AALBORG UNIVERSITY
DENMARK

Drone based 3D Area Localization

Master Thesis

Authors:

Beda Berner

Rahul Ravichandran

3rd June 2021

Attributions

This report was typeset using LaTeX

AALBORG UNIVERSITY
DENMARK

Department of Electronic Systems
Robotics

Fredrik Bajers Vej 7
9000 Aalborg
http://www.es.aau.dk

Title:

Drone based 3D Area Localization

Project Period:

Spring 2021

Participants:

Beda Berner
Rahul Ravichandran

Supervisor:

Anders la Cour-harbo

Number of Pages: 82
Date of completion: 03-06-2021

Preface

In recent times, drones are used in a wide range of applications. Many of these, such as wildfire or flood monitoring, include the classification of certain areas. Delivering drone images with these specific areas highlighted can provide a lot of information but leaves the association between the 2D image and the 3d world to the user. It would therefore be preferable to have a method that connects the information gained from the drone images to 3d positions. The authors were confronted with this challenge during their internship at Robotto, where they developed an algorithm to autonomously detect and map wildfire. During this internship, a simple projection from the drone image to an assumed ground plane was used to map the found fires. The accuracy of this method proved to be unsatisfactory. The aim of this master thesis is therefore to develop an improved method. For this purpose, an algorithm was developed that characterizes an area which is visible in many subsequent drone images as a 3d point cloud. It accomplishes this by tracking landmarks around the border of the target area and using Kalman filtering to find the 3d position of the individual landmarks. This algorithm was tested using state of the art simulation and showed promising results. It digests images subsequently and starts providing results as soon as possible. This means that, once optimized, it could run on the fly and provide input to the path planning of a drone. This differentiates it from alternatives, such as photogrammetry, that require the whole dataset to start processing.

B. Berner [illegible]

Beda Berner Rahul Ravichandran

Contents

List of Figures **vii**

List of Tables **x**

1 Introduction **1**

2 Problem Analysis **3**

2.1 Previous method 5

2.2 Related Works 9

3 Problem Formulation **10**

3.1 Research and Development Objectives 10

3.2 Requirements specification 11

3.3 Delimitation 11

4 Methodology **12**

4.1 Proposed Solution 12

4.2 Simulation 21

5 Implementation **30**

5.1 Data structure 31

5.2 Process new Image 35

5.3 Add Landmarks 40

5.4 Match and Estimate 41

5.5 Purge Landmarks 46

5.6 Challenges encountered 48

6 Analysis **51**

6.1 Initial results 52

6.2 Outlier analysis 53

6.3 Variation of drone speed 58

6.4 Variation of convergence threshold 59

6.5 Variation of AoI band width 60

6.6 Variation of Mahalanobis distance 61

6.7 Variation of downscaling factor 62

6.8 Best performing parameters 63

6.9 Cliff mission 65

6.10 Influence of static errors 67

6.11 Previous method test 70

7 Discussion **75**

8 Conclusion **78**

9 Future Works **79**

Bibliography **80**

List of Figures

2.1 Drone image of wildfire 3

2.2 Structure from Motion 4

2.3 Pin hole camera model 5

2.4 Limitation of the previous method 8

4.1 3D area localization process 12

4.2 Landmarks found in the area of interest 13

4.3 Kalman Estimation process 14

4.4 Comparison of EKF and UKF 15

4.5 Visualization of a descriptor 20

4.6 Gazebo and AirSim simulation environments 22

4.7 Data flow diagram between simulation and codebase 23

4.8 Coordinate Frame 24

4.9 ROS Architecture 25

4.10 Coastline in Unreal Editor 26

4.11 Coastline ground truth 27

4.12 Mission plan around lake 28

4.13 Cliff in Unreal Editor 29

4.14 Mission plan around cliff 29

5.1 Workflow of 3D area localization 30

5.2 Data structure of the Library of Landmarks (LoL) 32
5.3 Measurement of drone to target pitch and yaw angles 33
5.4 Process new Image flowchart 35
5.5 AirSim Segmented Image 36
5.6 Dilation operation 37
5.7 Process of how a band around the target area is generated 38
5.8 Comparison of feature detection with and without a mask 39
5.9 Flowchart of the Add Landmarks process. 40
5.10 Flowchart of the Match and Estimate process. 41
5.11 Feature matching procedure in MATCH block 43
5.12 Calculation of yaw (ψ_{img}) and pitch (θ_{img}) from the pixel coordinates. 44
5.13 Flowchart of the purging Landmarks process. 46
5.14 Unsynchronised Data collection 48
5.15 Effect of optimised synchronisation of data 49
5.16 Synchronisation of data using ROS ApproximateTime filter 49

6.1 3D estimates of landmarks 52
6.2 Extended Library of Landmarks 53
6.3 Outlier data plots 54
6.4 Inliers data plots 55
6.5 3D estimates of landmarks with base parameters 56
6.6 Histogram of distances achieved with base parameters 57
6.7 Speed graph 58

6.8 Convergence Threshold graph 59

6.9 Area of Interest (AoI) band width graph 60

6.10 Mahalanobis distance graph 61

6.11 Image resolution graph 62

6.12 3D estimates of landmarks with best performing parameters 63

6.13 Histogram of distances achieved with base parameters 64

6.14 Comparison of the histograms. y-axis in log scale. 65

6.15 Comparison between initial and best parameters in the cliff mission. 66

6.16 Yaw offset graph 67

6.17 3D plot yaw offset 68

6.18 Pitch offset graph 69

6.19 3D plot pitch offset 69

6.20 Location of ground reference points 71

6.21 Camera positions at 10 m, 30m and 50m in NED frame. 71

6.22 Data points generated from different camera positions 71

6.23 Prediction of marker locations at different camera positions. 73

7.1 Selection of AoI affecting the final boundary calculation 77

List of Tables

6.1 Baseline configuration for the parameters that will be varied. 56

6.2 Values of the parameters that will stay fixed 57

6.3 Results achieved with the base parameters 57

6.4 Configuration with the best performing parameters. 63

6.5 Results achieved with the best performing parameters 64

6.6 Results achieved with the best performing parameters 65

6.7 Markers placed on the field. 72

6.8 Offset corrections calculated by least squares optimisation. 72

6.9 Mean errors and standard deviation of original calculated data points and the optimised data points 74

Abbreviations

LoL	Library of Landmarks	viii
AoI	Area of Interest	ix
SfM	Structure from Motion	4
DEM	Digital Elevation Model	8
UAV	Unmanned Aerial Vehicle	9
EKF	Extended Kalman Filter	14
UKF	Unscented Kalman Filter	15
SIFT	Scale Invariant Feature Transform	18
ORB	Oriented FAST and rotated BRIEF	19
DoG	Difference of Gaussian	19
SITL	Software In the Loop	21
HITL	Hardware In the Loop	21
ROS	Robot Operating System	21
WGS 84	World Geodetic System 1984	24
NED	North-East-Down	24
RANSAC	Random Sample Consensus	42
RTK	Real Time Kinematic	70

1 Introduction

In this era of automation and aerial robots, some of the main tasks performed by drones are surveillance, mapping of areas and inspection. One of the big challenges in this context is connecting the information that is normally captured as a 2d image with the respective 3D position. While it is useful to see an image where wildfire is segmented and highlighted, this leaves the user to actually localize the area according to other features in the picture. Automating this process is not trivial. For real-time localization of single point targets, such as moving cars or persons, Kalman filtering has proven to be a good solution. This is widely researched, especially in the context of drone-based surveillance. For mapping static points or areas, Structure from Motion based photogrammetry can be used to get a very precise localization. This is a very mature process and a wide variety of commercial and open source tools are available. It is however very computationally expensive and needs to be provided with a complete dataset before computation can start. It can therefore not be run in real time on a drone and interact with the path planning. During the authors' internship at Robotto, they were introduced to the problem of fire area localization and mapping. They developed a simple algorithm which projects the captured images into an assumed ground plane (described in section 2.1). This process is very lightweight and can be run in real time on a drone. It does however, provide unsatisfactory accuracy even when tested on flat ground and would perform terrible in uneven areas.

This thesis proposes a new method to 3D localize areas. It leverages data from multiple subsequent images to gain information about the 3D position of landmarks that surround the area of interest. This solution was developed for feature sparse areas, but is equally valid for any area which can be detected and segmented. The main idea proposed is to create a band around the target area and detect features inside it. These features are tracked over time and their 3D position is estimated using Kalman filters.

This thesis is structured as follows. The theory of the algorithm developed at the internship and the projects done by other researchers in a similar field is explained in the Problem Analysis Chapter 2. In Chapter 3 Problem formulation, the major objectives that have to be accomplished throughout the project and the assumptions that were made, are listed. This is followed by Chapter 4 Methodology, in which the theory behind the developed solution is explained. Furthermore, it includes a description on how the simulation environment was chosen. Chapter 5 Implementation describes the software architecture and Chapter 6 Analysis describes the various tests that were conducted in order to find the best performing parameters. The subsequent Chapter 7 discusses the results obtained. Finally, the thesis ends with the conclusion and ideas for further research.

2 Problem Analysis

Drones are used in many applications such as wildfire monitoring, agriculture, or surveillance. They provide a top-down view of events on the ground that can greatly improve the user's situational awareness. For example fig. 2.1 shows aerial images of a wildfire. This can be very helpful for firefighters since fire and remaining hot spots are clearly visible, especially in the thermal channel. For this information to be of any use, the user has to know where the image contents are located. If he is very familiar with the area he is working in, he might be able to identify the location based on landmarks in the image he recognizes. Especially in an urban area with a high density of unique buildings, this is a valid option. In settings such as the one displayed in fig. 2.1, this approach is not viable since the landscape is too ambiguous. A method is needed to connect the information displayed in the image with an actual (ideally three-dimensional) position. This problem is known as target localization and good solutions exist for individual targets that are easily trackable such as cars or people. There are however use cases, in which the object of interest is not a point target but an area. Examples could be fires bigger than the one displayed in fig. 2.1, fields, flooded areas etc. In these cases, locating the area or the area borders is essential.

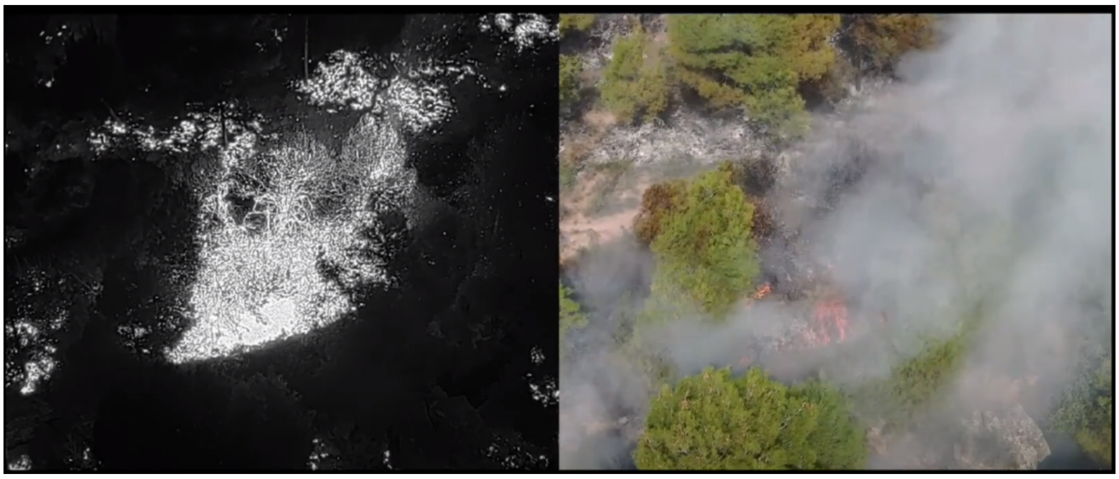


Figure 2.1: Image of a wildfire taken by a drone (DJI Matrice 300 using a Zenmuse H20T). The left side shows the Thermal channel which clearly displays the hot spots. On the right side the color image shows flames and smoke. While fire is clearly visible, locating this fire in the real world only based on this image would be very hard. Picture provided by Robotto.

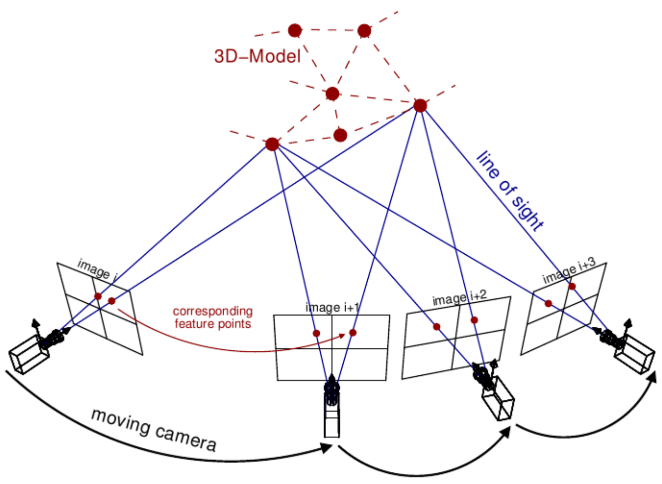


Figure 2.2: Structure from Motion (SfM) takes information from overlapping images taken from differnet points of view to create a 3D reconstruction of a subject. [1]

One of the most popular approaches used to locate areas is Structure from Motion (SfM) based photogrammetry. This process requires a high number of overlapping images of the area of interest. Based on regions that can be seen in multiple images and the movement of the camera between the image captures, a 3D reconstruction of the area can be created. If the original pictures are georeferenced, the reconstruction can be as well. This method is very mature and a wide variety of commercial software is available. It is very accurate and widely used in surveying and agriculture. Since SfM is computationally expensive and needs a high number of images before it can start processing, it is not suited to be run during a drone mission. This means it can not be used to influence the behaviour of an autonomous drone on the fly. Furthermore, it requires the area to stay consistent during the whole time that is needed to take the images. This makes this method unsuited for applications in dynamic environments such as wildfire.

During their internship at Robotto, the authors were confronted with the requirement for a method that can perform target localization of wildfires during the flight to influence the autonomous behaviour of the drone. They developed a simple method to tackle this problem, which is described in the following section.

2.1 Previous method

2.1.1 Idea

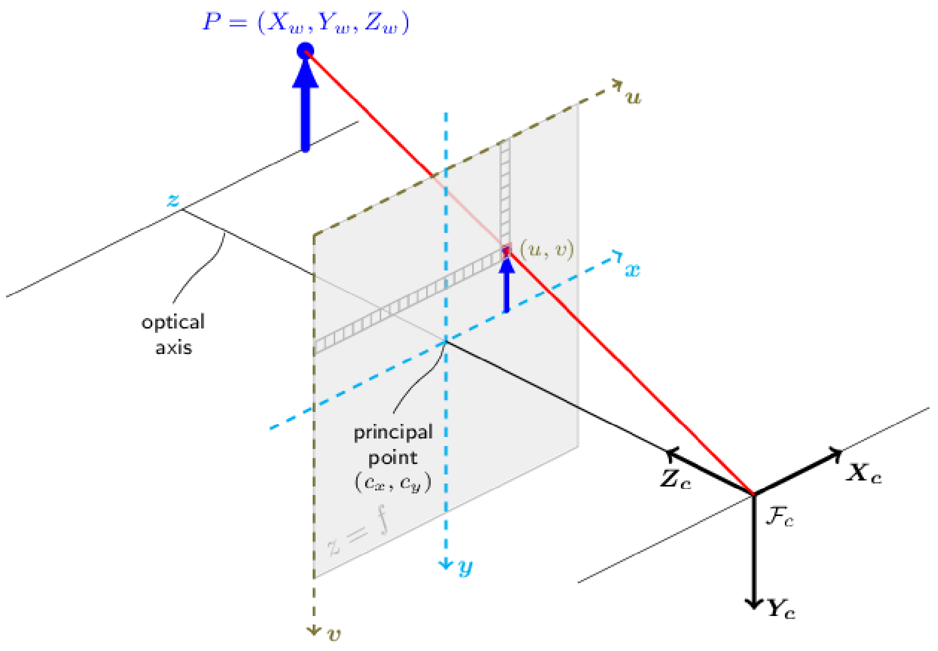


Figure 2.3: The pinhole camera model describes the image taking process. It allows the projection of a 3D point into the image plane given extrinsic and intrinsic camera parameters. This process is only reversible, if some information about the 3D point is known. For example if Z_w is known. [2]

The main idea of this approach is to invert the image taking process. The pinhole model [2] is a mathematical model of a camera. It describes how a 3D position is projected onto the image plane (a visual representation can be seen in fig. 2.3). If it could simply be inverted, any pixel of the image could be localized in 3D space. However, as can be seen in fig. 2.3, a pixel position on the image plane defines a ray of possible 3D positions. Since there is no depth information available, the 3D position associated with the pixel remains ambiguous. An assumption about the position has to be taken to solve the problem. If the world is assumed to be an entirely flat plane at a certain altitude, the intersection between the ray and the ground plane can be found, which will define the 3D position associated with the pixel. This position will at first be in relation to the camera frame. Since the camera's location and rotation are known, it can be transformed into a local or global reference frame.

2.1.2 Implementation

First, the transformations between the local coordinate system and the camera gimbal have to be defined. Vector $\boldsymbol{t}^{L\to G}$ defines the translation from local frame to the gimbal position:

$$\boldsymbol{t}^{L\to G} = \begin{bmatrix} G_x \\ G_y \\ G_z \end{bmatrix} \tag{2.1}$$

The rotation matrix $\boldsymbol{R}^{L\to G}$ defines the rotation from local frame to the current gimbal orientation. It can be separated into the static transformation from local to the gimbal origin pose $\boldsymbol{R}^{L\to G_0}$ and the dynamic transformation between gimbal origin and the actual gimbal orientation $\boldsymbol{R}^{G_0\to G}$.

$$\boldsymbol{R}^{L\to G} = \boldsymbol{R}^{L\to G_0}\boldsymbol{R}^{G_0\to G} \tag{2.2}$$

The rotation from local frame to gimbal origin position is equal to a rotation of 90° around z-axis followed by a rotation of 90° around x-axis:

$$\boldsymbol{R}^{L\to G_0} = \begin{bmatrix} \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) & 0 \\ \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\frac{\pi}{2}) & sin(\frac{\pi}{2}) \\ 0 & sin(\frac{\pi}{2}) & cos(\frac{\pi}{2}) \end{bmatrix} \tag{2.3}$$

Assuming the kinematic chain of the gimbal is yaw->pitch->roll like it is on the zenmuse H20T:

$$\begin{gathered} \boldsymbol{R}^{G_0\to G} = \boldsymbol{R}_{\text{yaw}}\boldsymbol{R}_{\text{roll}}\boldsymbol{R}_{\text{pitch}} \\ \boldsymbol{R}_{\text{yaw}} = \begin{bmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{bmatrix} \\ \boldsymbol{R}_{\text{roll}} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \boldsymbol{R}_{\text{pitch}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \end{gathered} \tag{2.4}$$

where ψ,ϕ and θ are the yaw, roll and pitch angles.

The transformations from gimbal frame to local frame $\boldsymbol{t}^G \rightarrow L$ and $\boldsymbol{R}^{G\rightarrow L}$ can now be derived easily:

$$\boldsymbol{R}^{G\rightarrow L} = (\boldsymbol{R}^{L\rightarrow G})^{-1} = (\boldsymbol{R}^{L\rightarrow G})^T \tag{2.5}$$

$$\boldsymbol{t}^{G\rightarrow L} = -\boldsymbol{R}^{G\rightarrow L}\boldsymbol{t}^{L\rightarrow G} \tag{2.6}$$

Using these transformations, the intrinsic matrix of the camera $\boldsymbol{A}$ and the pixel coordinates u,v the local coordinates of the point can be calculated

$$\begin{bmatrix} X_{\text{local}} \\ Y_{\text{local}} \\ Z_{\text{local}} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^{G\rightarrow L} & | & \boldsymbol{t}^{G\rightarrow L} \\ 0 \;\; 0 \;\; 0 & & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{A}^{-1} s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \\ 1 \end{bmatrix} \tag{2.7}$$

where s is a scaling variable which decides where on the ray the point is. As previously described the assumption is taken that Z_{local} is known.

$$\boldsymbol{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$\boldsymbol{R}^{G\rightarrow L} = \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} & \boldsymbol{R}_{13} \\ \boldsymbol{R}_{21} & \boldsymbol{R}_{22} & \boldsymbol{R}_{23} \\ \boldsymbol{R}_{31} & \boldsymbol{R}_{32} & \boldsymbol{R}_{33} \end{bmatrix} \tag{2.9}$$

s can then be calculated as:

$$s = \frac{(R_{13}R_{22}R_{31} - R_{12}R_{23}R_{31} - R_{13}R_{21}R_{32} + R_{11}R_{23}R_{32} + R_{12}R_{21}R_{33} - R_{11}R_{22}R_{33})f_x f_y (G_z - Z_{local})}{(R_{22}R_{31} - R_{21}R_{32})(u - c_x)f_y + f_x(-R_{12}R_{31}v + R_{11}R_{32}v + (R_{12}R_{31} - R_{11}R_{32})c_y + (R_{12}R_{21} - R_{11}R_{22})f_y)} \tag{2.10}$$

Applying s to eq. (2.7) now allows the determination of X_{local} and Y_{local}.

