinkInterfaceNode
{
    public:
        MavlinkInterfaceNode();
        void run();
    private:
        ros::NodeHandle n;
        ros::Publisher cmdPub;
        std::string url;
        std::chrono::milliseconds boot_time;
        mavconn::MAVConnInterface::Ptr mavInterfacePtr;
        tf::TransformListener poseListener;
        std::string world_frame_ned;
        std::string world_frame_ecef;
        std::string body_frame;
        std::string world_frame_mocap;
        void recv_message(const mavlink_message_t *msg, uint8_t sysid,
     uint8_t compid);
        void send_heartbeat();
        void send_nav_controller_packet();
        void send_waypoint_ack();
        void send_local_pose(const tf::StampedTransform& transform,
    uint32_t time_msec);
        void send_global_position(const tf::StampedTransform&
    transform, uint32_t time_msec);
};
```

code/mavlink_interface.h

A.2.2 Motion capture state transform node

```
#!/usr/bin/env python
# ROS node to get 6-dof state from the mocap_optitrack driver
# and send it to the ''tum_ardrone'' ardrone autopilot
# Calculates velocities via finite difference using lookupTwistFull
import rospy
import tf
from geometry_msgs.msg import PoseStamped
from tum_ardrone.msg import filter_state
from tf.transformations import euler_from_quaternion
from math import pi

def getFilterState(pose, twist):
    pos = pose[0]
    orient_quat = pose[1]
    vel = twist[0]
    orient = map(lambda r : 180*r/pi,
            euler_from_quaternion(orient_quat, axes='szyx'))
    state = filter_state(x = pos[0], y = pos[1], z = pos[2],
            yaw = orient[0], pitch = orient[1], roll = orient[2],
            dx = vel[0], dy = vel[1], dz = vel[2],
            ptamState = filter_state.PTAM_BEST, scaleAccuracy=1)
    return state

def publishState(pose, twist, pub):
    state = getFilterState(pose, twist)
    pub.publish(state)

def listener():
    rospy.init_node('state_listener')
    statePublisher = rospy.Publisher('/ardrone/predictedPose',
            filter_state, queue_size=10)
    tfListener = tf.TransformListener()
    # Frame names are configured in the ''mocap_optitrack'' driver
    world_f = '/world'
    ardrone_f = '/Robot_1/base_link'
    rate = rospy.Rate(60)
    while not rospy.is_shutdown():
        try:
            # get pose
            pose = tfListener.lookupTransform(world_f, ardrone_f,
                    rospy.Time(0))
            # get body velocities
            twist = tfListener.lookupTwistFull(ardrone_f, world_f,
                    ardrone_f, (0,0,0), ardrone_f,
                    rospy.Time(0), rospy.Duration(1.0/30))
            publishState(pose, twist, statePublisher)
        except (tf.LookupException,
                tf.ConnectivityException,
                tf.ExtrapolationException):
            continue
        rate.sleep()
if __name__ == '__main__':
    listener()
```

code/mocap_state_transform.py

A.2.3 Ice mapping class definition

```
#include <string>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/core/core.hpp>

class CameraModel {
  public:
    // file format is an openCV camera calibration XML/YAML file
    CameraModel(const std::string& paramFilename);
    const cv::Matx33f getCamMatrix() { return camMatrix_; };
    const cv::Mat getDistortionCoeffs() { return distortionCoeff_; };
    void undistortImage(const cv::Mat& input, cv::Mat& output);
  private:
    cv::Matx33f camMatrix_;
    cv::Mat distortionCoeff_;
};

/* The grid map is a planar occupancy grid map with a finite number of
     grid cells. The occupancy probabilities are represented in log-
    odds form.
 * I.e. map_(i,j) = log(P(grid cell i,j occupied)/P(grid cell i,j not
    occupied)).
 * A value of zero means equal probability, while a value of infinity
    implies that P(grid cell i,j occupied) = 1 */
class GridMap {
  private:
    // metric grid size in the x direction (meters)
    double dx_;
    double dy_;
    // Map grid width (num cells in the x direction)
    uint32_t width_;
    uint32_t height_;
    // map origin relative to a world fixed frame, expressed in the
    world frame
    cv::Point3f originWorldMap_;
    // map rotation matrix from a world fixed frame to the map frame
    cv::Matx33f dcmWorldMap_;
    // transformation from planar homogeneous map coordinates (u',v
    ',1) metric map coordinates (x_m,y_m,z_m)
    cv::Matx33f mapMatrix_;
    // grid map representation in log odds form
    cv::Mat map_;
  public:
    static const int MAP_DTYPE = CV_32F;
    GridMap(const uint32_t width, const uint32_t height, const double
    dx, const double dy,
        const cv::Point3f origin, const cv::Matx33f dcmWorldMap) :
      dx_(dx), dy_(dy), width_(width), height_(height),
      originWorldMap_(origin), dcmWorldMap_(dcmWorldMap),
      mapMatrix_(dx, 0, 0.5*dx,
          0, dy, 0.5*dy,
          0, 0, 0),
      map_(cv::Mat::zeros(width, height, MAP_DTYPE)) {};
    uint32_t getWidth();
    uint32_t getHeight();
    cv::Size getSize();
```