2.1.3 Limitations

While this method allows the localization of any pixel in a given picture, it is limited by the assumptions taken. In real life applications, the identified area is often not flat and its altitude is not known. The difference between the real altitude and the assumed altitude can lead to considerable errors as shown in fig. 2.4. This is especially true, if the gimbal pitch is shallow. The performance of this method has been evaluated as a part of the thesis in Section 6.11. Furthermore, this method only uses the information of one single picture. This has its own advantages and disadvantages. It allows for an instant localization information but also means

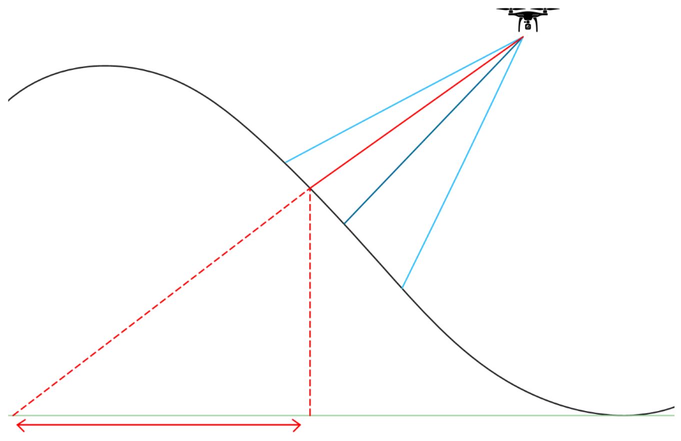


Figure 2.4: Any deviation from the assumed ground plane (light green) will lead to considerable errors in the localization of the point of interest (errors are displayed in dotted red). [3]

that no information from other pictures is used to minimize noise and improve localization accuracy. Merino et al. [4] use a similar method but get altitude information from a Digital Elevation Model (DEM) and calculate probabilities over multiple pictures to reduce these errors.

2.1.4 Expectations for the new method

The improvement of the previous method must overcome most of the limitations listed in the Section 2.1.3. The new method must not assume the location of the area to be in a flat plane. It should be able to calculate the 3D position of the pixel instead of 2D position on a plane. The new method must also use information from multiple views and not just a single image. Although this will not give immediate results, it should provide a much more accurate 3D estimation of the pixel.

2.1.5 Problems that may arise

To keep track of a certain pixel on the contour of an area, it must have good features associated with it. These can then be matched between frames to keep track of it. In some cases, such as uniform areas, good consistent features are not available. In these cases, feature matching is not possible and hence it is not possible to keep track of a pixel through multiple frames.

2.2 Related Works

A review of related literature showed that most of the authors only discuss the issue related to single point 3D target estimation and not an estimation of an area. Only Merino et al. [4] address the issue of 3D area localization. Although they follow a similar strategy to the one done in the internship, they improve the localization significantly by using information from multiple drones and digital elevation maps. Moreover, they also generate a probability map of the area location. The drones are equipped with infrared cameras to detect the fire contour without any hindrance from smoke. Each individual Unmanned Aerial Vehicle (UAV) estimates the fire front pixels and fire top pixels and sends it to a central decision system located in the ground control station. The position of the fire pixels changes more slowly at the front compared to the top of the fire. The characterisation of the top of the fire from the front is done by applying a low pass filter over a sequence of consecutive segmented images. The data from the fire area pixel points are then projected onto the ground plane by multiple drones and is used to create a probability map of fire on the ground plane. The results shown in this paper look promising, however the method used requires additionaly information in form of a Digital Elevation Model and multiple UAVs

The 3D Area localization can also be viewed as a series of 3D point target localization applied to static features in or around an area. Under this premise work on 3D point target localization will also give a lot of information on improving 3D area localization. A number of papers have discussed the 3D point target localization problem both for stationary[5][6] and moving[7][8] targets. Ponda et al. [9] and Wang et al. [10] discuss optimal trajectories during the drone flight to ensure the fastest convergence.

Especially interesting is "Unscented Kalman Filter for Vision-Based Target Localisation with a Quadrotor" by Dena and Aouf [5]. This paper uses Unscented Kalman Filtering to get an estimate of the 3D position of the target using the drone location , camera gimbal angles, and the intrinsics of the camera. The authors also suggest that the estimates observed are prone to bias due to noise in the measurements. They consider the target to be stationary, which is similar to the problem addressed in this thesis proposal. This makes the state matrix an identity matrix which is linear, but the target position is calculated using the gimbal angles of the camera, which makes the measurement model a nonlinear function of elevation and azimuth from the target. Due to this nonlinearity, the unscented Kalman filter is used. This estimates the nonlinear function better than an extended Kalman filter using sigma points.

3 Problem Formulation

From the analysis done on the algorithm developed during the internship, it was discovered that the method has its limitations. The most severe one is, it assumes that the area is flat. This leads to inaccurate 3D localization of uneven areas. Even when tested on flat ground, small deviations in the measured gimbal pitch angle lead to big errors (see section 6.11). Therefore, to overcome the limitations of the previous method, it is proposed to use feature tracking and state estimation to provide more accurate results (explained in detail in section 4.1). This leads to the following problem statement:

> *"Develop an algorithm which performs 3D area localization based on feature tracking and state estimation"*

3.1 Research and Development Objectives

To achieve the set problem statement, some basic goals were set to develop the base of the algorithm.

1. Choose a suitable simulation environment where feature detection can be performed.
2. Choose a suitable feature detector and matcher.
3. Add a target area to the simulation and extract its segmented image output.
4. Plan a drone mission to fly around the target area to capture images and the necessary meta data.
5. Develop a feature tracking algorithm which can track individual features.
6. Develop a state estimation algorithm which is able to estimate the 3D position of the tracked features.

In order to achieve the set goals, certain assumptions were made regarding the environment available:

1. The region around the target area is feature rich.
2. The segmented image of the target area is available beforehand.

3.2 Requirements specification

To meet the standards of real world applications, the final simulation results must meet the following requirements:

1. The feature detection algorithm must be able to detect and track static and consistent features throughout the mission.
2. The average distance of final localised features to the ground truth of the boundary of target area, must be within 5m.
3. The algorithm must be able to perform, even when noise is added to simulate real sensor data.
4. The developed algorithm must be scalable for different camera models, i.e. the algorithm should function equally well for reduced image resolution.

3.3 Delimitation

Due to the COVID-19 outbreak and the worldwide lockdown, most of this project has been limited to simulation. Access to the drone was possible only on limited days due to the restrictions active at the time in Denmark. The drone was therefore only used for early tests while later work focused on simulation.

4 Methodology

4.1 Proposed Solution

The objective of the thesis is to improve upon the 3D area localization that was developed as a part of the internship. As described in Chapter 2, the new method is expected to calculate the 3D area location and should fuse information gathered from multiple frames as seen in fig. 4.1.

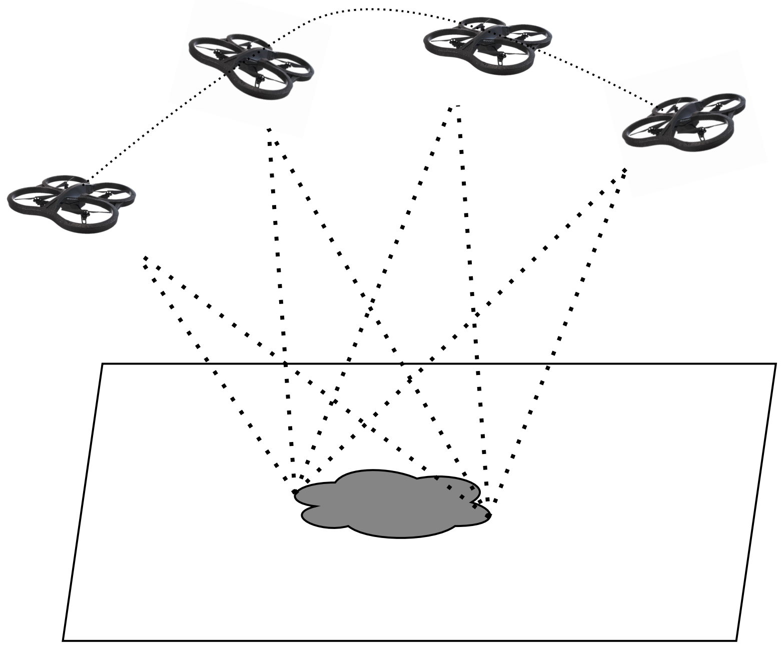


Figure 4.1: The drone views the area from multiple perspectives.This allows the estimation of an accurate 3D location for the area.

While exploring possible solutions from previous works on related fields as explained in Section 2.2, only one paper was found that tackles this specific problem. Merino et al. [4] solved the problem using a very similar approach to the one described in section 2.1. They further improved upon it by incorporating a Digital Elevation Model (DEM) and probability estimation. This relative lack of research motivated the authors of the thesis to propose a new idea for 3D area localization.

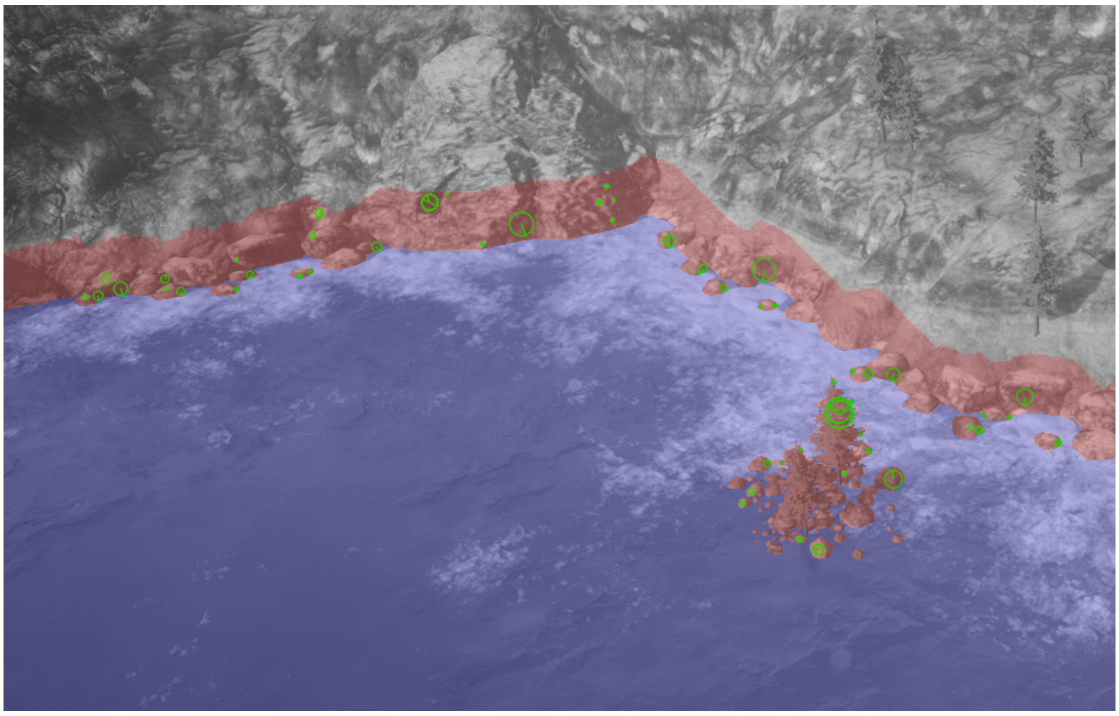


Figure 4.2: Based on the segmented target area (blue) a band around it is created (red). In this band notable landmarks are found (green) and tracked over subsequent images.

The proposed idea is to localize a detected area by searching for static landmarks around the area of interest. This is accomplished by applying a standard feature detector like SIFT [11] in a band around the target (see fig. 4.2). Tracking these landmarks over time and using state estimation such as Kalman filtering, accurate 3D positions for each landmark can be found. These can then be joined, to generate a good 3D estimate of the area of interest. The use of state estimation to localize single point targets is well documented in literature for both static [5] [6] and moving targets [7] [8]. The nonlinear nature of the bearing-only localization problem requires the use of nonlinear state estimation such as Extended or Unscented Kalman filtering.

This proposed idea is implemented as a proof of concept in simulation software to evaluate its viability.

4.1.1 Unscented Kalman Filter

The information gathered about a landmark over multiple frames has to be combined to improve the estimation of its position. This can be solved using a Kalman filter which is a two step process. First, there is a prediction step where the final mean and covariance of the state are predicted using a given model. Secondly, there is an update step where the state is measured using given sensors and their associated uncertainties. Combining predicted and measured state results in a new estimate with higher confidence(see fig. 4.3).

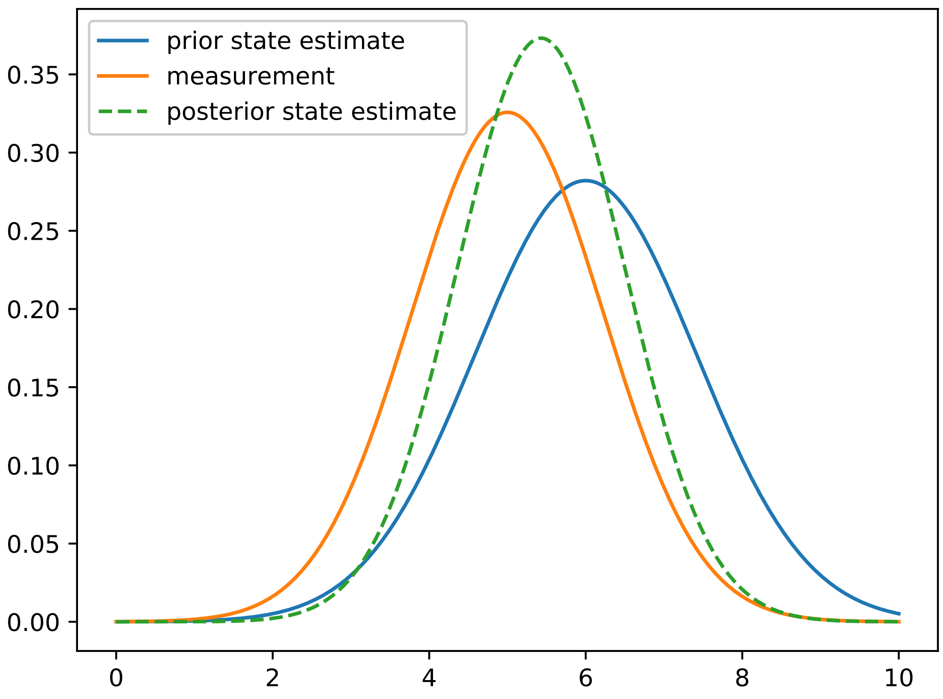


Figure 4.3: Kalman filter uses previous state estimate and the measurement to get a better estimate of the current state

While the position of the landmark is stationary and therefore linear, the measurements (yaw and pitch) are nonlinear. Therefore, a nonlinear Kalman filter has to be used. The most widely used nonlinear Kalman filter is the Extended Kalman Filter (EKF). It models nonlinear behaviour by linearizing around the current state and estimating a new Gaussian distribution.

Unscented Transform

The Unscented Kalman Filter (UKF) [12] improves upon this concept by applying nonlinearity to a low number of particles, the so-called sigma points. These are chosen to represent the original Gaussian distribution. These points are passed through a nonlinear function and the output is then used to generate a closer Gaussian distribution estimate (see fig. 4.4).

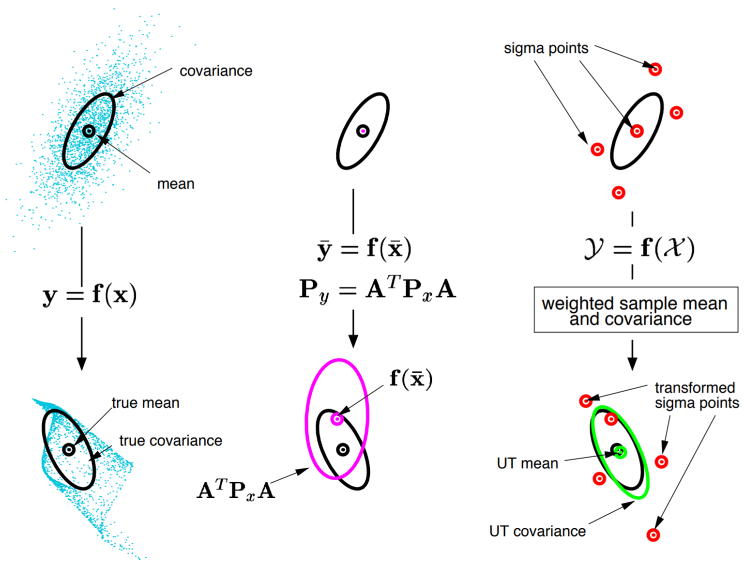

covariance
mean
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
true mean
true covariance
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
sigma points
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
transformed sigma points
UT mean
UT covariance

Figure 4.4: The above plots show the propagation of mean and covariance through nonlinear functions using different methods. The left plot shows the propagation of all the points of the state (Monte Carlo), the middle plot shows the propagation results after linearising the model (EKF) and the last plot shows propagation of only the sigma points to estimate the new mean and covariance (UKF). [12]

This so-called unscented transform is performed in a series of steps:

- Calculate the sigma points
- Assign weights to the sigma points
- Transform the points through a nonlinear function
- Compute the Gaussian distribution from the transformed points

The number of sigma points to choose depends on the dimensionality of the system. It is $2n + 1$, where n is the dimensionality of the system.

The sigma points and their weights are calculated based on Van der Merwe's 2004 dissertation [13]. The sigma points are calculated as follows:

$$\boldsymbol{U} = \sqrt{(n + \lambda)\boldsymbol{\Sigma}} \tag{4.1}$$

$$\begin{aligned} \boldsymbol{\mathcal{X}}_0 &= \boldsymbol{\mu} \\ \boldsymbol{\mathcal{X}}_i &= \boldsymbol{\mu} + \boldsymbol{U}_i \quad \text{for } i = 1, ..., n \\ \boldsymbol{\mathcal{X}}_i &= \boldsymbol{\mu} - \boldsymbol{U}_{i-n} \quad \text{for } i = n + 1, ..., 2n \end{aligned} \tag{4.2}$$

Where every $\mathcal{X}$ is one sigma point, μ is the mean of the Gaussian distribution, λ is the scaling factor which decides how far from mean the sigma points should be, and $\boldsymbol{\Sigma}$ is the covariance matrix which is a square matrix of size n. λ can be calculated as follows:

$$\lambda = \alpha^2(n + \kappa) - n \tag{4.3}$$

where α and κ are tuning parameters. α is normally chosen as a small positive value while κ is usually set to zero [12].

Weights for the sigma points are chosen such that the sum of all weights is equal to one and more weight is given to the points which are closer to the mean. These weights are calculated as follows:

$$\begin{aligned} \boldsymbol{W}_0^{\mu} &= \frac{\lambda}{n + \lambda} \\ \boldsymbol{W}_0^{\Sigma} &= \frac{\lambda}{n + \lambda} + (1 - \alpha^2 + \beta) \\ \boldsymbol{W}_i^{\mu} = \boldsymbol{W}_i^{\Sigma} &= \frac{1}{2(n + \lambda)} \quad \text{for } i = 1, ..., 2n \end{aligned} \tag{4.4}$$

where β is a tuning parameter which is normally set to 2 [12].

The resulting Gaussian after passing through the nonlinear function is then calculated:

$$\begin{aligned} \boldsymbol{\mu}' &= \sum_{i=0}^{2n} \boldsymbol{W}_i^{\mu} g(\mathcal{X}_i) \\ \boldsymbol{\Sigma}' &= \sum_{i=0}^{2n} \boldsymbol{W}_i^{\Sigma} g(\mathcal{X}_i - \boldsymbol{\mu}')(g(\mathcal{X}_i) - \boldsymbol{\mu}')^T \end{aligned} \tag{4.5}$$

Where, $\boldsymbol{\mu}'$ is the predicted mean, $\boldsymbol{\Sigma}'$ is the predicted covariance and $g : \mathbb{R}^n \to \mathbb{R}^n$ is the state transition function.

Prediction

A new state $\boldsymbol{x}[k+1]$ is predicted based on the old state $\boldsymbol{x}[k]$ using the state transition model:

$$\boldsymbol{x}[k+1] = \Phi(\boldsymbol{x}[k]) + \boldsymbol{w}[k] \tag{4.6}$$

Where $\boldsymbol{x}[k]$ is the prior state, $\boldsymbol{x}[k+1]$ is the current state, $\Phi(\boldsymbol{x}[k])$ is the state transition model between k and $k+1$, $\boldsymbol{w}[k]$ is the process noise.

Since Φ might be nonlinear, unscented transform as described in (4.1) to (4.5) is used to calculate the predicted mean ($\boldsymbol{\mu}'$) and covariance ($\boldsymbol{\Sigma}'$) of state $\boldsymbol{x}[k+1]$, where the function g is replaced with Φ.

Update

The measurement $\boldsymbol{z}[k]$ associated with state $\boldsymbol{x}[k]$ can be calculated using the measurement model:

$$\boldsymbol{z}[k] = h(\boldsymbol{x}[k]) + \boldsymbol{v}[k] \tag{4.7}$$

Where $h : \mathbb{R}^n \to \mathbb{R}^m$ is the measurement model, $\boldsymbol{v}[k]$ is the measurement noise and m is the dimension of the measurement.

Since h might be nonlinear, unscented transform as described in (4.1) to (4.5) is used to calculate the mean $\hat{\boldsymbol{z}}$ and covariance $\boldsymbol{S}$ of the predicted measurement as shown below.

$$\begin{gathered}
\boldsymbol{\mathcal{Z}_i} = h(\mathcal{X}_i) \\
\hat{\boldsymbol{z}} = \sum_{i=0}^{2n} \boldsymbol{W}_i^{\mu} \boldsymbol{\mathcal{Z}}_i \\
\boldsymbol{S} = \boldsymbol{V} + \sum_{i=0}^{2n} \boldsymbol{W}_i^{\Sigma} (\boldsymbol{\mathcal{Z}}_i - \hat{\boldsymbol{z}})(\boldsymbol{\mathcal{Z}}_i - \hat{\boldsymbol{z}})^T
\end{gathered} \tag{4.8}$$

Where, $\boldsymbol{\mathcal{Z}}$ is the transformed sigma points in the measurement space and $\boldsymbol{V}$ is a diagonal matrix which represents the measurement noise.

To calculate the error in prediction, it is necessary to calculate the cross-correlation between sigma points in state space and measurement space:

$$\begin{gathered}\boldsymbol{T}=\sum_{i=0}^{2n}\boldsymbol{W}_i^{\Sigma}(\boldsymbol{\mathcal{X}}_i-\boldsymbol{\mu}')(\boldsymbol{\mathcal{Z}}_i-\hat{\boldsymbol{z}})^T\\ \boldsymbol{K}=\boldsymbol{T}\boldsymbol{S}^{-1}\end{gathered}\tag{4.9}$$

Where $\boldsymbol{T}$ is the cross-correlation matrix between state space and the measurement space, $\boldsymbol{S}$ is the predicted covariance matrix and $\boldsymbol{K}$ is the Kalman gain. The final state and covariance are calculated incorporating the actual measurement $\boldsymbol{z}$ as follows:

$$\begin{gathered}\boldsymbol{\mu}=\boldsymbol{\mu}'+\boldsymbol{K}(\boldsymbol{z}-\hat{\boldsymbol{z}})\\ \boldsymbol{\Sigma}=\boldsymbol{\Sigma}'-\boldsymbol{K}\boldsymbol{S}\boldsymbol{K}^T\end{gathered}\tag{4.10}$$

4.1.2 Feature tracking

For the described Kalman filters to work, input in the form of pixel coordinates of the same landmark over several pictures has to be provided. For this purpose, a feature detection algorithm is used to find and track notable features over multiple images. Numerous feature extraction algorithms such as SIFT [11], SURF [14], ORB [15] and KAZE [16] exist. They vary widely in computational cost and accuracy depending on the use case. In this application, the feature detection is going to be used to track individual features. For this reason, high confidence in the matching is required since outlier removal with techniques like RANSAC is not possible. Tareen and Saleem [17] compared various feature detectors with multiple datasets. They show that Scale Invariant Feature Transform (SIFT) has generally the best matching performance and it is therefore used in this project. This comes at the cost of relatively high computational requirements. It is therefore important to limit the number of required features as much as possible.

Feature Extraction

The process of feature extraction can be split in 2 independent parts:

First, distinctive points inside the picture have to be found. This usually involves finding corners or extrema of some sort. The resulting keypoint contains information about the features location and its strength.

In a second step, a descriptor for the found keypoint is calculated. This is achieved by describing the area around the point in some way that can later be compared to other descriptors.

Feature extractors usually consist of both keypoint detection and description. However, since the 2 processes are independent, they can also be mixed. Oriented FAST and rotated BRIEF (ORB) is as the name implies, a mixture of keypoint extractor and descriptor of 2 other algorithms.

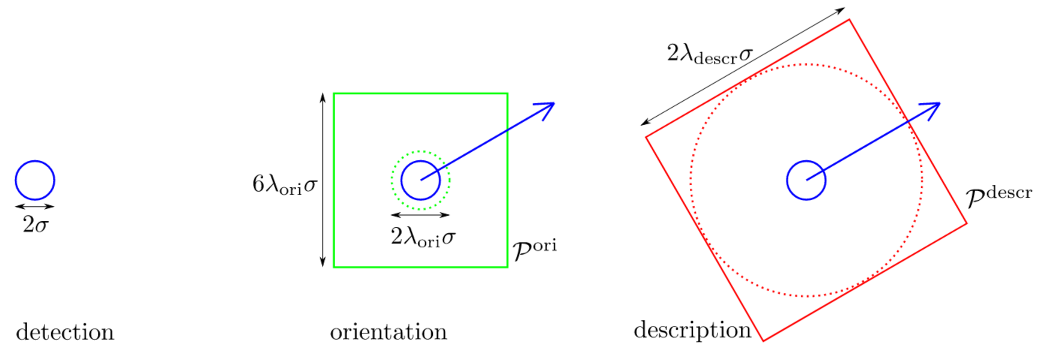
SIFT keypoint detection

Based on an original greyscale image, Scale Invariant Feature Transform (SIFT) approximates a series of images of the same object taken from different ranges, the so-called scale space. This is accomplished by applying increasing levels of Gaussian blur and downscaling. Neighbouring images in the scale space are then subtracted which results in the Difference of Gaussian (DoG). Like the scale space, the DoG space can be seen as a 3D space in which 2 dimensions are the pixel position of the image and the remaining one is the amount of blur applied (the approximated range the image is taken at). Keypoint candidates are now identified by finding local extrema in this 3D DoG space.

These keypoint candidates are restricted to the pixel grid, which is a problem especially at higher levels of downscaling. Therefore, the position of the keypoints is therefore refined to a subpixel level. This is done by creating a local quadratic model in the space around a given keypoint. The extrema of this local model is then identified to define the subpixel position of the keypoint. Unstable keypoints are eliminated by deleting low contrast candidates. This removescandidates with weak local minima which are not very well defined. Keypoints that lie on edges are also purged, since they are normally not distinct.

SIFT descriptor

In the first step, an orientation for the descriptor has to be generated to give it rotation invariance. This is done by finding the predominant direction of gradients in the area of the keypoint. First, the size of the to be evaluated area is chosen based on the scale of the keypoint (see green rectangle in.fig. 4.5). The orientation of the gradient of every pixel in this area is then determined and added to a histogram of directions (normally with 36 bins for 360 degrees). The contribution of every pixel is weighed by the intensity of its gradient and its distance from the keypoint. The highest bin bin_{max} of the histogram and bins that reach above $0.8 * bin_{max}$ are then considered the predominant direction of gradients and descriptors for them are calculated. For this reason, a keypoint may have multiple descriptors if the direction of gradients in its surroundings is ambiguous.



2σ
detection
$6\lambda_{\text{ori}}\sigma$
$2\lambda_{\text{ori}}\sigma$
$\mathcal{P}^{\text{ori}}$
orientation
$2\lambda_{\text{descr}}\sigma$
$\mathcal{P}^{\text{descr}}$
description

Figure 4.5: Visualization of the different patches used during the calculation of the descriptor. σ is the scale of the keypoint.The tuning parameters λ_{ori} and λ_{descr} are parameters are normally set to 1.5 and 6 respectively. [18]

In a second step, the actual descriptor is calculated. For this purpose, a square patch centred on the keypoint and oriented along the keypoints location is taken. Its size is once again dependant on the scale of the keypoint, but larger than in the previous step (see the red rectangle in fig. 4.5). This patch is then subdivided into 16 subpatches. For every subpatch, the predominant orientation of gradients is calculated. This is done in a similar way to the first step, however only 8 bins are used per histogram. The 16 resulting histograms with 8 bins each are then stored as the 128 values long feature vector of the keypoint. This vector characterizes the area around the keypoint and can now easily be compared to other descriptors.

4.2 Simulation

4.2.1 Environment

Simulation modelling is almost always used when developing robotic systems. It is used to speed up the overall software development cycle and to reduce the project's overall cost. It allows to solve real-world problems in a safe and efficient manner and provides the ability to analyse the outcome of an algorithm by simulating various scenarios in a physics simulation environment. The physical interaction between the robot and the environment can be safely analysed before deploying it into the real world.

This situation is equally true for testing vision related algorithms, where the simulation environment provides access to a simulated camera. To make it more similar to that of a real camera, additional noise and distortion can also be easily introduced. Apart from having the ability to test the algorithm in a simulation , it is also necessary to be able to transfer the algorithm to a real world system with ease without modifying the structure of the algorithm. For this reason autopilot software such as PX4 or ardupilot is used to actually control the drone. They provide both Software In the Loop (SITL) and Hardware In the Loop (HITL) modes to test the solution in simulation before actually implementing it on a real drone. SITL is used to simulate the system entirely on a computer and does not require any hardware. Therefore, it is usually the first step in the development process, since the hardware is not available initially. In a later stage, HITL runs the autopilot on the actual flight controller hardware, while physics and control algorithms are still simulated on the computer.

Both of these methods require a simulation environment which simulates a 3D world and the necessary physics. One of the most popular software to provide such an environment is Gazebo. It offers the ability to accurately simulate the physics of any robotic system and its interaction with the environment. It is compatible with Robot Operating System (ROS) and it also has an active community support which is very helpful in debugging solutions. However, when dealing with vision algorithms, the simulation should not only posses realistic physics but also realistic graphics. One of the main problems in using Gazebo is its low visual fidelity. Focused on physics and good performance, the visual details in the simulation are not very realistic. This provides a challenge when testing computer vision algorithms, because the simulation and the real world are not comparable. The identification of features in the simple images provided by the simulation is therefore not representative. One way to get around this issue would be to place visual markers around the area and detect these. This would however be unrealistic and cannot be directly transferred into the real world application.

The unsuitability of widely used robot simulators for computer vision tasks has also been recognised by Microsoft. They developed and subsequently released AirSim [19] as an open source project. AirSim is based on Epic Games Unreal Engine 4, which allows for close to photo-realistic visualization in real time. Mainly developed to help with the generation of machine learning datasets, AirSim allows the simulation of both quadcopters and cars. The simulation environment can be created using the Unreal Engine Editor which allows the creation of very detailed worlds. Moreover, various premade environments with high visual fidelity are available online. These include realistic lighting and world geometry and should therefore be suitable for computer vision tasks. AirSim interfaces with the most commonly used drone autopilots PX4, Ardupilot and, using a wrapper, with ROS.

When comparing Gazebo and Airsim visually, as it is illustrated in fig. 4.6, the more realistic representation of the world is clearly visible. AirSim is therefore used as a simulation environment for this project. Building realistic worlds in Unreal Editor is a very complex and time consuming task. Therefore "Landscape Mountains" [20] provided by Epic Games is used as a base environment which can then be modified. Since the flight model of the drone used is not relevant for this thesis, the default drone delivered by AirSim is used.

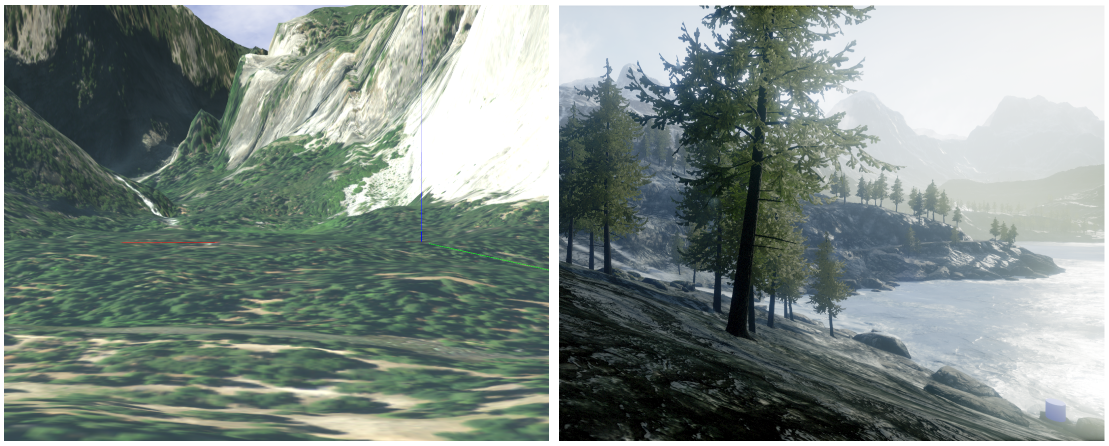

(a) Gazebo environment *(b) Airsim environment*

Figure 4.6: Comparison of Gazebo and AirSim shows the much higher visual fidelity of AirSim

As mentioned above, both PX4 and ardupilot would be suitable autopilots. They both interact with AirSim. However PX4 proved to be more stable, which is why it was chosen for this project. The basic structure of all simulation components

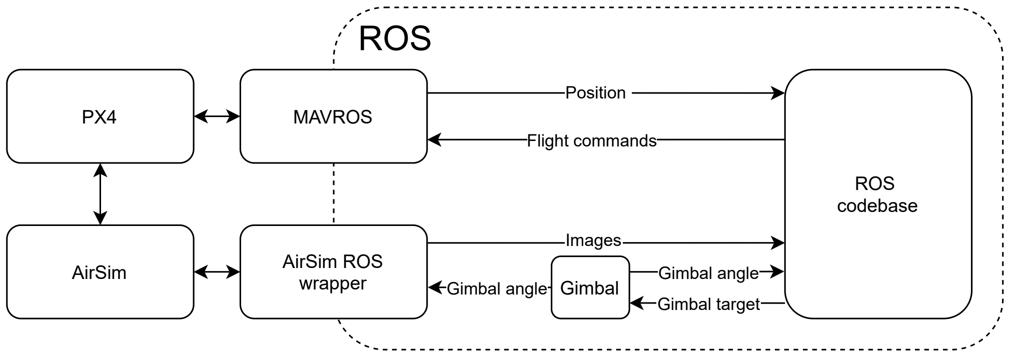


Figure 4.7: This figure shows the data flow between PX4 autopilot, AirSim environment and ROS codebase

can be seen in fig. 4.7. AirSim provides its own ROS wrapper which provides the camera data and gimbal control, but does not provide access to read gimbal orientation or re-position the gimbal. Hence, a gimbal node is introduced in between. It takes in the angle commands from the algorithm and publishes them to the AirSim ROS wrapper to actuate the gimbal. Furthermore, it publishes the current gimbal angles to a ros topic, simulating a sensor. For position data and flight commands the process is easier since MAVROS can be used to interact between the ROS codebase and the autopilot.

4.2.2 Coordinate Systems

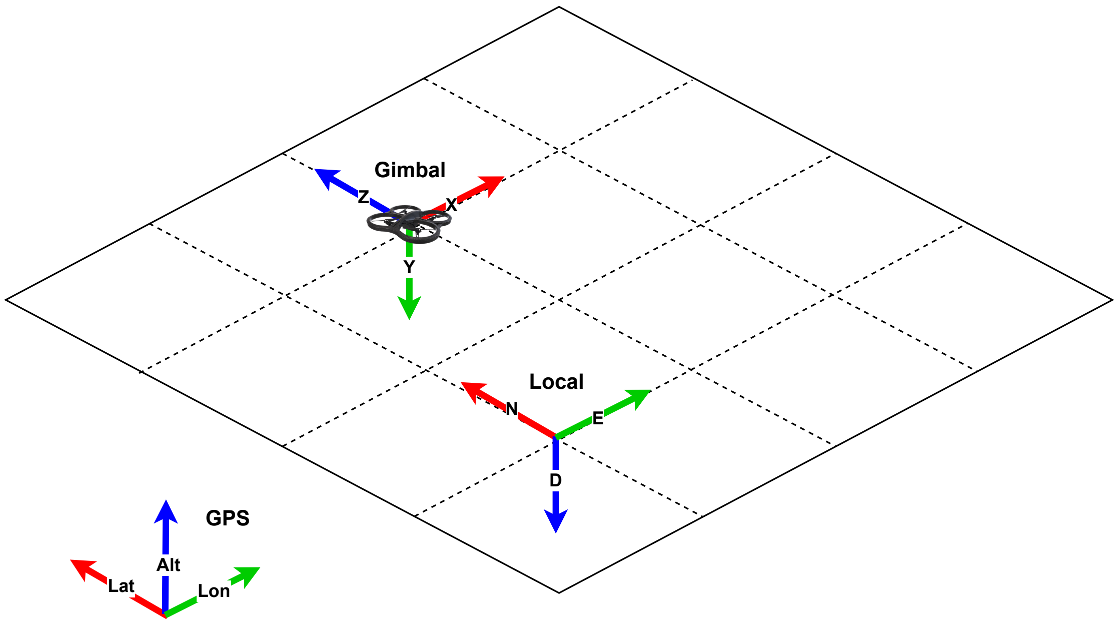


Figure 4.8: This figure displays the coordinate reference frame used for 3D area localization

- GPS coordinate frame: GPS coordinates are handled in the World Geodetic System 1984 (WGS 84) reference frame [21]. In fig. 4.8, the x-axis refers to latitude, the y-axis to longitude and the z-axis to altitude.
- Local coordinate frame: Coordinate frame centred at the start location of the drone.This is a North-East-Down (NED) coordinate system resulting in negative z values for positive altitudes.
- Gimbal coordinate frame: The zero position of the gimbal is looking to the north, parallel to ground. Applying the pinhole camera model [2] results in a coordinate frame that is centred at the drone position with z-axis pointing north, x-axis pointing east and y pointing down.

4.2.3 Communication Architecture

To connect different parts of the system such as the autopilot and the core algorithm, ROS is used. ROS is a flexible open source framework for writing robotic software. It consists of libraries and tools which simplify the task of creating complex algorithms. ROS also supports communication between different language codes, using the concept of nodes, topics, publishers and subscribers as seen in Figure 4.9. Nodes are individual programs which can send and receive data from a topic. Topics act as a middle ground to help exchange information from one node to another. Publishers and Subscribers respectively are the means to send and receive data through a node. ROS has its own data classes that can be used to exchange data with a topic. It also provides the freedom to create custom data classes. Each functionality of a robotic system is made as a ROS node which can send and receive data from topics. For example, in a mobile robot system, each of the sensors (like lidars, encoders, etc.) are individual nodes which publish appropriate data to their respective topics. These topics are then subscribed by the controller node which calculates the required motor input and then publishes the motors commands to another topic. Motor controller is another node which subscribes to these commands and sends it to the appropriate motor device. The advantage of this architecture is that the nodes are completely isolated and are only connected through topics. This allows the use of different languages (C++ and python) for different nodes based on the ease and the computation speed required. Apart from this, ROS has a vast community support to help debug any issues that may arise.

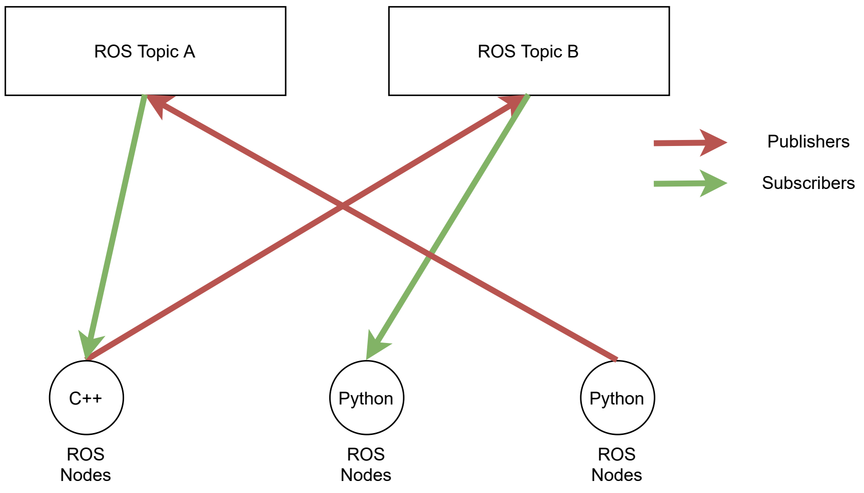


Figure 4.9: This figure describes how nodes communicate with each other using publisher and subscribers through topics.

4.2.4 Mission plan

A 3D area localization mission involves performing state estimation on the detected and matched features from the drone images. While the eventual aim is to perform this algorithm in real time, during the development, datasets where created first and then the algorithm was run on these datasets. The mission plan for capturing such a dataset requires the drone to fly around the target area and collect images of its boundaries. Apart from this, the information regarding the gimbal angles and drone position also has to be recorded for every image taken. The first mission was planned around a lake in the unreal environment as shown in fig. 4.10.

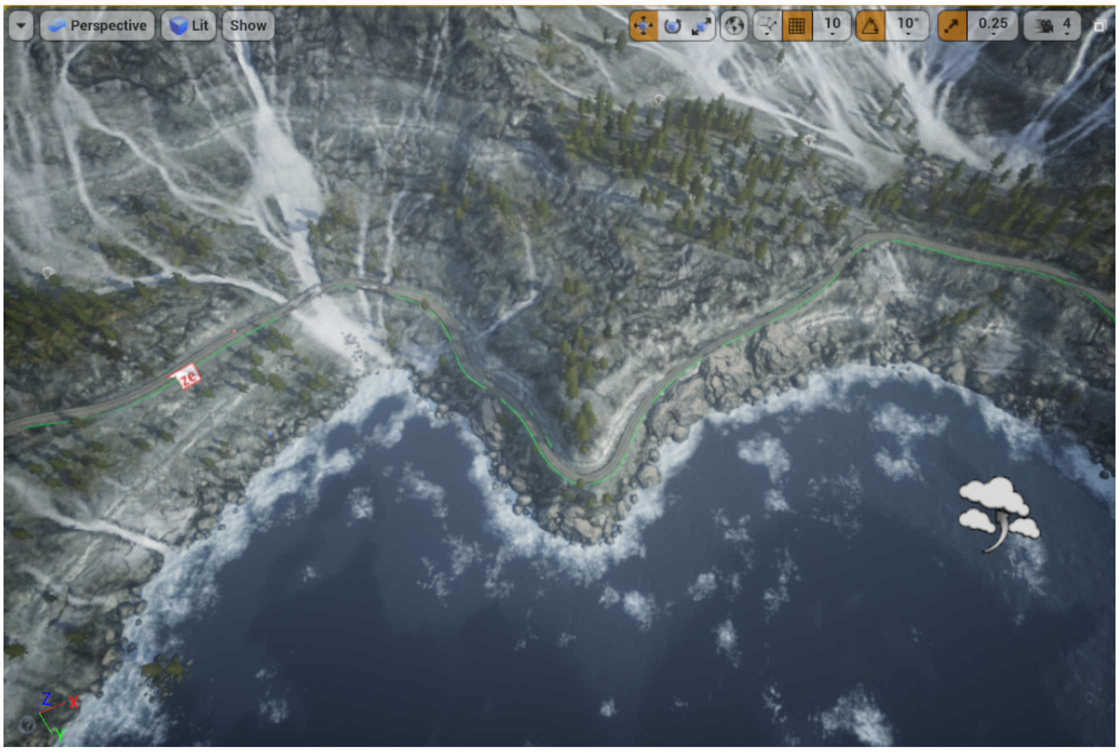


Figure 4.10: Overview of the coastline that serves as a target for the tests. Taken in Unreal Editor.