```
    cv::Point2i toMapCoordinates(cv::Point2f p_map);
    cv::Point3f toMapFrame(cv::Point3f pointWorld);
    void addMeasurement(const cv::Mat measurement);
    /* Returns a matrix that transform a map homogeneous coordinate
     * (u',v',1) to a world fixed coordinate (x_w, y_w, z_w) */
    const cv::Matx33f getMapWorldTransform();
    const cv::Mat& getMap() { return map_; };
};

// Simple struct to represent ground-truth ice features in the world
    frame
struct IceFeature {
  IceFeature(float x, float y, float r) : r(r), posWorld(x, y, 0.0)
    {};
  cv::Point3f posWorld;
  float r;
};

class RecursiveMapEstimator {
  public:
    RecursiveMapEstimator(const std::string& paramFilename) :
      gridMap_(), camModel_(paramFilename) {};
    // Update map estimate based on a segmented image. Expects a
    binary segmented image and camera pose as input
    void updateMap(cv::Mat& segmentedImage, const cv::Matx33f&
    dcmCamWorld, const cv::Point3f& originCamWorld);
    // Extra functions for testing
    void projectImageToMap(const cv::Mat& input, cv::Mat& output,
    const cv::Matx33f dcmCamWorld, const cv::Point3f originCamWorld);
    void getProbMap(cv::Mat& output);
    void drawFeaturesOnMap(std::vector<IceFeature> features, cv::Mat&
    output);
  private:
    GridMap gridMap_;
    CameraModel camModel_;
    const cv::Matx33f getPerspectiveTransform(const cv::Matx33f
    dcmCamWorld, const cv::Point3f originCamWorld);
    void toLogOdds(cv::Mat& inputProb, cv::Mat& outputLogOdds);
    void toProbability(cv::Mat& inputLogOdds, cv::Mat&
    outputProbability);
    void segmentedToProb(cv::Mat& segmentedImage, cv::Mat&
    detectionProb);
};
```

code/gridmapping.hpp

A.2.4 Autopilot patch

```
diff --git a/src/autopilot/ControlNode.cpp b/src/autopilot/ControlNode
    .cpp
index 6a6babd..e9a7501 100644
--- a/src/autopilot/ControlNode.cpp
+++ b/src/autopilot/ControlNode.cpp
@@ -413,10 +413,10 @@ void ControlNode::toogleLogging()
 void ControlNode::sendControlToDrone(ControlCommand cmd)
 {
         geometry_msgs::Twist cmdT;
-        cmdT.angular.z = -cmd.yaw;
+        cmdT.angular.z = cmd.yaw;
         cmdT.linear.z = cmd.gaz;
-        cmdT.linear.x = -cmd.pitch;
-        cmdT.linear.y = -cmd.roll;
+        cmdT.linear.x = cmd.roll;
+        cmdT.linear.y = cmd.pitch;

         // assume that while actively controlling, the above for will
    never be equal to zero, so i will never hover.
         cmdT.angular.x = cmdT.angular.y = 0;
diff --git a/src/autopilot/DroneController.cpp b/src/autopilot/
    DroneController.cpp
index e07575e..d1736ca 100644
--- a/src/autopilot/DroneController.cpp
+++ b/src/autopilot/DroneController.cpp
@@ -138,11 +138,11 @@ void DroneController::calcControl(TooN::Vector
    <4> new_err, TooN::Vector<4> d_err

         // rotate error to drone CS, invert pitch
         double yawRad = yaw * 2 * 3.141592 / 360;
-        d_term[0] = cos(yawRad)*d_error[0] - sin(yawRad)*d_error[1];
-        d_term[1] = - sin(yawRad)*d_error[0] - cos(yawRad)*d_error[1];
+        d_term[0] = cos(yawRad)*d_error[0] + sin(yawRad)*d_error[1];
+        d_term[1] = - sin(yawRad)*d_error[0] + cos(yawRad)*d_error[1];