To achieve this, a few points along this coastline were extracted manually and the polyline they define is used as ground truth as shown in fig. 4.11. This ground truth line is used in calculating the path to be followed by the drones during the mission.

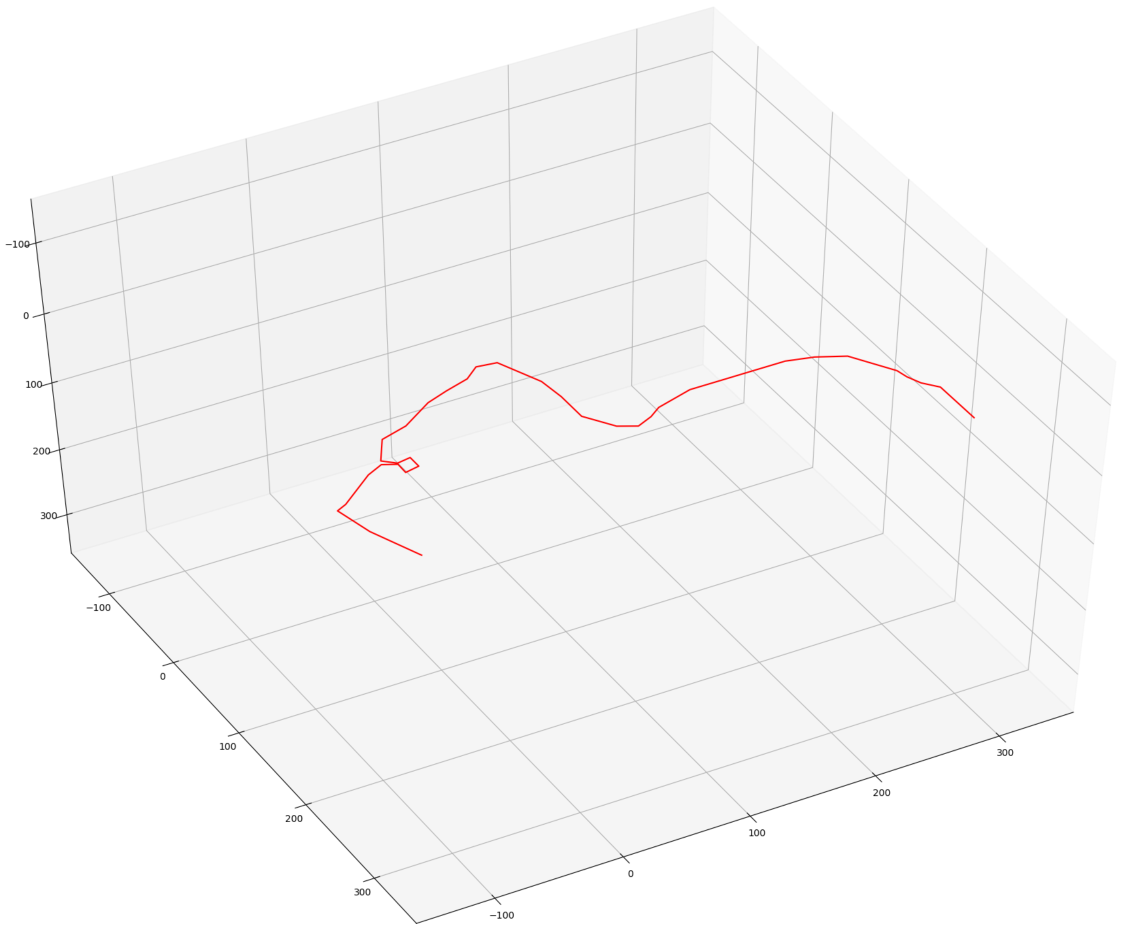


Figure 4.11: Manually created ground truth consisting of 37 extracted points and the polyline they define.

One of the important criteria for a good convergence of the 3D position of a feature is to view the feature from multiple perspectives. To introduce this variation in perspective, the paths must have a certain offset to the ground truth along x, y and z-axis.

The first mission was performed around a lake, which is mostly flat. The drone follows a set of waypoints that were offset along all 3-axis from the ground truth line, at a constrained maximum velocity. The gimbal is pointed towards the closest waypoint on the ground truth line to capture images at regular intervals as shown in fig. 4.12. This simulates either a computer vision algorithm or a human user focusing on the area border during the mission.

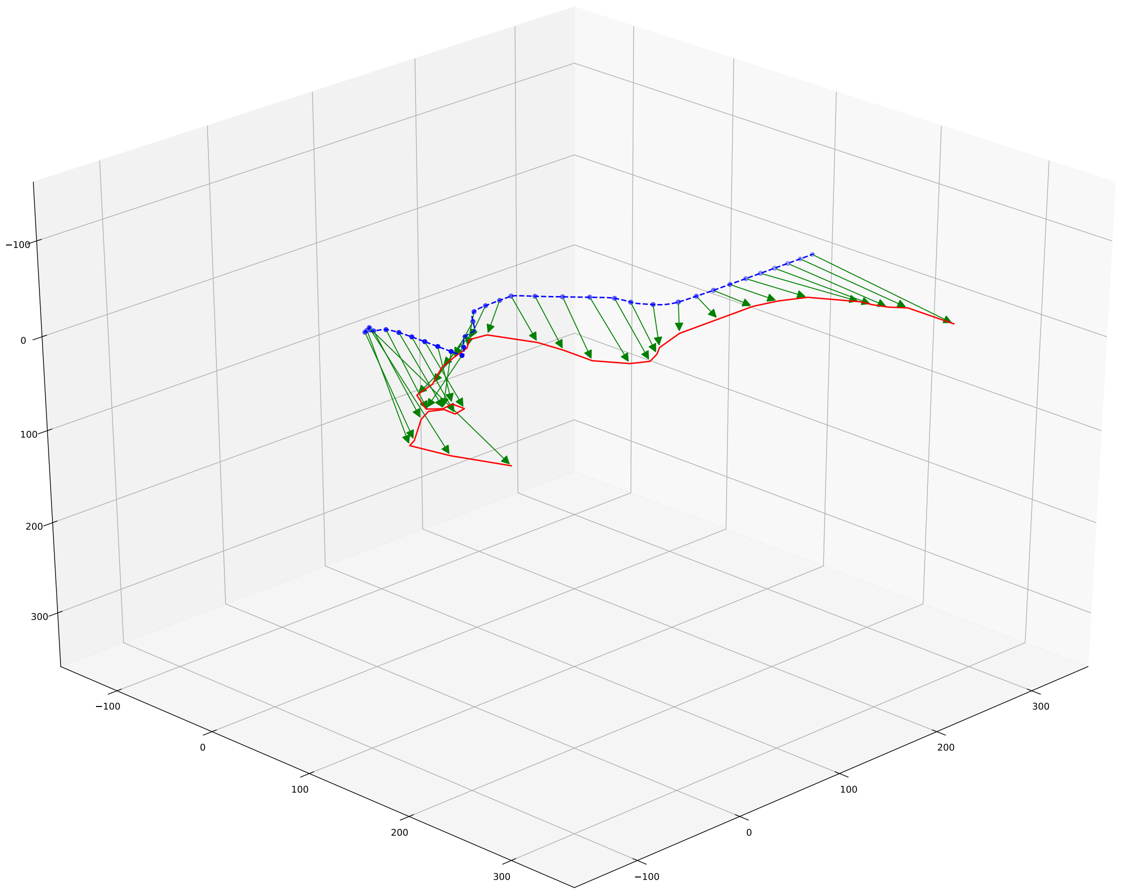


Figure 4.12: The drone follows the path denoted by dashed blue line and focuses the camera onto the boundary of the lake which is denoted by the red line. At every waypoint, the camera is oriented a point on the ground truth line. This orientation is denoted by green arrows.

The second mission is planned around a cliff, which is more of a steeper terrain as shown in fig. 4.13. Similar to the previous mission, a set of waypoints were calculated around the cliff at an offset. The gimbal is made to focus on the cliff's boundary, while the drone follows the waypoints around the area as shown in fig. 4.14. Here, the drone's path is represented by the blue line, the gimbal orientation by green arrows and the ground truth of the cliff's boundary is represented as a red line.

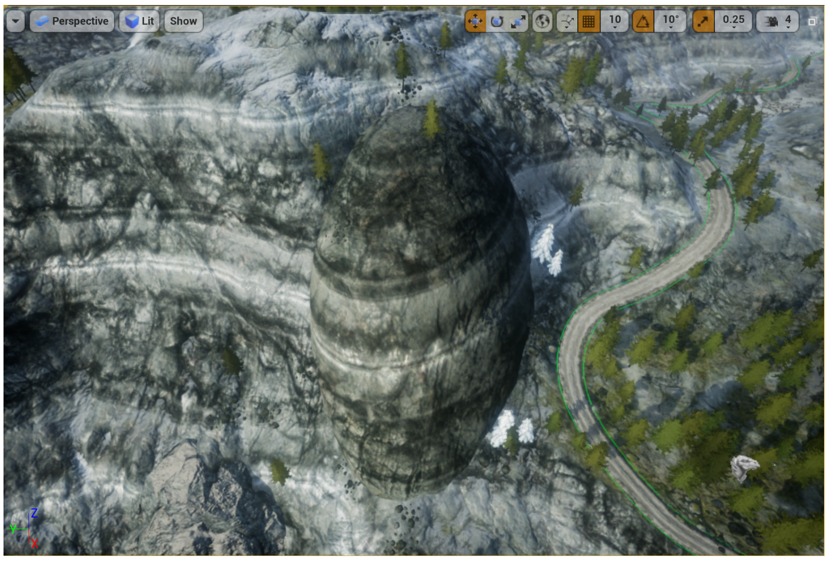


Figure 4.13: Overview of the cliff that serves as a target for the tests.The area that should be localized is of slightly darker colour than the surrounding ground. Taken in Unreal Editor.

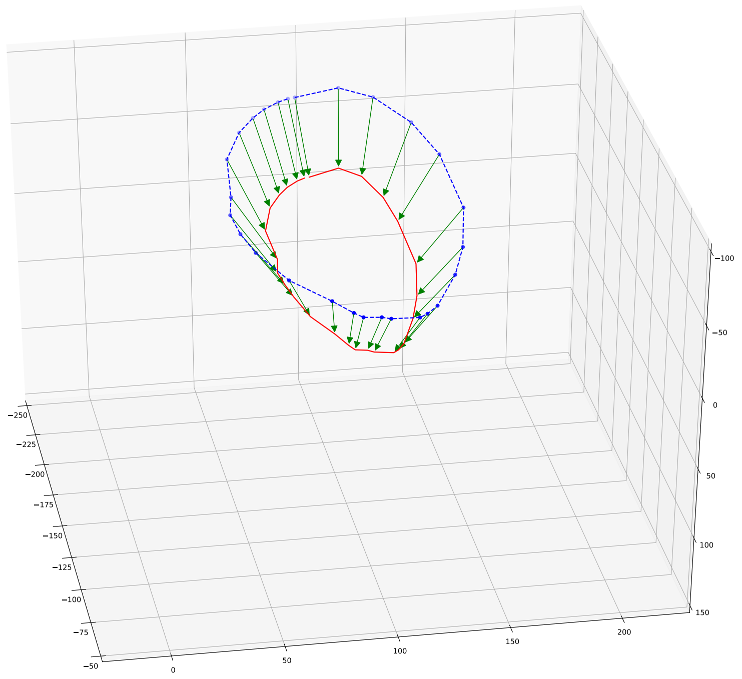


Figure 4.14: The drone follows the path denoted by dashed blue line and focuses the camera onto the boundary of the lake which is denoted by the red line. At every waypoint, the camera is oriented a point on the ground truth line. This orientation is denoted by green arrows.

5 Implementation

Based on the ideas and principles described above, a working algorithm has been developed. It tracks notable landmarks around the target area over a large number of frames. By applying a Kalman filter to every single landmark, it eventually gets accurate estimates of numerous points in space around the target area.

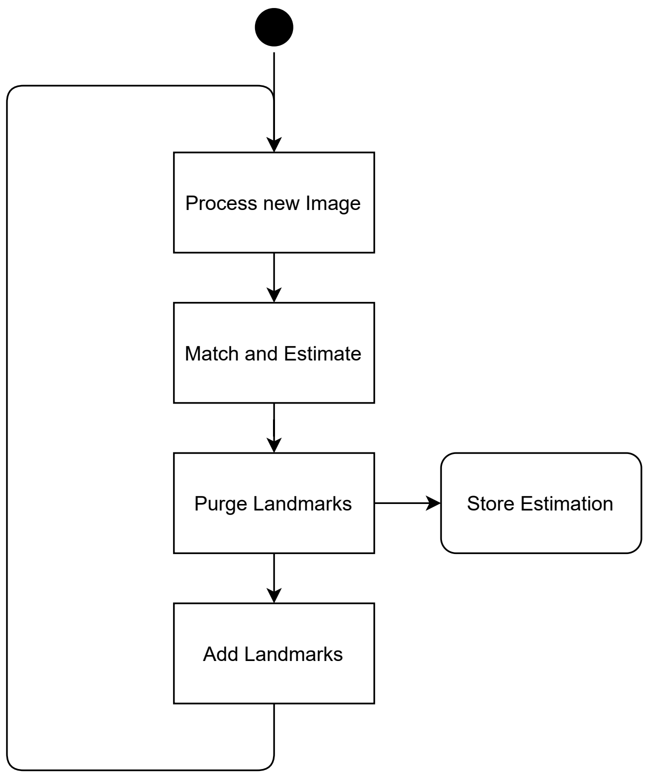


Figure 5.1: This figure gives an overview of workflow of 3D area localization process. The features area detected in Process new Image and are added to LoL in Add landmarks and are matched with the previous landmarks in Match and Estimate. The worst landmarks are purged in Purge landmarks and the 3D estimate of landmarks are saved in Store Estimation.

A simplified flowchart of the algorithm is displayed in fig. 5.1. The algorithm mainly consists of 4 steps:

1. *Process new Image*: Acquires new Image, detects the target area, obtains a mask around it and extracts relevant features.
2. *Match and Estimate*: Matches the current features with the landmarks.When a match is found, updates the landmark and its UKF estimation based on this match.
3. *Purge Landmarks*: Removes bad or old landmarks. If the to be removed landmarks satisfy the convergence criteria, their final estimation is stored.
4. *Add Landmarks*: Adds new landmarks to the LoL

This is an indefinite process and does not have a defined end point. The process can be run during the entire duration of the drone's flight time or can be stopped at any moment. The exact functionality of each block is further explained in detail in section 5.2 - 5.5.

5.1 Data structure

5.1.1 Features

To keep track of landmarks, their associated features and Kalman filters, the following data structure is used. The landmarks are assembled in a Library of Landmarks (LoL) which is displayed in fig. 5.2. Each landmark contains all features associated with it. Every feature consists of keypoints and its corresponding descriptors which will eventually be matched to other features. Features are added to a landmark every time it is rediscovered in an image (see section 5.4). To keep track from which image a feature originated, the image index is also stored. This allows the calculation of metrics such as how often a landmark is seen or when it was last seen.

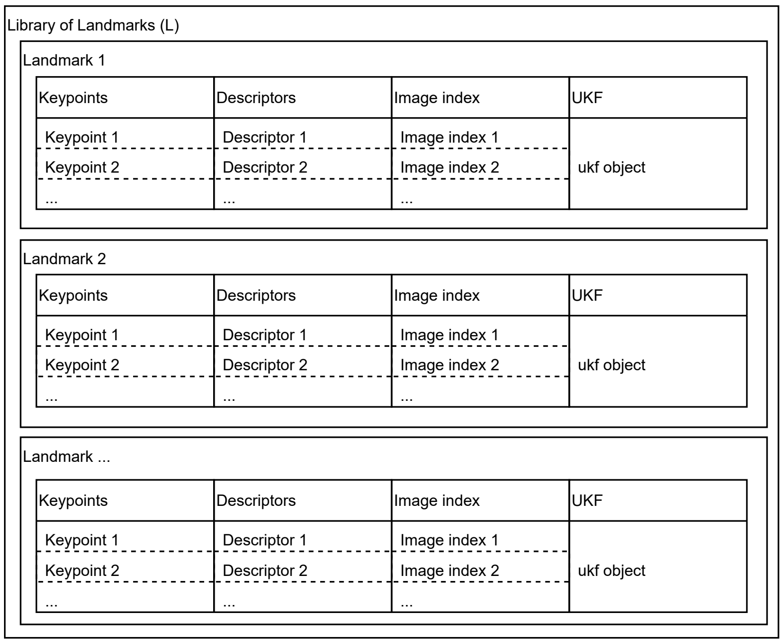


Figure 5.2: Data structure of the LoL. Every Landmark contains information about the associated features, the frame in which a feature has been found and the current state of the UKF estimation.

5.1.2 UKF object

Every landmark also has its own UKF filter which estimates the 3D position of the landmark in the world. To make all information about the UKF filter easily accessible, it is stored in a so called UKF object class. Every landmark contains exactly one such object (see fig. 5.2). While every landmark has its distinct filter, they are all modeled the same. A stationary target is assumed, therefore the state of the target only consists of its position and has dimension $n = 3$:

$$\boldsymbol{x} = \begin{bmatrix} north \\ east \\ down \end{bmatrix} \tag{5.1}$$

When applied to the 3D localization of a static target, the state transition model in (4.6) is linear:

$$\Phi(\boldsymbol{x}[k]) = \boldsymbol{x}[k] \tag{5.2}$$

Because the process noise $\boldsymbol{w}[k]$ of a non-moving target is equal to zero, resulting in an absolutely static state transition.

$$\boldsymbol{x}[k+1] = \boldsymbol{x}[k] \tag{5.3}$$

The measurements taken by the drone consist of the bearing from the drone to the target point expressed as yaw and pitch in the drone frame as shown in fig. 5.3. The measurement space therefore has dimension $m = 2$.

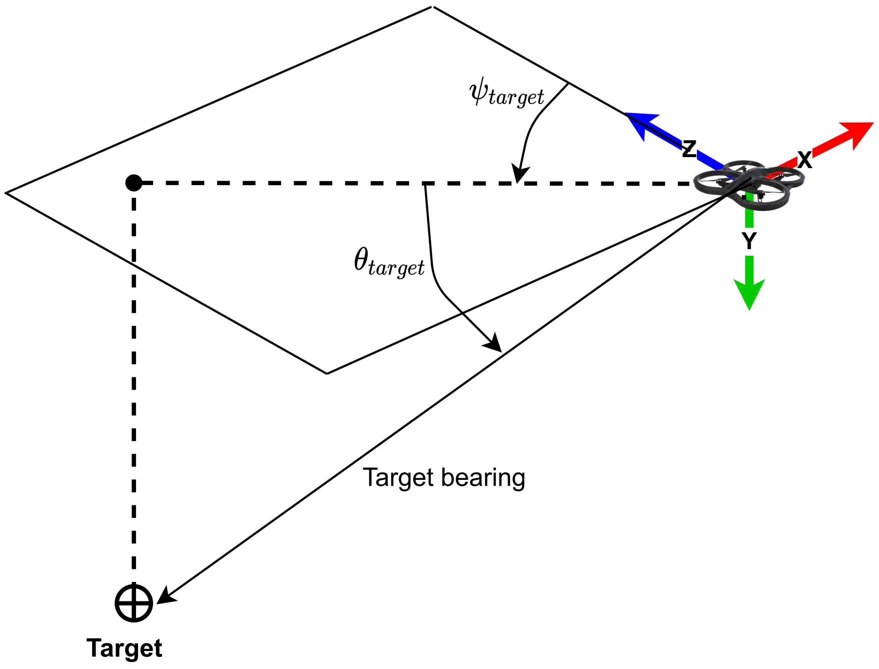


Figure 5.3: The bearing from drone to target can be expressed as yaw angle ψ_{target} and pitch angle θ_{target}. The angles are applied in order yaw → pitch.

$$\boldsymbol{z} = \begin{bmatrix} \psi_{\text{target}} \\ \theta_{\text{target}} \end{bmatrix} \tag{5.4}$$

Since the measurement is expressed as the absolute bearing between the drone and the target, h is also dependent on drone position $\boldsymbol{p}$:

$$\begin{gathered} h(\boldsymbol{x}) = f(\boldsymbol{x} - \boldsymbol{p}) \\ f(\boldsymbol{r}) = \begin{bmatrix} \arctan(\boldsymbol{r}_y/\boldsymbol{r}_x) \\ -\arctan(\boldsymbol{r}_z/\sqrt{\boldsymbol{r}_x^2 + \boldsymbol{r}_y^2}) \end{bmatrix} \end{gathered} \tag{5.5}$$

The measurement model is nonlinear and therefore an unscented transform as described in (4.8) has to be applied. The measurement noise $\boldsymbol{V}$ has to be chosen according to the measurement errors expected. For the simulation tests performed with the simulated errors described in section 6.2.1, the following setting was used:

$$\boldsymbol{V} = \begin{bmatrix} (0.5^\circ)^2 & 0 \\ 0 & (0.5^\circ)^2 \end{bmatrix} \tag{5.6}$$

To select the proper sigma points which define the variance of the state, it is necessary to choose the appropriate tuning parameters as described in (4.1) to (4.5). The tuning parameters chosen after experimentation were:

$$\begin{aligned} \alpha &= 0.7 \\ \beta &= 2 \\ \kappa &= 0 \end{aligned} \tag{5.7}$$

The above-described implementation of the UKF filter is done with the help of the filterpy library [22].

5.2 Process new Image

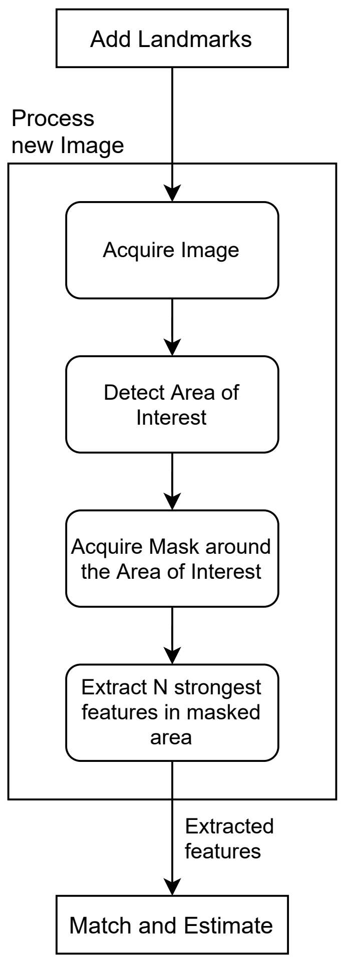


Figure 5.4: The Process new Image block takes in the image, extracts features around the target area and sends it to Match and Estimate block.

The algorithm starts by acquiring a new image from the camera. The image is segmented to get a binary mask of the target area. Segmenting an image depends on the type of area to be segmented and can be achieved by using either simple color segmentation or complex machine learning algorithms. The segmentation is an input for the developed algorithm and how it is created does not affect the result. When the algorithm is applied to the simulation setup described in section 4.2, the segmentation is provided by the AirSim. It segments every object by its unreal engine object class (see fig. 5.5).

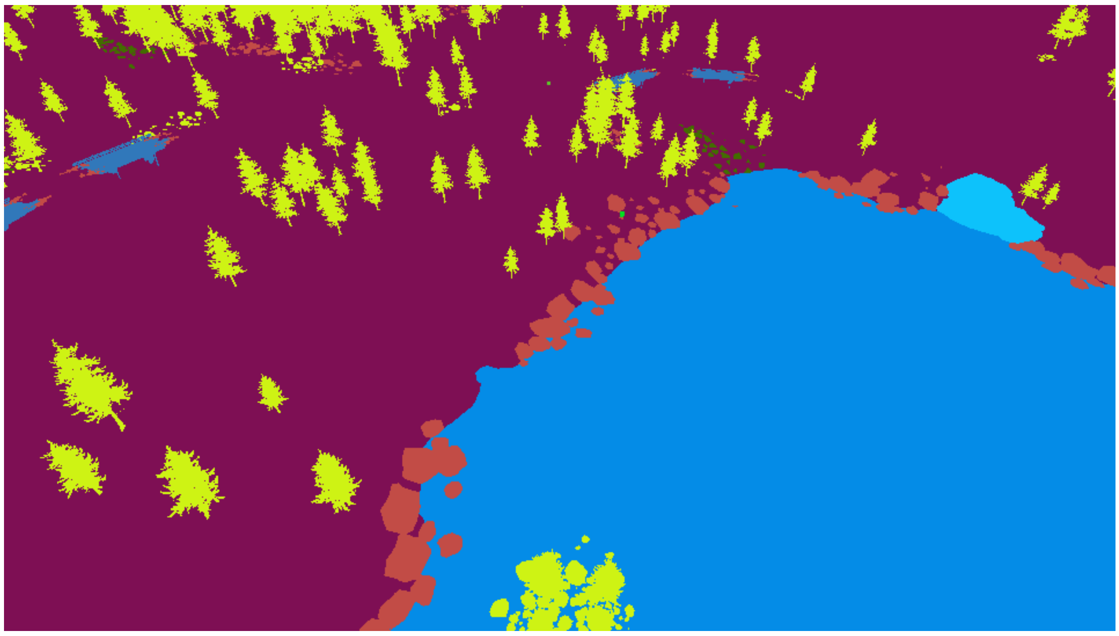


Figure 5.5: The AirSim segmented image output assigns a unique color to object classes in the simulation environment. It is not a result of applying image processing techniques.

The detected area is dilated by performing morphological operations as shown in fig. 5.6. The dilated area is subtracted from the initial segmented area to get a band around the target area as shown in fig. 5.7. This band is the Area of Interest (AoI), in which new features are detected and matched continuously. By selecting a small AoI around the target area, the performance of the feature matching algorithm is improved. This is because, it restricts the detection of features to a limited region as shown in fig. 5.8. Furthermore, the AoI width is a tuning parameter, which determines the amount of features found and how close the 3D position estimates are to the boundary of the target area. Once the mask of the AoI around the target area is calculated, SIFT feature detection algorithm is performed to find the best features. In the first iteration the number of features detected is L_{max}. Here, L_{max} defines the size of landmark library and is user defined. However, in subsequent steps a much larger number of features have to be extracted to match with the existing landmark library. The number of features should be high enough that a majority of the landmarks in the library can be matched and low enough so the matching process can be accomplished in a reasonable time. During the experiments, $10L_{max}$ proved to be a good value. The extracted features are then passed to *Match and Estimate* for updating the position of the landmarks. The only exception is the case of the first iteration.Since the landmark library is

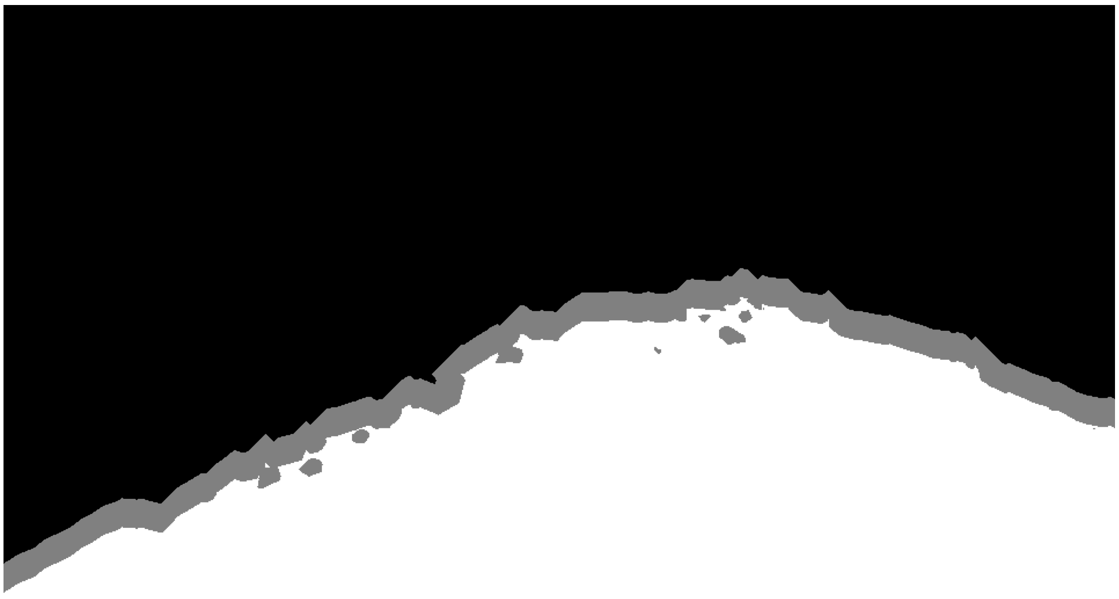
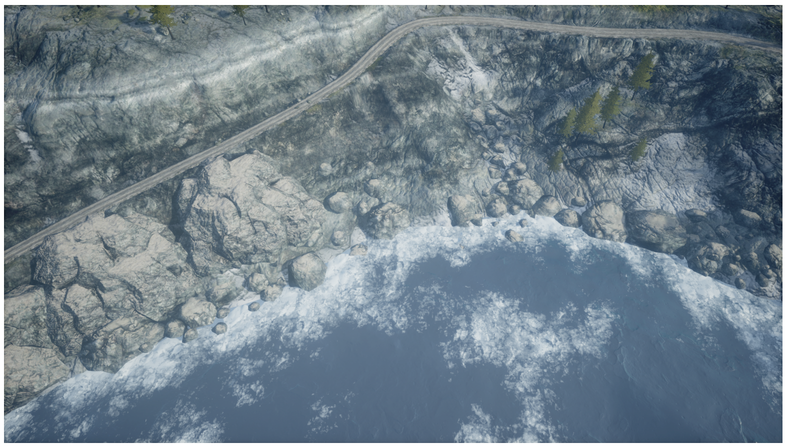
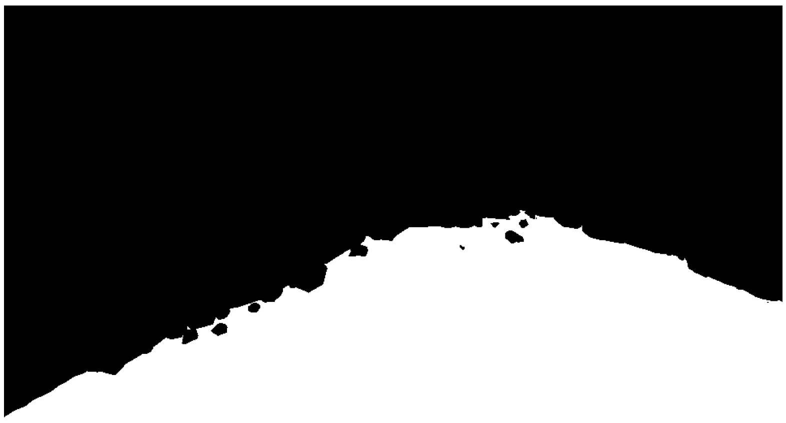


Figure 5.6: In this figure, the white pixels represent the target area, black pixels represent the background and the grey area represents the dilated region.

empty, the extracted features go directly to *Add Landmarks* without performing any action in *Match and Estimate* and *Purge Landmarks*.


(a) RGB Image


(b) Segmented target area

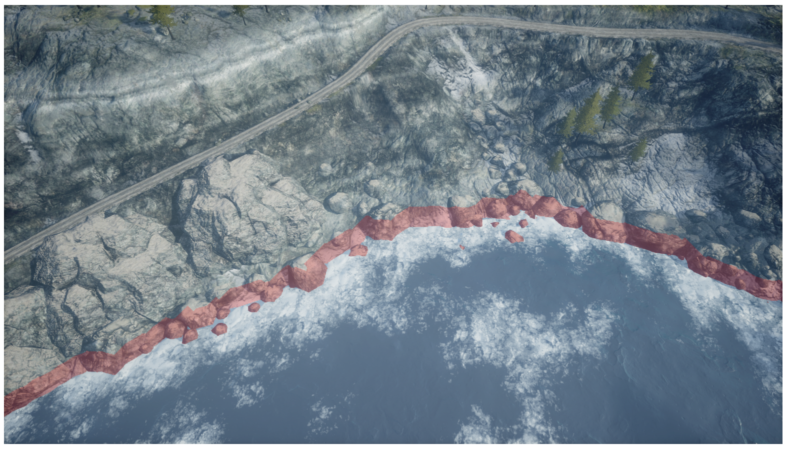
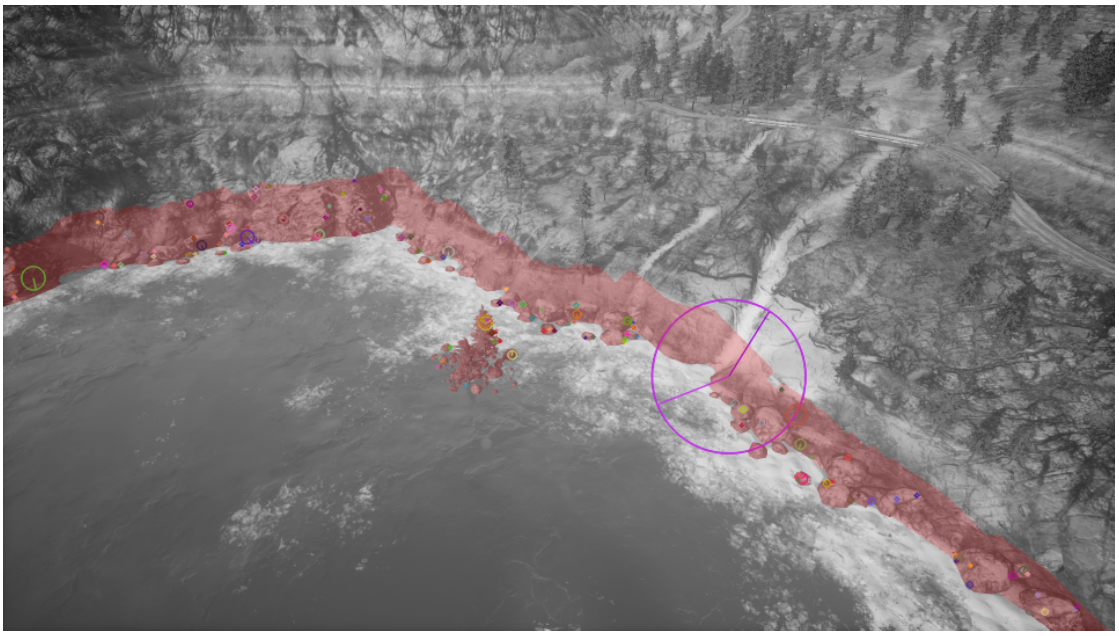
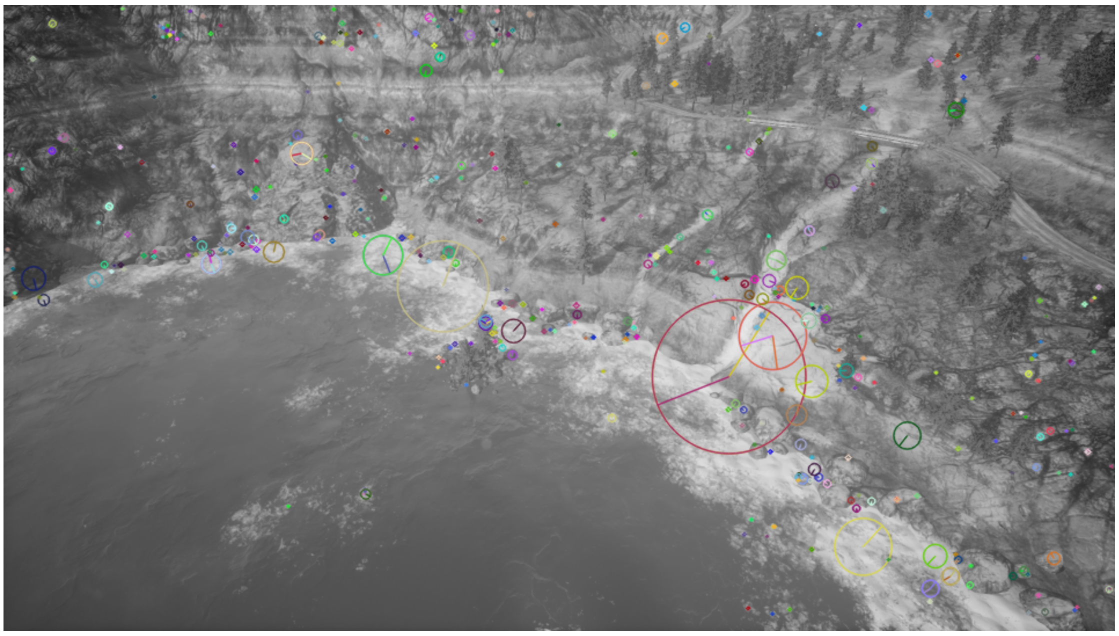

(c) Final output image

Figure 5.7: The RGB image (a) is first segmented to get a mask of the water area (b). This mask is subtracted from a dilated version of itself to get a band around the water (c).



(a) 116 features detected within the masked area



(b) 500 features detected throughout the image

Figure 5.8: In (a) the detection of features are restricted to the AoI and this in turn helps the feature matching process. Whereas in (b) the features are calculated all over the image which might lead to more false positives during feature matching.

5.3 Add Landmarks

The purged Library of Landmarks (LoL) received from *Purge Landmarks* is updated using the new features extracted from *Process new Image*. The LoL maximum size L_{max} is given by the user. Using this and the current size L of the LoL, the number of new landmarks L_{new} to add is calculated as:

$$L_{new} = L_{max} - L \tag{5.8}$$

In the first iteration the LoL is empty. Therefore, the best L_{max} landmarks are selected from all the new features and are added as new landmarks into the library. Once a new landmark is added, the initial 3D position estimate is calculated using the method explained in section 2.1, with height assumed to be zero. This estimate is added into the UKF object of the landmark. The flowchart of the adding landmarks process is displayed in fig. 5.9.

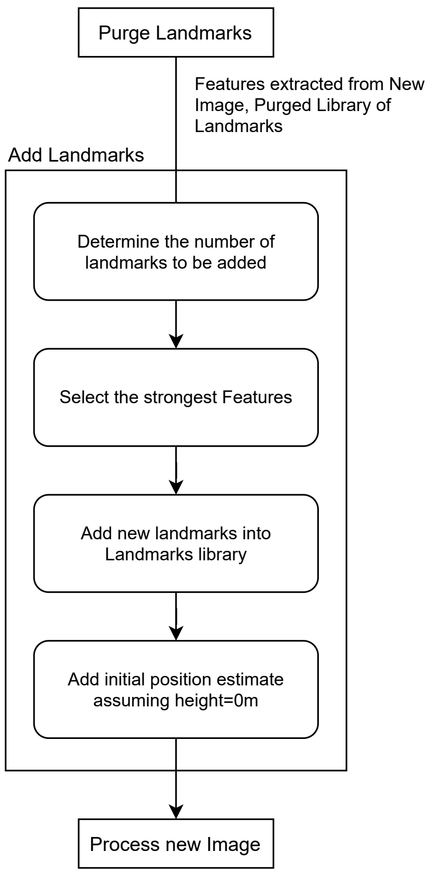
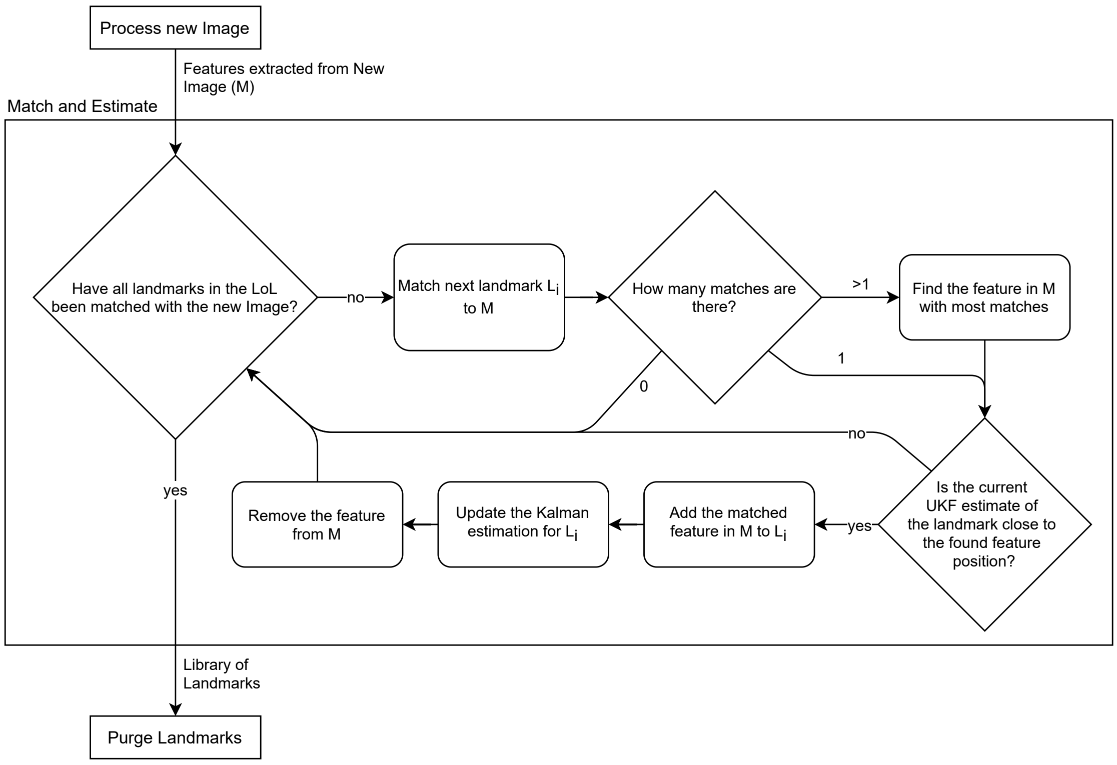

Purge Landmarks
Features extracted from New Image, Purged Library of Landmarks
Add Landmarks
Determine the number of landmarks to be added
Select the strongest Features
Add new landmarks into Landmarks library
Add initial position estimate assuming height=0m
Process new Image

Figure 5.9: Flowchart of the Add Landmarks process.

5.4 Match and Estimate

5.4.1 Match



Process new Image
Features extracted from New Image (M)
Match and Estimate
Have all landmarks in the LoL been matched with the new Image?
no
Match next landmark L$_i$ to M
How many matches are there?
>1
Find the feature in M with most matches
1
0
no
Is the current UKF estimate of the landmark close to the found feature position?
yes
Add the matched feature in M to L$_i$
Update the Kalman estimation for L$_i$
Remove the feature from M
yes
Library of Landmarks
Purge Landmarks

Figure 5.10: Flowchart of the Match and Estimate process.