-        p_term[0] = cos(yawRad)*new_err[0] - sin(yawRad)*new_err[1];
-        p_term[1] = - sin(yawRad)*new_err[0] - cos(yawRad)*new_err[1];
+        p_term[0] = cos(yawRad)*new_err[0] + sin(yawRad)*new_err[1];
+        p_term[1] = - sin(yawRad)*new_err[0] + cos(yawRad)*new_err[1];


         // integrate & cap
```

code/tum_ardrone.diff

A.3 ROS package documentation

Note: this section is a converted "README" markdown file from the ROS package used in the laboratory setup described in this thesis. The complete source code for this package is included in the accompanying zip file.

General Information

This package provides a MAVLINK interface to the Natural Point OptiTrack motion capture system at the "Slangerobotlab" in room B333. It also provides an interface to the AR Drone via MAVLINK. The package can be used to control the AR Drone with DUNE/Neptus. It has currently been tested to follow "goto" commands successfully. The package is implemented as an ROS package and has dependencies to a few other ROS packages. To build and use the package you need to have ROS installed. For installation instructions see http://wiki.ros.org/.

Build instructions

The package has been developed using the *indigo* ROS distribution in the catkin (http://wiki.ros.org/catkin) build environment, but in principle should build and run on any ROS distribution newer than this. The only build dependency not included in the standard ROS libraries is the *libmavconn* package. If you are running Ubuntu this package is most likely available in your added ROS repositories. To install it, run

```
sudo apt-get install ros-indigo-libmavconn
```

Additionally, all nodes in this package require an OptiTrack motion capture driver to be running, this can be found in the *mocap-optitrack* ROS package. As of writing this README the package was not available in the indigo repositories. It can be built from source by adding the package to your catkin workspace. To add the OptiTrack driver package, run the following commands:

```
cd catkin_ws/src
git clone https://github.com/ros-drivers/mocap_optitrack.git
```

To build the packages, check out this repository to your catkin workspace. For help on how to set up a catkin workspace see `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`. With your catkin workspace setup, clone the repository into your workspace

```
cd catkin_ws/src
git clone git@uavlab.itk.ntnu.no:uavlab/ros_optitrack_mavlink.git
cd ../ && catkin_make
```

Motion capture and MAVLINK

There is a dedicated ROS launch file to start the motion capture driver together with the MAVLINK publisher node. To launch and set the tcp address and listening port run the following command with the

```
cd catkin_ws source devel/setup.bash
roslaunch optitrack_mavlink mocap_mavlink.launch url:="tcp-l
  ://129.241.208.70:5760"
```

AR Drone autopilot

The autopilot has a few run dependencies. Depending on your environment, they might be available through your package manager. The ROS packages required are - ardrone_autonomy (AR drone driver) - tum_ardrone (AR drone autopilot) - joy (joystick control of the AR drone) - mocap_optitrack (motion capture driver, installation instructions already explained) All dependencies can be built from source, but many of them can be found in the official Ubuntu ROS repositories. To install these on Ubuntu run:

```
sudo apt-get install ros-indigo-ardrone-autonomy ros-indigo-joy
```

The tum_ardrone package was not available in the repositories as of writing this README. So this can be built from source by adding it to your catkin workspace by running the following commands

```
cd catkin_ws/src
git clone https://github.com/tum-vision/tum_ardrone.git
```

Unfortunately the *tum_ardrone* package includes a lot of unneccessary functionality in the form of computer vision algorithm implementations. This is not used in practice but it is still neccessary to build the entire package and link to its dependencies. To get the dependencies on Ubuntu run:

```
rosdep install tum_ardrone
sudo apt-get install freeglut3-dev
```

Where the *freeglut3-dev* package is needed to link to glut. If you are running Linux mint rosdep might fail to find the dependencies. You can fake your distribution by changing this to:

```
rosdep -os=ubuntu:trusty install tum_ardrone
```

Run the autopilot nodes

There is a launch file set up to run the autopilot. Simply run:

```
roslaunch optitrack_mavlink ardrone.launch
```

Bibliography

Baltsavias, Emmanuel P. (1999). "A comparison between photogrammetry and laser scanning." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 54.2-3, pp. 83–94.

Clausi, D.A. (2005). "Unsupervised segmentation of synthetic aperture Radar sea ice imagery using a novel Markov random field model." In: *IEEE Transactions on Geoscience and Remote Sensing* 43.3, pp. 528–538.

Davison, Andrew J et al. (2007). "MonoSLAM: real-time single camera SLAM." In: *IEEE transactions on pattern analysis and machine intelligence* 29.6, pp. 1052–67.

DUNE (2015). URL: `https://github.com/LSTS/dune` (visited on 02/04/2015).

Egeland, Olav and Jan Tommy Gravdahl (2002). *Modeling and Simulation for Automatic Control*. Marine Cybernetics AS, p. 639.

Eik, Kenneth (2008). "Review of Experiences within Ice and Iceberg Management." In: *Journal of Navigation* 61.04, p. 557.

Engel, Jakob, Jürgen Sturm, and Daniel Cremers (2014). "Scale-aware navigation of a low-cost quadrocopter with a monocular camera." In: *Robotics and Autonomous Systems* 62.11, pp. 1646–1656.

Fischler, Martin a. and Robert C. Bolles (1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." In: *Communications of the ACM* 24.6, pp. 381–395.

Foote, Tully (2013). "Tf: The transform library." In: *IEEE Conference on Technologies for Practical Robot Applications, TePRA*. IEEE, pp. 1–6.

Forster, Christian, Matia Pizzoli, and Davide Scaramuzza (2014). "SVO : Fast Semi-Direct Monocular Visual Odometry." In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 15–22.

Fossen, Thor I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, p. 596.

Goldberg, Steven B., Mark W. Maimone, and Lany Matthies (2002). "Stereo vision and rover navigation software for planetary exploration." In: *IEEE Aerospace Conference Proceedings* 5, pp. 2025–2036.

Hardkernel (2014). URL: `http://www.hardkernel.com` (visited on 12/01/2014).

Hartley, Richard and Andrew Zisserman (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Heikkila, J. and O. Silven (1997). "A four-step camera calibration procedure with implicit image correction." In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Comput. Soc, pp. 1106–1112.

Klein, Georg and David Murray (2007). "Parallel Tracking and Mapping for Small AR Workspaces." In: *IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE, pp. 1–10.

Levinson, Jesse, Michael Montemerlo, and Sebastian Thrun (2008). *Map-Based Precision Vehicle Localization in Urban Environments*. MIT Press, pp. 121–128.

Mali OpenCl bug (2015). URL: `http://code.opencv.org/issues/4010` (visited on 04/30/2015).

MAVLINK official site (2015). URL: `http://www.qgroundcontrol.org/mavlink/start` (visited on 04/30/2015).

Mouragnon, E. et al. (2009). "Generic and real-time structure from motion using local bundle adjustment." In: *Image and Vision Computing* 27.8, pp. 1178–1193.

OpenCV (2015). URL: `https://github.com/Itseez/opencv` (visited on 02/04/2015).

Otsu, N. (1979). "A Threshold Selection Method from Gray-Level Histograms." In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66.

Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System." In: *ICRA workshop on open source software*. Vol. 3. 3.2, p. 5.

Sadano, O, H Kawazoe, and M Shimakage (2002). "Lane following vehicle control and process." In: *US Patent 6,463,369*.

Schulz, D. et al. (2001). "Tracking multiple moving objects with a mobile robot." In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* 1, pp. 371–377.

Sezgin, Mehmet and Bülent Sankur (2004). "Survey over image thresholding techniques and quantitative performance evaluation." In: *Journal of Electronic Imaging* 13.1, p. 146.

Shi, Jianbo and Jitendra Malik (2000). "Normalized cuts and image segmentation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8, pp. 888–905.

Strasdat, Hauke, J. M M Montiel, and Andrew J. Davison (2010). "Real-time monocular SLAM: Why filter?" In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, pp. 2657–2664.

Thrun, Sebastian (2002). *Probabilistic robotics*. Vol. 45. 3. MIT Press, p. 647.

Urmson, Chris et al. (2009). "Autonomous driving in Urban environments: Boss and the Urban Challenge." In: *Springer Tracts in Advanced Robotics* 56, pp. 1–59.

Walt, Stéfan van der et al. (2014). "Scikit-image: image processing in Python." en. In: *PeerJ* 2, e453.

Weiss, SM (2012). "Vision based navigation for micro helicopters." PhD thesis. ETH.

Weiss, Stephan et al. (2013). "Monocular vision for long-term micro aerial vehicle state estimation: A compendium." In: *Journal of Field Robotics* 30.5, pp. 803–831.

Zhang, Zhengyou (2000). "A flexible new technique for camera calibration." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11, pp. 1330–1334.