Every landmark in the LoL is then individually matched with the features extracted in the previous step. Since every landmark holds at most a few hundred features, the extracted features from the Image are limited to a few thousand. Therefore the matching process is not computationally expensive. For this reason and because a fairly low amount of matches are expected, a brute force matcher is used in this process. This provides for every landmark feature descriptor $\boldsymbol{l}$ the two best matching image feature descriptors $\boldsymbol{m_1}$, $\boldsymbol{m_2}$. How well an image feature matches the given landmark feature can be expressed by calculating the distance $s(\boldsymbol{l}, \boldsymbol{m})$:

$$s(\boldsymbol{l}, \boldsymbol{m}) = \|\boldsymbol{l} - \boldsymbol{m}\| \tag{5.9}$$

To decide if $\boldsymbol{l}$ and $\boldsymbol{m_1}$ are a legitimate match, distance ratio r between the best and second best match is calculated:

$$r = \frac{s(\boldsymbol{l}, \boldsymbol{m}_1)}{s(\boldsymbol{l}, \boldsymbol{m}_2)} \tag{5.10}$$

The lower r is, the higher the probability that $\boldsymbol{l}$ and $\boldsymbol{m}_1$ are a correct match. Values close to 1 indicate that the match between $\boldsymbol{l}$ and $\boldsymbol{m}_1$ is not unique, but due to randomness. In the original SIFT paper, Lowe [11] suggests using $r < 0.8$, as it eliminates 90% of false positives while only discarding 5% of positive matches. These are good values for comparing two images with 1000s of matches which allow the use of techniques such as Random Sample Consensus (RANSAC) to get rid of outliers. However, when comparing a landmark to an image, only a few matches are expected. These therefore need to be more reliable. For this reason, the threshold used in this algorithm is $r < 0.5$, which should discard approximately 97% of false matches [11].

Especially for a young landmark with a low number of stored features, there will often be only one match. With older landmarks however, there will often be multiple matches. If matching performance was perfect, all landmark features would match with the same image feature. In reality however, it is possible that the matching is not unanimous. In this case, the image feature with the most matches is declared the true match for that landmark (see fig. 5.11).

This whole process works very well as long as there is an actual match in the new image. If there is none, mismatches can not be eliminated by the majority of good matches and the system may produce false matches. To get rid of the most outrageous outliers, it is checked if the found feature is at least in the general direction of the estimate provided by the UKF filter. This is accomplished by calculating the Mahalanobis distance d [23] between the UKF estimate and the feature to be added.

$$\begin{gathered} \boldsymbol{y} = \boldsymbol{z} - \hat{\boldsymbol{z}} \\ d = \sqrt{\boldsymbol{y}^T \boldsymbol{S}^{-1} \boldsymbol{y}} \end{gathered} \tag{5.11}$$

The Mahalanobis distance measures the distance between a point and a distribution in the measurement space and expresses it in terms of standard deviation. From eq. (5.11), $\boldsymbol{z}$ is the measurement of the feature which is represented as a point, $\hat{\boldsymbol{z}}$ and S represents the distribution of the current UKF estimated measurement.

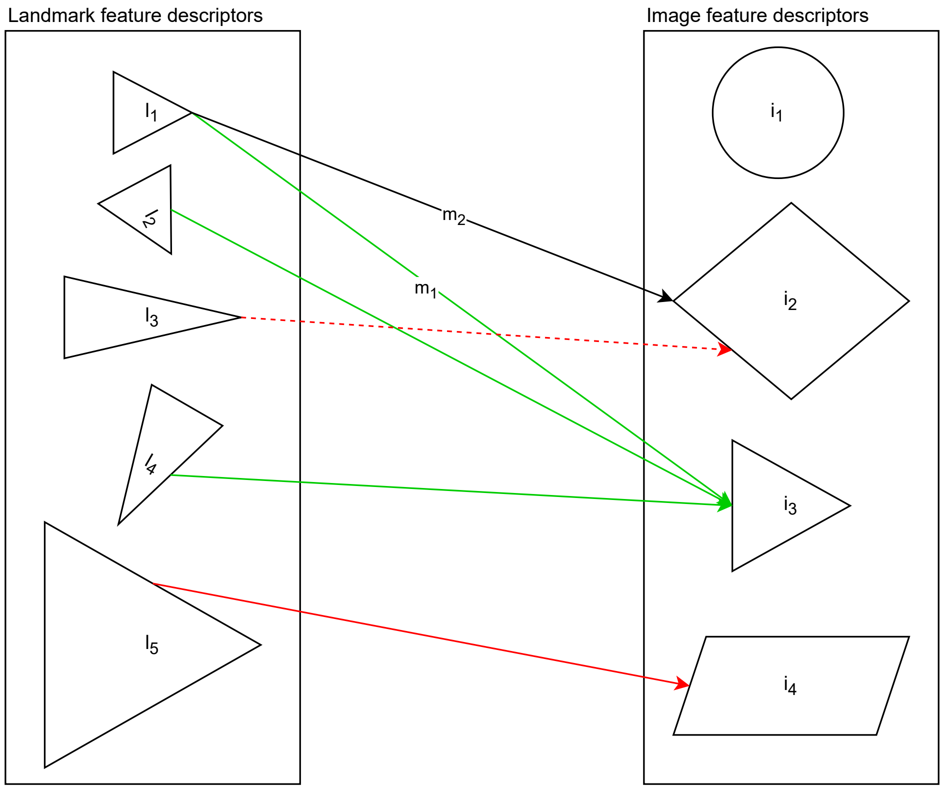


Figure 5.11: For every landmark feature descriptor l_i the best (m_1) and second best (m_2, only displayed for l_1) matching image feature descriptor is found. Depending on the quality ratio between those two, m_1 is declared a real match (straight line) or a mismatch (dotted line). Since there is still the possibility of a false positive match (between l_5 and i_4), the number of real matches every image feature descriptor gets is counted and the most frequently matched one is declared the absolute match between the whole landmark and this image (in this case i_3).

Since d is expressed in standard deviations, a feature is almost guaranteed to be an outlier if $|d| > 3$ and can therefore be discarded. If a match has passed all checks, the image feature will be added to the landmark it matches. This means that the size of a landmark increases with every match found.

5.4.2 Estimate

Once a new feature is added to a landmark, the information about its position must also be added to the landmarks UKF filter. Each feature contains information about its position in an image as pixel coordinates u, v. The UKF filter however requires information in its measurement space, which are bearings from the drone to the target position. First, u and v are reprojected into the focal plane:

$$\begin{bmatrix} u_f \\ v_f \\ 1 \end{bmatrix} = \boldsymbol{A}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5.12}$$

Where $\boldsymbol{A}$ is the intrinsic matrix of the camera.

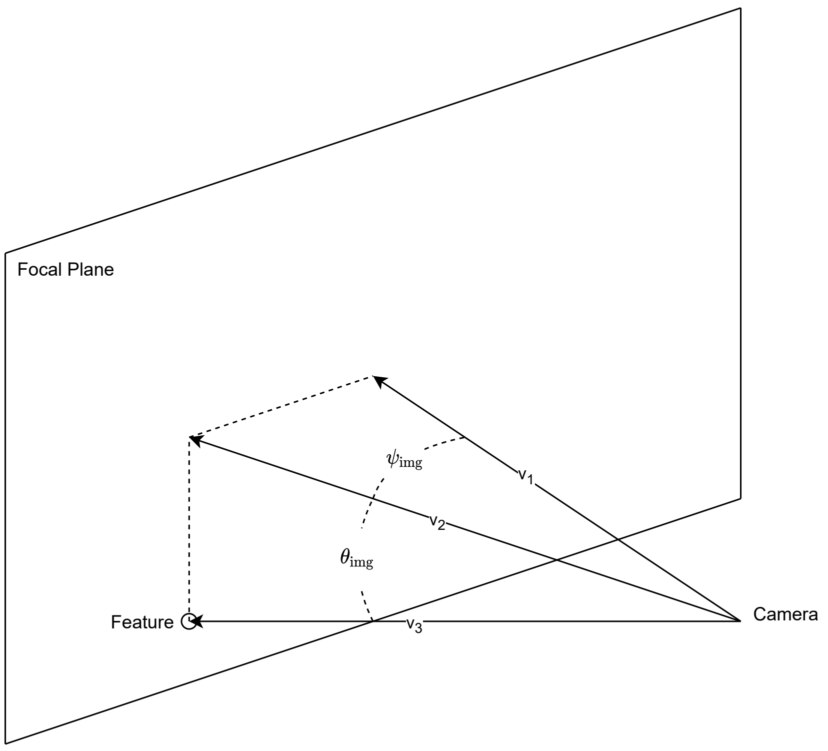


Figure 5.12: Calculation of yaw (ψ_{img}) and pitch (θ_{img}) from the pixel coordinates.

The 3 vectors displayed in fig. 5.12 can now be defined:

$$\boldsymbol{v}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \boldsymbol{v}_2 = \begin{bmatrix} u_f \\ 0 \\ 1 \end{bmatrix} \quad \boldsymbol{v}_3 = \begin{bmatrix} u_f \\ v_f \\ 1 \end{bmatrix} \tag{5.13}$$

ψ_{img} and θ_{img} can then be calculated. They are the yaw and pitch angles from the camera center axis to the target.

$$\begin{aligned}
\psi_{\text{img}} &= \begin{cases} \arccos(\frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{\|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\|}) & u_f \geq c_x \\ -\arccos(\frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{\|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\|}) & u_f < c_x \end{cases} \\
\theta_{\text{img}} &= \begin{cases} \arccos(\frac{\boldsymbol{v}_2 \cdot \boldsymbol{v}_3}{\|\boldsymbol{v}_2\| \cdot \|\boldsymbol{v}_3\|}) & v_f \leq c_y \\ -\arccos(\frac{\boldsymbol{v}_2 \cdot \boldsymbol{v}_3}{\|\boldsymbol{v}_2\| \cdot \|\boldsymbol{v}_3\|}) & v_f > c_y \end{cases}
\end{aligned} \tag{5.14}$$

Where c_x and c_y are the principal point offsets stored in the intrinsic matrix as can be seen in (2.8). This assumes the order of rotations to be yaw first and pitch second.

Applying these rotations after each other results in the rotation matrix from gimbal frame to the target $\boldsymbol{R}^{G \to T}$:

$$\boldsymbol{R}^{G \to T} = \begin{bmatrix} \cos(\psi_{\text{img}}) & 0 & \sin(\psi_{\text{img}}) \\ 0 & 1 & 0 \\ -\sin(\psi_{\text{img}}) & 0 & \cos(\psi_{\text{img}}) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{\text{img}}) & \sin(\theta_{\text{img}}) \\ 0 & \sin(\theta_{\text{img}}) & \cos(\theta_{\text{img}}) \end{bmatrix} \tag{5.15}$$

The total rotation matrix from local frame to the target $\boldsymbol{R}^{L \to T}$ can then be calculated:

$$\boldsymbol{R}^{L \to T} = \boldsymbol{R}^{L \to G} \boldsymbol{R}^{G \to T} \tag{5.16}$$

Where $\boldsymbol{R}^{L \to G}$ is the rotation matrix from local frame to gimbal frame. Its calculation can be seen in (2.2) to (2.4). Since (2.4) is dependant on the kinematic chain of the gimbal, it has to be modified depending on the gimbal used:

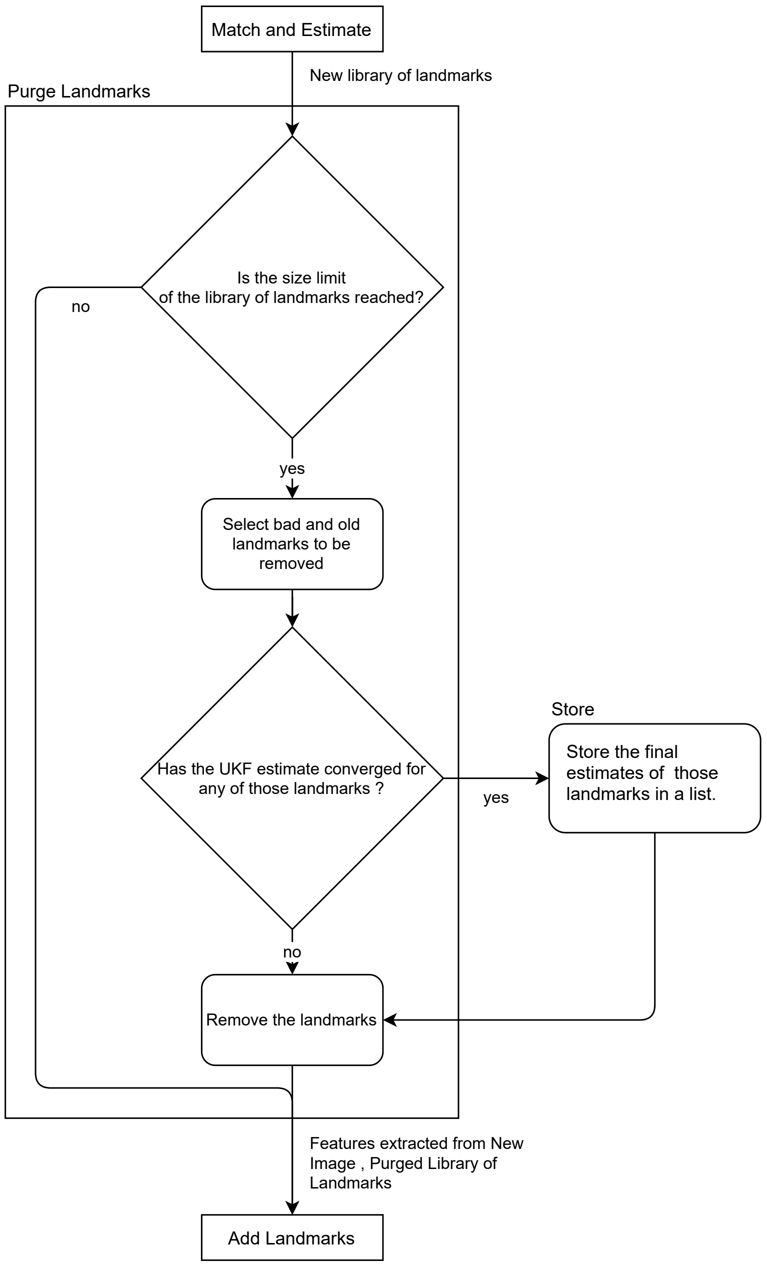
$$\boldsymbol{R}^{G_0 \to G} = \begin{cases} \boldsymbol{R}_{\text{yaw}} \boldsymbol{R}_{\text{roll}} \boldsymbol{R}_{\text{pitch}} & \text{Zenmuse h20T} \\ \boldsymbol{R}_{\text{yaw}} \boldsymbol{R}_{\text{pitch}} \boldsymbol{R}_{\text{roll}} & \text{AirSim} \end{cases} \tag{5.17}$$

The measurement $\boldsymbol{z}$ can now be calculated:

$$\boldsymbol{z} = \begin{bmatrix} \psi_{\text{target}} \\ \theta_{\text{target}} \end{bmatrix} = \begin{bmatrix} \arctan\left(\frac{\boldsymbol{R}^{L \to T}_{1,2}}{\boldsymbol{R}^{L \to T}_{0,2}}\right) \\ \arctan\left(\frac{\boldsymbol{R}^{L \to T}_{2,2}}{(\boldsymbol{R}^{L \to T}_{1,2})^2 + (\boldsymbol{R}^{L \to T}_{0,2})^2}\right) \end{bmatrix} \tag{5.18}$$

$\boldsymbol{z}$ is then used to update the UKF filters estimation as described in (4.10).

5.5 Purge Landmarks



Match and Estimate
New library of landmarks
Purge Landmarks
Is the size limit of the library of landmarks reached?
no
yes
Select bad and old landmarks to be removed
Store
Store the final estimates of those landmarks in a list.
Has the UKF estimate converged for any of those landmarks ?
yes
no
Remove the landmarks
Features extracted from New Image , Purged Library of Landmarks
Add Landmarks

Figure 5.13: Flowchart of the purging Landmarks process.

This is the final step of the algorithm. Its purpose is to avoid slowing down the algorithm with an ever increasing number of landmarks. As the drone moves through an area, landmarks are detected, matched and their positions are estimated. Most of the landmarks tracked are lost as the drone moves, since the features go out of frame. These features have to be discarded in order to make space for new features. The purging process is done only when the LoL is above its purging threshold L_{purge}, which is equal to:

$$L_{purge} = L_{max} - T \tag{5.19}$$

Where T is the turnover rate. This parameter is a user-defined tuning parameter. It controls the amount of landmarks to be deleted in every iteration and should be selected based on the speed of the drone and the image acquisition frequency. Choosing the turnover rate too high, results in the removal of landmarks which are yet to converge. Choosing it too low would fail to remove all bad landmarks. When the number of landmarks exceeds the purging threshold, the excess $L - (L_{purge})$ worst landmarks are selected. The worst landmarks are selected based on their quality Q, which is determined by the ratio of features it contains to the age of the landmark.

$$Q = \text{size}(L_i)/\,\text{age}(L_i) \tag{5.20}$$

where L_i is the evaluated landmark. By this criteria, the landmarks that do not have frequent matches, also referred to as bad landmarks, will be queued up for purging. Furthermore, landmarks that have converged but are now out of frame, are eventually included for purging. Therefore,the purging process does not just remove bad landmarks, but also the good ones that have not been seen for a while. Before they are purged, the landmarks whose position estimates have converged are stored in a global list as shown in fig. 5.13. This is done to ensure that the information contained in these converged landmarks is stored before they are deleted. Converged landmarks are selected and saved based on their low covariance value. This is evaluated using the covariance matrix given after the UKF state estimation. First, the eigenvalues $\boldsymbol{e}$ of the covariance matrix are calculated. The real parts of the eigenvalues describe the variance of the distribution along the three principle axes. The square roots of these variances are the standard deviations of the distribution along these axes. The value of the maximum standard deviation is used as the parameter to evaluate the convergence of the 3D position estimate of the landmark.

$$\eta = \max(\sqrt{\text{Re}(\boldsymbol{e})}) \tag{5.21}$$

η is the convergence criteria that is checked for every landmark that is to be deleted. If it falls below a chosen threshold, the landmark will be added to the final results. After checking for convergence, all selected features are deleted.

5.6 Challenges encountered

5.6.1 Synchronization

For the best results, position and angle data associated with a picture must be as accurate as possible. This can be a challenge since the data for these values follow different paths. As illustrated in the simulation structure displayed in fig. 4.7. Pictures are provided by AirSim directly, position data by PX4 and gimbal angles by a self written ROS script. This information is published at different intervals. Furthermore, the picture data is heavily dependent on the performance of the simulation and might not always be published with the same frequency. In initial experiments, the Image topic was used as a trigger to assemble the information from the other topics (see fig. 5.14). The Image topic was chosen because it is published at the lowest frequency and the resulting discrepancies with the other topics should therefore be the lowest.

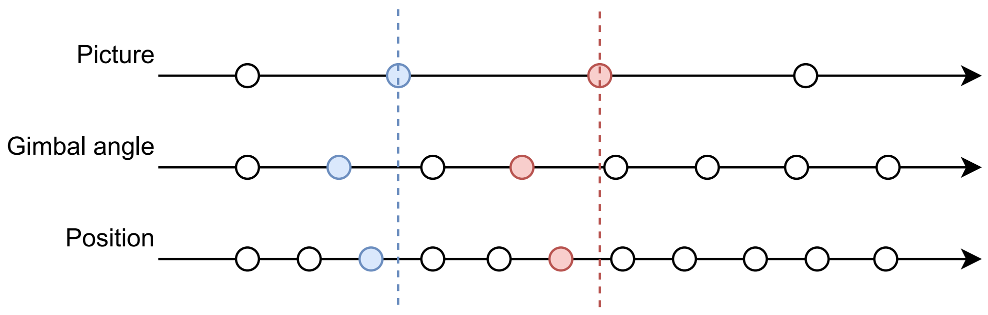
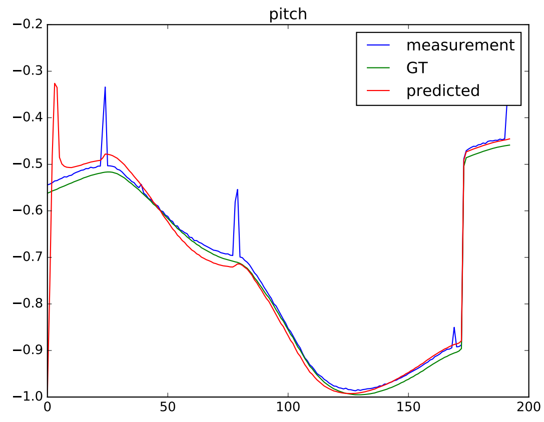
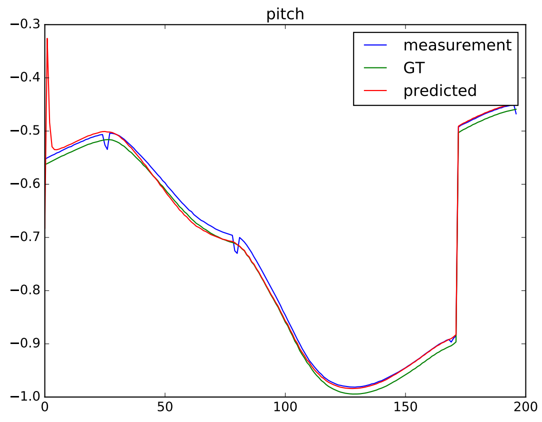


Figure 5.14: Information is collected whenever a picture is published. This can lead to sub optimal data association.

There are 2 main problems with this approach: First, there are situations where a position or gimbal angle update is published shortly after the picture (see red data point in fig. 5.14). This new information would fit the actual state of the image way better than the old information that is used. Secondly, the topics are evaluated based on when they are received, not when they are created. A slowdown in one of the pipelines could therefore lead to mismatched data. This is especially true for the gimbal angle topic, which is calculated based on information depending on the picture topic. As can be seen in fig. 5.15a, this can lead to bad measurements. Because of the time differences between measurement and acquisition of the image, the gimbal angles are trailing the image information by approximately one frame. While this is not noticeable when gimbal angles change very slowly, it leads to big spikes when the gimbal is slewed very fast. These big spikes do have an effect on the



(a) *before optimizing synchronization*



(b) *after optimizing synchronization*

Figure 5.15: Example of target pitch angle from the drone to a Landmark. The ground truth (GT) is calculated based on the drone position and the known Landmark position. The measurement combines information from drone position, gimbal angles and image. The spikes in the measurement are a result of mismatched image and gimbal data when the gimbal is moved sharply.

UKF prediction and slow down the convergence. ROS provides some tools to solve these problems. Most topics include a timestamp of their creation. Based on these time stamps ROS ApproximateTime filter can associate data with the smallest ammount of time difference (see fig. 5.16). Furthermore the timestamps can be manually set. This allows topics that are calculated based on information received from other topics to share the original timestamp. Applying these techniques

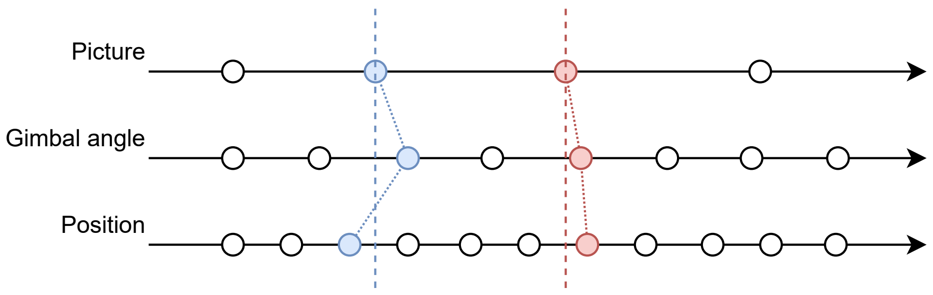


Figure 5.16: ROS ApproximateTime filter is used to synchronize the data depending on their creation time stamp. While this will introduce some latency since the filter has to wait for some topics, the data will be synchronized much better.

resulted in better results and faster convergence of the UKF prediction. This can be seen when comparing fig. 5.15a and 5.15b. The differences between measurement and GT are generally lower and the UKF prediction converges more steadily.

5.6.2 Numerical problems

One of the problems faced during UKF are numerical issues during the calculation of sigma points. This calculation involves computing the square root of a matrix (see (4.1)). This is done using the Cholskey decomposition, which decomposes a matrix $\boldsymbol{B}$ into a combination of a matrix $\boldsymbol{L}$ and its transpose $\boldsymbol{L}^T$.

$$\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^T \tag{5.22}$$

Applying the Cholskey decomposition presupposes a positive semidefinite matrix. A symmetric matrix $\boldsymbol{B}$ with real entries is said to be positive semi-definite if $\boldsymbol{yBy}^T$ is positive or zero for all non-zero real values of $\boldsymbol{y}$. While covariance matrices generally should be positive semidefinite, this was not always the case with the matrices numerically calculated by the Kalman filter. This is a common problem faced by many programmers and discussed in several forums, but with no general solution. One proposed solution is to ensure matrix symmetry by applying the following fix:

$$\boldsymbol{B} = (\boldsymbol{B} + \boldsymbol{B}^T)/2 \tag{5.23}$$

This reduced the occurrence of nonpositive semidefinite matrices but did not completely solve the problem. Another possible solution is to calculate the closest positive semidefinite matrix as described by Higham [24]. Even though this worked in most cases, it also slowed down the process significantly and could not eliminate all occurrences. During the tuning of the sigma point parameters, the authors noticed that the value of α had a significant influence on the frequency of these errors. By choosing a suitable value of 0.7, the observed errors could be completely eliminated.

6 Analysis

Simulation

The tests are conducted in the AirSim simulation environment with water as the target area to be localized. The drone used in this simulation is a standard drone provided by AirSim, configured with a full-hd 3-axis gimbal camera. The images and data required are collected through the simulation and stored as a dataset. The algorithm is then run at a later time on the collected dataset. Since it is computationally expensive and not yet optimized, it is run at less than real time speed.

To run the tests, a part of the coastline in the unreal environment is chosen as the target (see fig. 4.10). 37 points along this coastline were extracted manually and the polyline they define serves as ground truth (see fig. 4.11).

To evaluate the quality of the resulting estimated landmark positions, the minimal distance from each landmark to the ground truth polyline is calculated. Low distances are desirable but because of the nature of the algorithm, a perfect 0 m is impossible. Since the landmarks are chosen from a band along the ground truth, they will have a certain distance even if the Kalman estimation is perfect.

The landmarks are further classified into three groups:

- Distance < 10 m: useful landmarks which can be used to estimate the contour of the target area.
- 10 m $<$ distance < 20 m: landmarks that might have converged to their real position but are too far away to be useful or landmarks that did not fully converge.
- distance > 20 m: landmarks that converge to a wrong position (outliers) and require analysis.

6.1 Initial results

The results obtained by an initial test of the 3D target localization algorithm are displayed in fig. 6.1. Although this figure provides a good representation of the contour of the area, it also presents a lot of outliers (displayed in red). In order to reduce the number of outliers, the reason for them being present had to be identified.

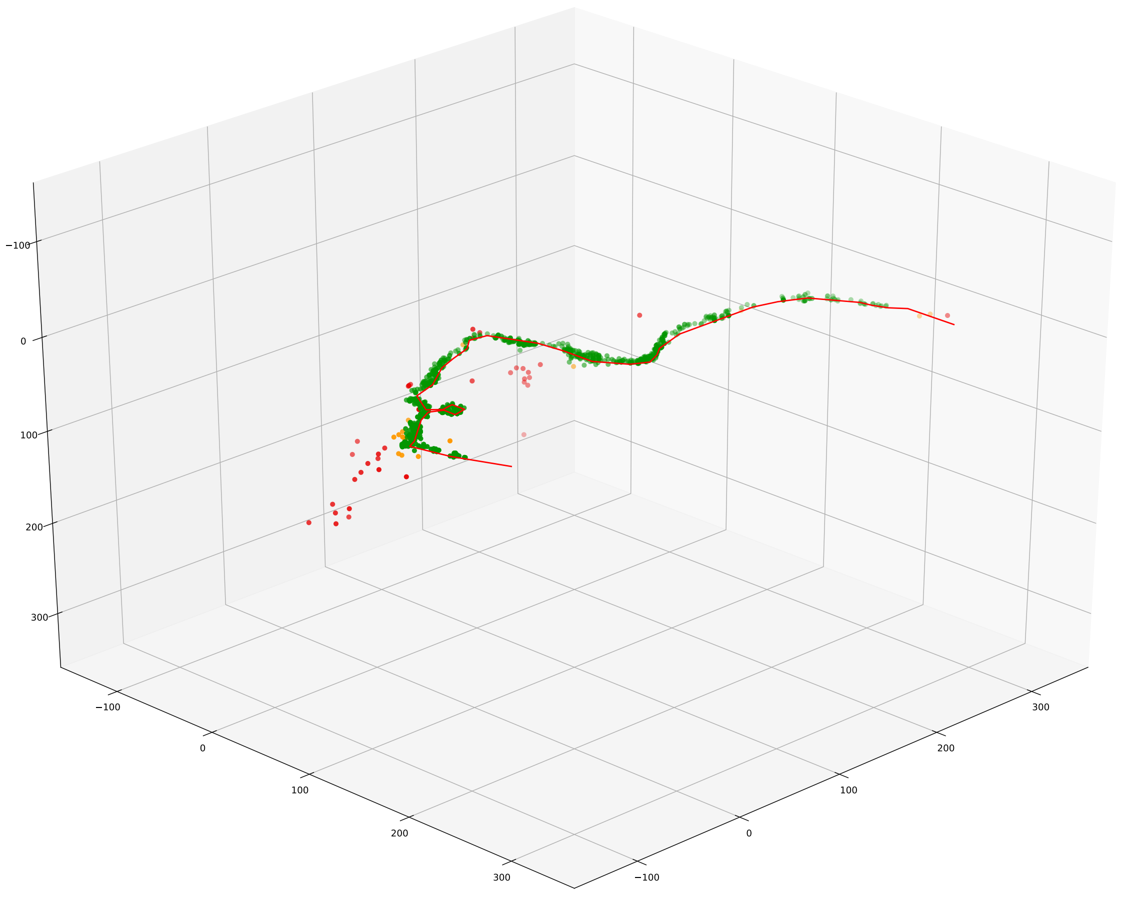
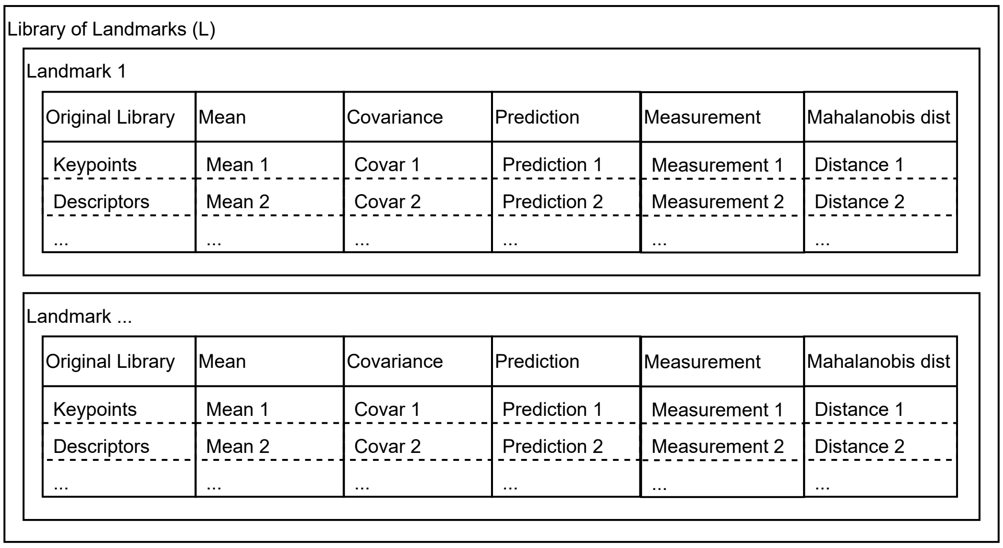


Figure 6.1: Displays the 3D estimates of the landmarks found around the lake. The red line denotes the ground truth, green points represent the inliers, yellow points are one which are considered not useful for 3D area localization and red point are the outliers.

6.2 Outlier analysis

In order to analyse the outliers, the information that was used to localize every landmark, needs to be stored at every frame. This is done by modifying the LoL to accommodate information about the measurement (target yaw and pitch), prediction (target yaw and pitch), mean of estimation (x,y,z), covariance of estimation and the Mahalanobis distance as shown in fig. 6.2. This allows to track the history of every landmark and analyse every single update.



Library of Landmarks (L)

Landmark 1

Original Library	Mean	Covariance	Prediction	Measurement	Mahalanobis dist
Keypoints	Mean 1	Covar 1	Prediction 1	Measurement 1	Distance 1
Descriptors	Mean 2	Covar 2	Prediction 2	Measurement 2	Distance 2
...	...	...	...	...	...

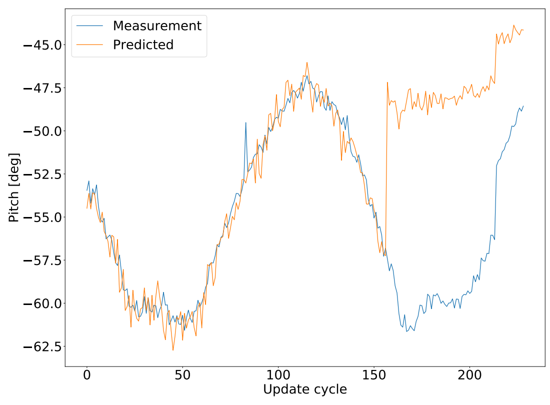
Landmark ...

Original Library	Mean	Covariance	Prediction	Measurement	Mahalanobis dist
Keypoints	Mean 1	Covar 1	Prediction 1	Measurement 1	Distance 1
Descriptors	Mean 2	Covar 2	Prediction 2	Measurement 2	Distance 2
...	...	...	...	...	...

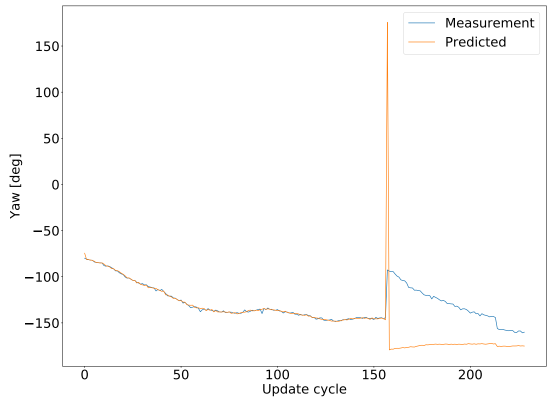
Figure 6.2: Additional elements are added to the library of landmarks, they are required to allow analysis of all updates of every landmark.

The outliers are caused by landmarks that are not consistently matched. One such example can be seen in fig. 6.3. This outlier data is compared with the data plots of one of the inlier points in fig. 6.4 to find the reason for its inaccuracy.

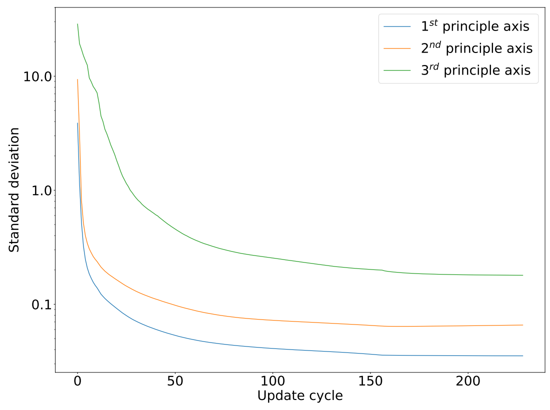
From fig. 6.3a and fig. 6.3b it can be observed that, at around update cycle 150, there is jump in the measured pitch and yaw angles. This indicates that there was a false feature matching at that instant. The feature matcher then follows the new feature and the estimates converge to a new position as seen in fig. 6.3e. In fig. 6.3d it can be seen that the miss match leads to a very big mahalanobis distance since it is very unlikely that the new point is part of the current UKF-estimate distribution. These kind of outliers can therefore be removed by applying a threshold criteria on mahalanobis distance as described in section 5.4.1 (this part of the algorithm was only added as a reaction to the initial tests described here).



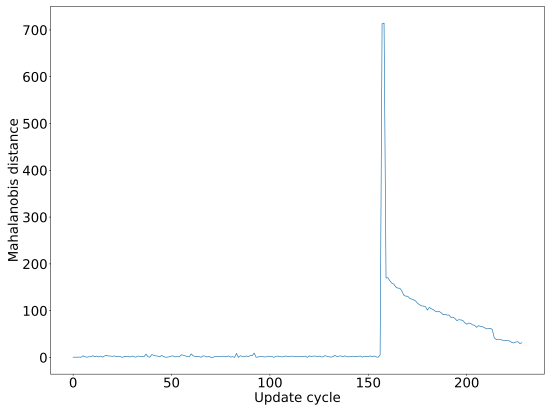
(a) Target pitch angle



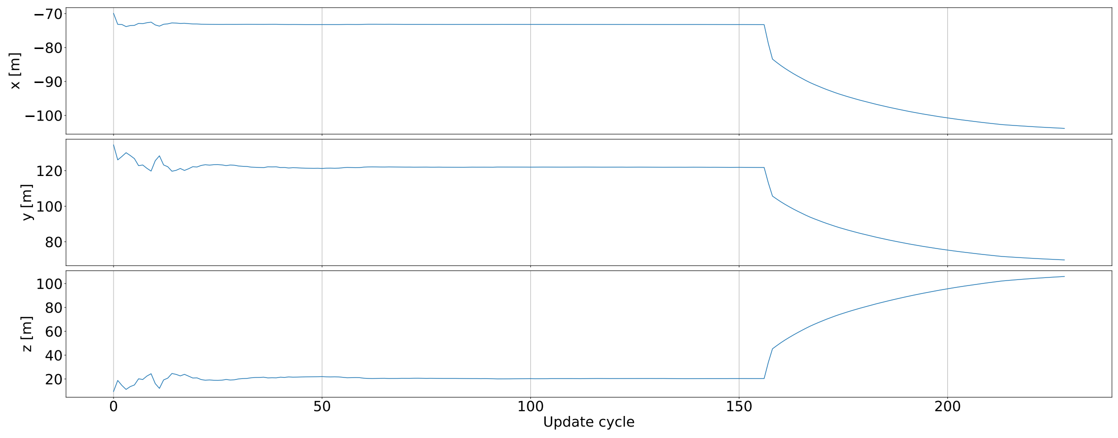
(b) Target yaw angle



(c) Standard deviations (y in log. scale)

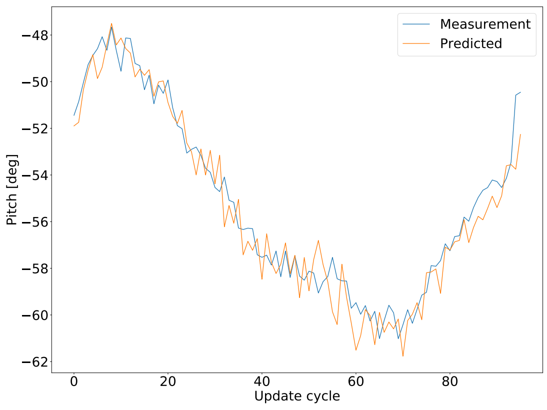


(d) Mahalanobis distance

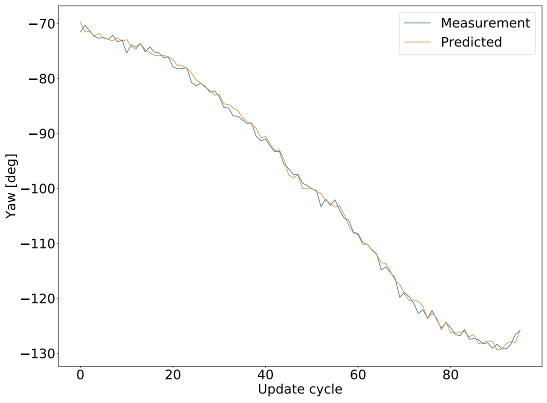


(e) Estimated target position

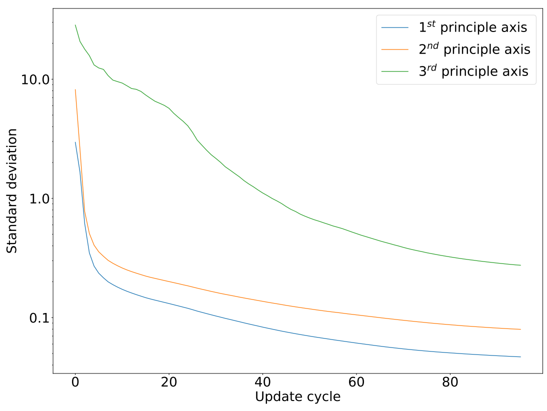
Figure 6.3: Data of outlier which is caused due to false feature matching, where x-axis represents the iteration number. After an initial phase of convergence a substantial change in measured pitch and yaw occurs. This leads the estimation to change a lot and eventually converge to another position. The Mahalanobis distance of this update is very high which indicates a missmatch.



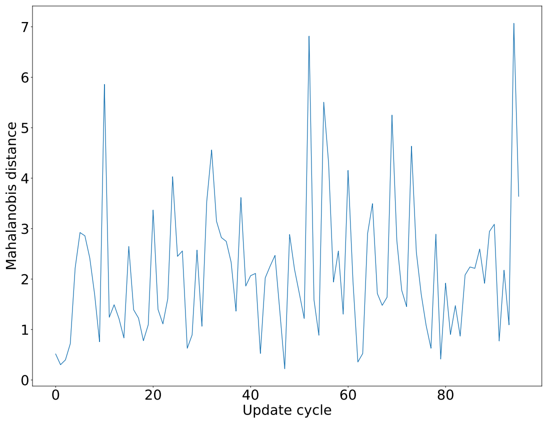
(a) Target pitch angle



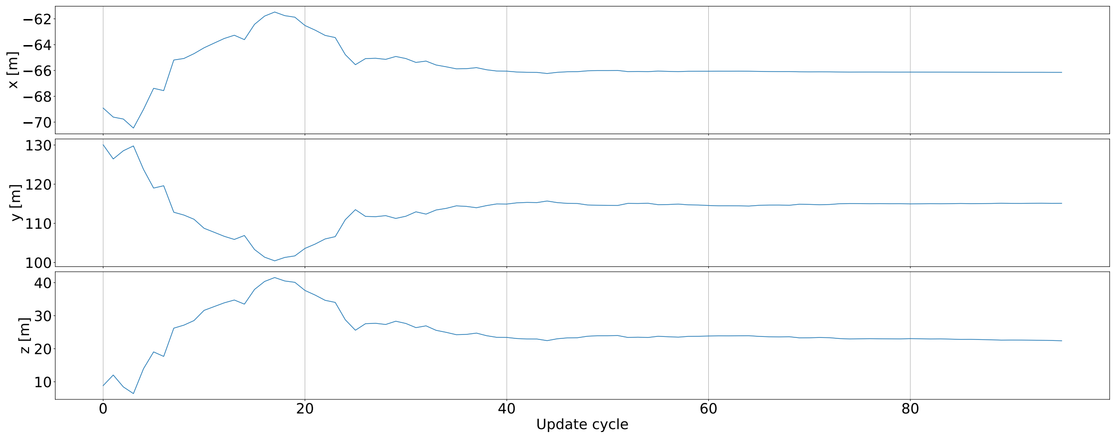
(b) Target yaw angle



(c) Standard deviations (y in log. scale)



(d) Mahalanobis distance



(e) Estimated target position

Figure 6.4: Data of one of the inliers, where x-axis represents the iteration number. The estimatiod target converges nicely and no sudded changes in yaw or pitch can be observed. The mahalanobis distance of all updates stays below 10.

When this criteria is applied, the number of outliers decreases significantly. The feature mismatches are completely avoided and the results converge to stable 3D positions as seen in fig. 6.5

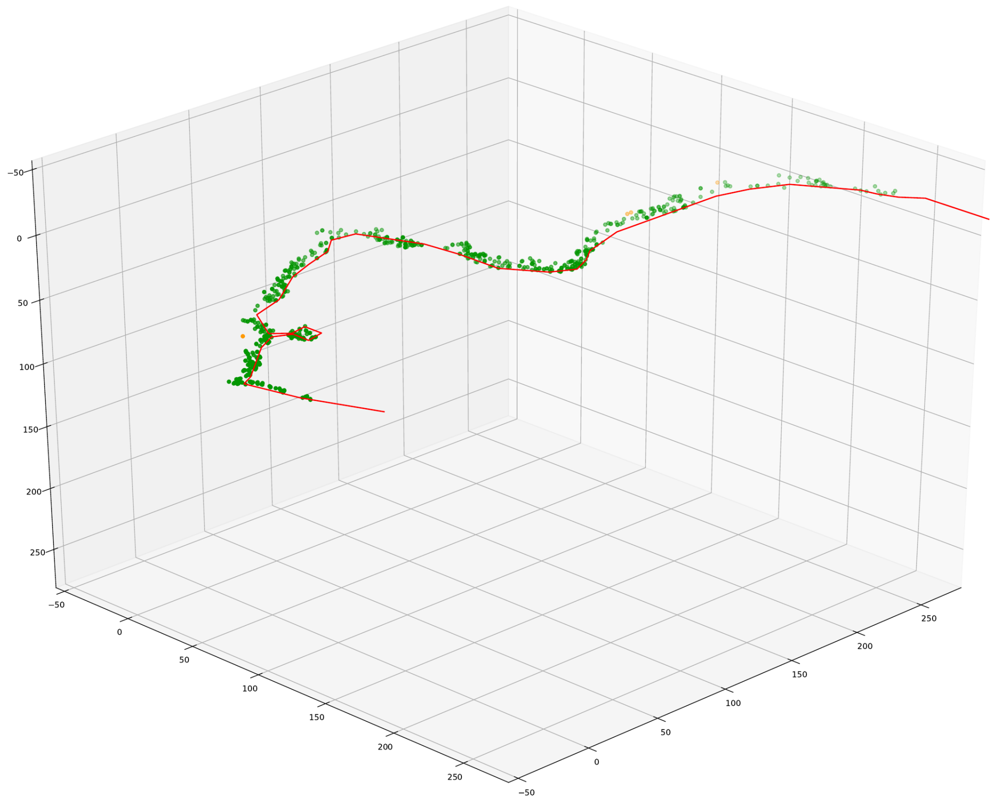


Figure 6.5: Results achieved with the base parameters shown in table 6.1. The red line denotes the ground truth, green points represent the inliers.

6.2.1 Base parameters

Based on the experience gathered during the development and the initial tests, a configuration of base parameters is chosen, which can be seen in table 6.1. These will then be varied to find ideal parameters.

speed	convergence	band width	mahalanobis dist.	downsclaing factor
3 m/s	0.25	100 px	10	2

Table 6.1: Baseline configuration for the parameters that will be varied.

Other parameters such as the waypoints used, the drone altitude and added errors stay the same over all tests and can be seen in table 6.2. Normally distributed

Altitude (local)	SD gimbal yaw	SD gimbal pitch	SD x	SD y	SD z
30 m	0.5°	0.5°	0.5m	0.5 m	2 m

Table 6.2: Values of the parameters that will stay fixed

errors with the indicated standard deviations are added to the measurements used in the algorithm. The values of standard deviation were chosen based on the estimated accuracy displayed by a DJI matrix 300 during a real test (see. section 6.10.2).

These parameters lead to the results which can be seen in table 6.3, fig. 6.5 and fig. 6.6.

Nr. of Landmarks	dist < 10m	10m $<$ dist $<$ 20m	dist $>$ 20m	avg. dist to GT
809	805	4	0	3.43 m

Table 6.3: Results achieved with the base parameters

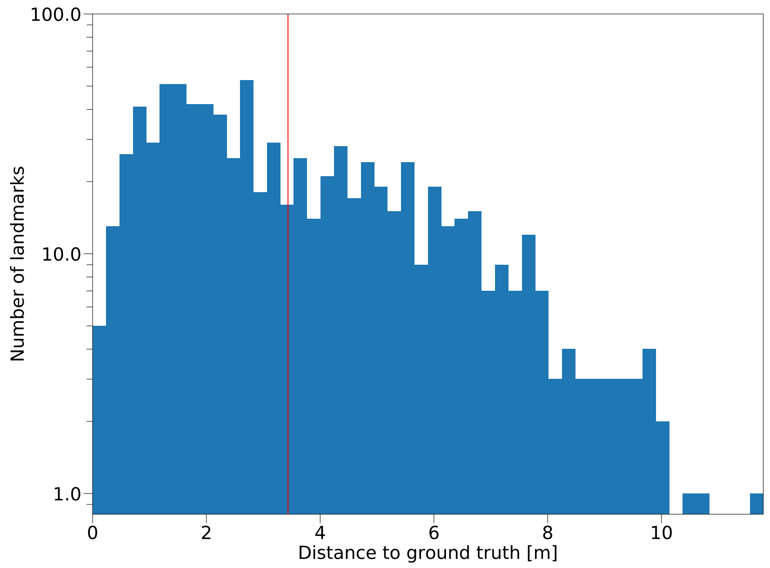


Figure 6.6: This histogram shows the distribution of the landmarks according to their minimal distance to ground truth. The red line indicates the average value. To make outliers visible, y-axis is in log scale.

6.3 Variation of drone speed

The localization algorithm requires tracking of features throughout its flight and requires a certain amount of time for the UKF estimates to converge. This process depends on the camera frequency and the speed of the drone.

While capturing datasets, AirSim provides images at a frequency of approximately 6Hz. Varying the speed of the drone leads to variation in the overlap between images. To test the maximum drone velocity which can still provide satisfactory results, the performance of the algorithm is evaluated while limiting the drone maximum velocity to 1, 3, 5 or 10 m/s. These velocity limits are applied to each axis separately. The output of the algorithm is then evaluated based on the distance between the landmark and the ground truth line.

6.3.1 Results

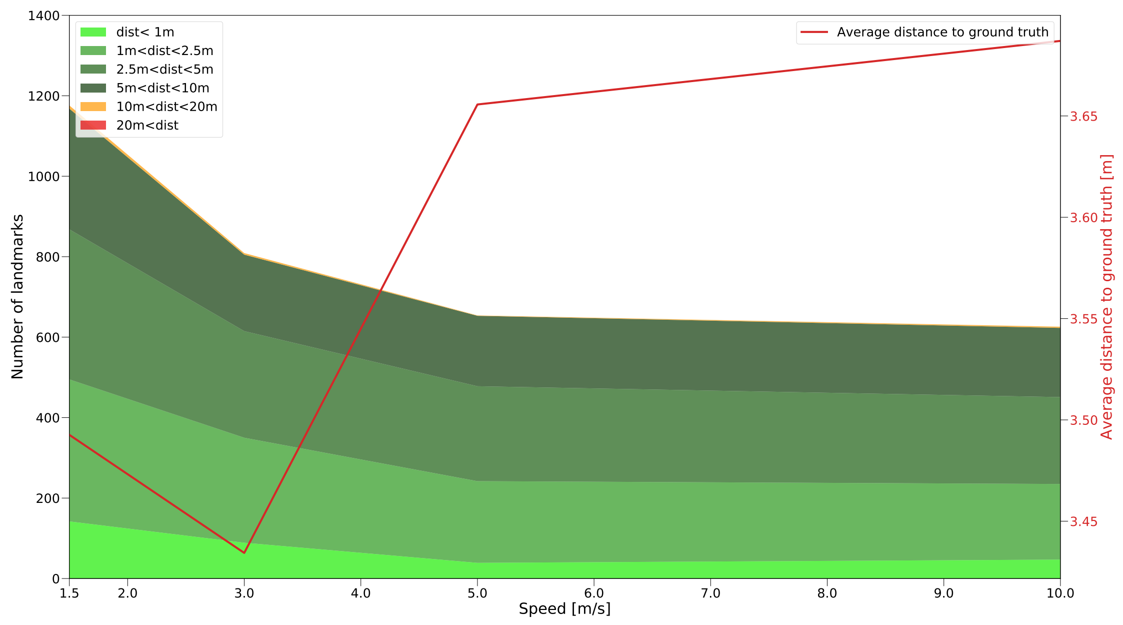


Figure 6.7: Displays the plot for various drone speeds vs the number of landmarks found. This plot also shows the average distance from the landmark to the ground truth for different drone speeds.

As can be seen in fig. 6.7, lower drone speed leads to a higher number of landmarks. The small difference between 5 and 10 m/s might be because the drone does not actually fly faster, even though the limits on the controller were set higher. The base value of 3 m/s seems to be a good tradeoff between the number of landmarks detected, average distance and drone speed.

6.4 Variation of convergence threshold

A landmark is considered good if it satisfies the convergence criteria (see (5.21) in section 5.5). It is therefore crucial to choose an optimal threshold to select the right landmarks.

6.4.1 Results

From fig. 6.8, it can be inferred that at higher thresholds the number of converged landmarks is higher but their quality is reduced. Values below a threshold of 1 show good results and 0.5 is chosen as a good compromise between the number of landmarks and their quality.

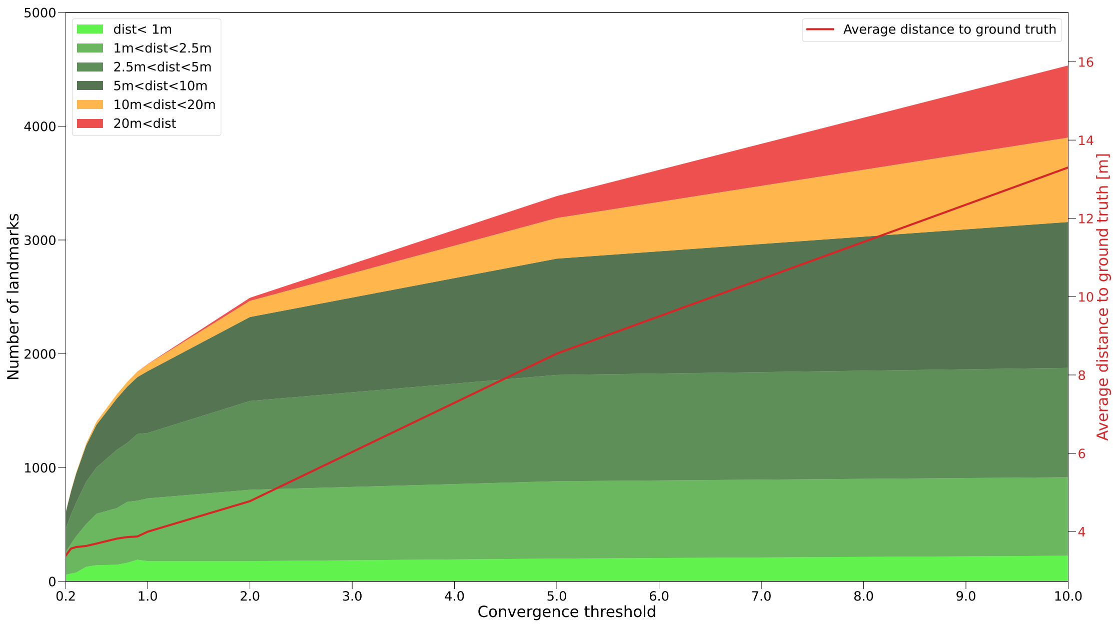


Figure 6.8: Displays the plot for convergence threshold vs the number of landmarks found. This plot also shows the average distance from the landmark to the ground truth for different band widths.

6.5 Variation of AoI band width

The size of the Area of Interest (AoI) band is a crucial tuning parameter for the algorithm. If it is chosen too small, not a lot of features might be found. If it is chosen too big, some of the found landmarks might be very far away from the border and the information gained might therefore be of reduced value. To find the optimal value, the drone mission is performed around the lake while varying the width of the band between 10 and 200 pixels. Since the downscaling factor of the base configuration is 2, the actual band width used by the algorithm varies between 5 and 100 pixels (see explanation in section 6.7).

6.5.1 Results

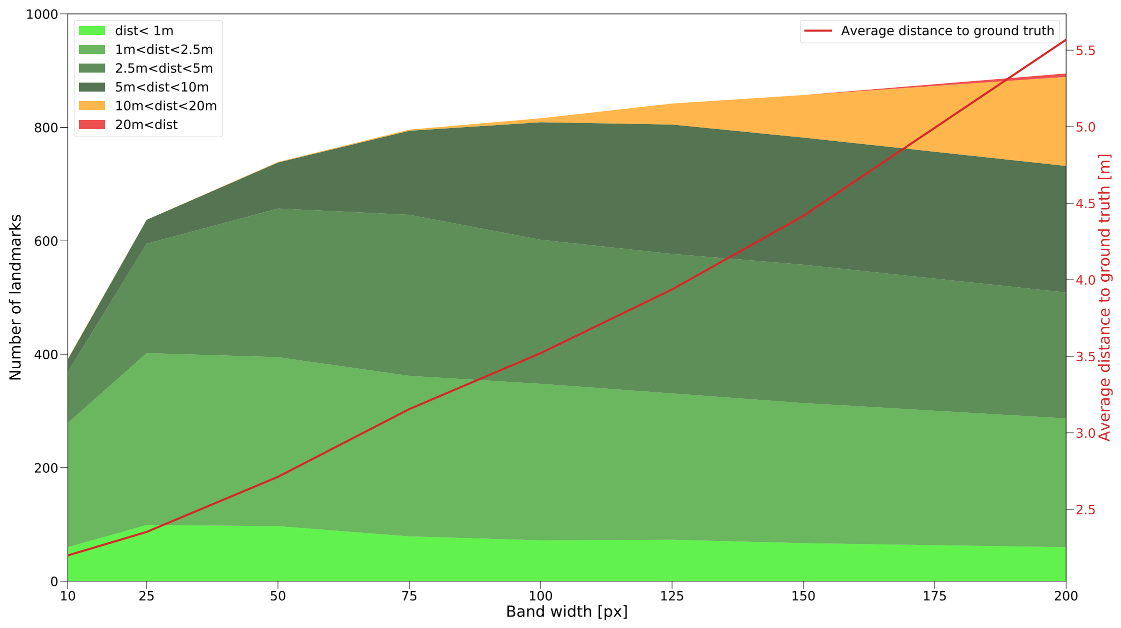


Figure 6.9: Displays the plot for various band widths vs the number of landmarks found. This plot also shows the average distance from the landmark to the ground truth for different band widths.

As expected, bigger band width leads to more but lower quality landmarks. The ideal value seems to be between 25 and 50 pixels where a high amount of good quality landmarks is present. It has to be said, that these results are related to the altitude the drone is flying at. If the flight height is changed significantly, the ideal band width might change.

6.6 Variation of Mahalanobis distance

The Mahalanobis distance threshold is used during the matching process to determine whether a matched feature belongs to the same landmark or not (see section 5.4.1). The threshold is varied between 2, the lowest setting that actually produces some converged landmarks, and 1000 which is basically equal to no thresholding at all.

6.6.1 Results

As can be seen in fig. 6.10, values between 3 and 4 seem to give the highest amount of good quality landmarks. Increasing the threshold allows more and more false matches to occur and therefore lowers the quality of the results. Lowering it below 3 results in the removal of many legitimate matches.

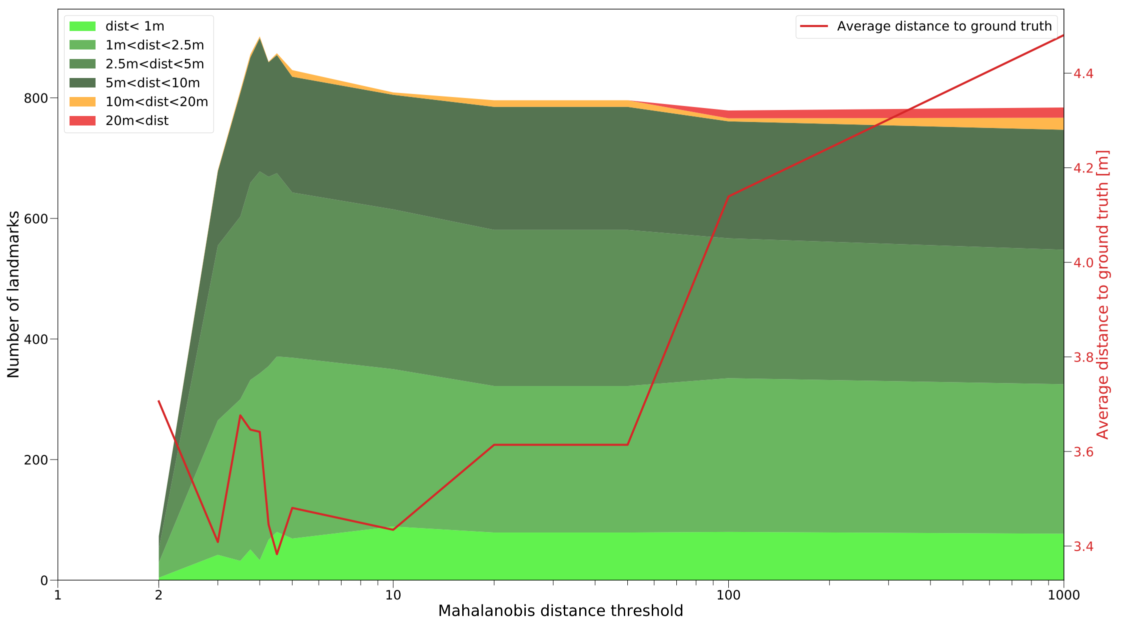


Figure 6.10: Displays the plot for various mahalanobis distance threshold vs the number of landmarks found. This plot also shows the average distance from the landmark to the ground truth for different mahalanobis thresholds. x-axis is displayed in log scale.

6.7 Variation of downscaling factor

The simulated drone is equipped with a full HD (1920x1080) camera. High Definition images provide more detail, which is advantageous when performing feature matching. In this test, the input full HD image is downscaled by factors of 2, 4 and 8 to evaluate the performance of the developed algorithm on images with lower resolution. Since the band width is expressed in pixels at full resolution, it is also reduced with the downscaling factor (e.g. band width 100 with downscaling factor 4 means that the algorithm uses a band of 25 pixels in the low resolution image). Choosing the factors as powers of 2 should lead to ideal downscaling results by avoiding any interpolation between pixels.

6.7.1 Results

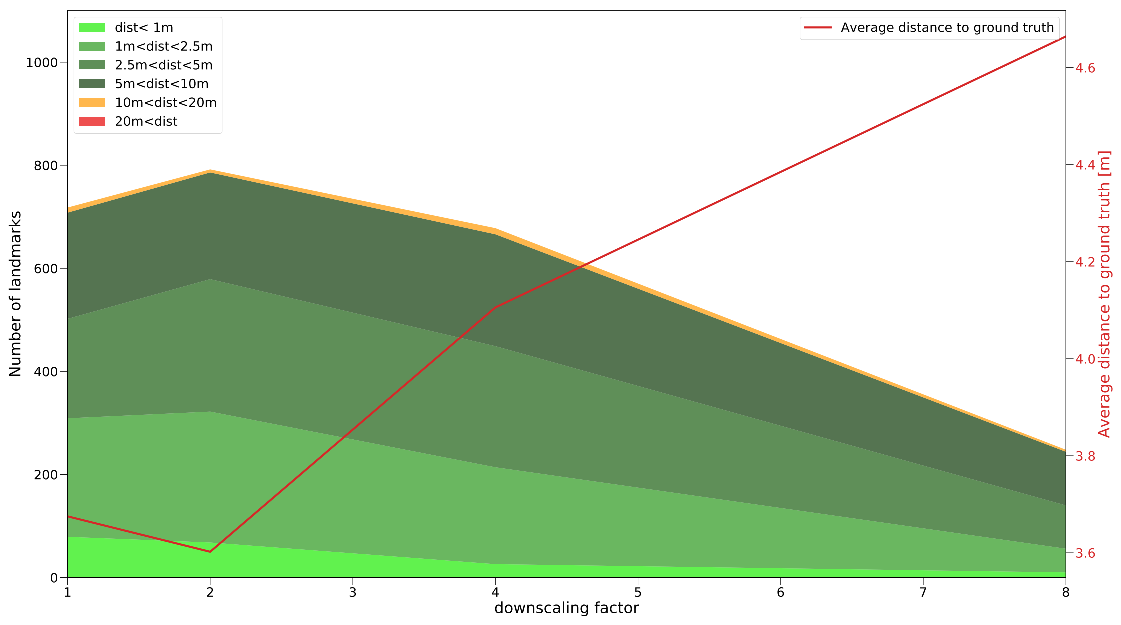


Figure 6.11: Displays the plot for various downscaling factors. vs the number of landmarks found. This plot also shows the average distance from the landmark to the ground truth for different image resolutions.

As can be seen in fig. 6.11, a downscaling factor of 2 gives not only the highest number of landmarks, but also the lowest average distance to ground truth. The only reason to choose another downscaling factor would be to improve computation speed which was not evaluated in this test.

6.8 Best performing parameters

Based on the tests described above, the set of best performing parameters is chosen, which can be seen in table 6.4.

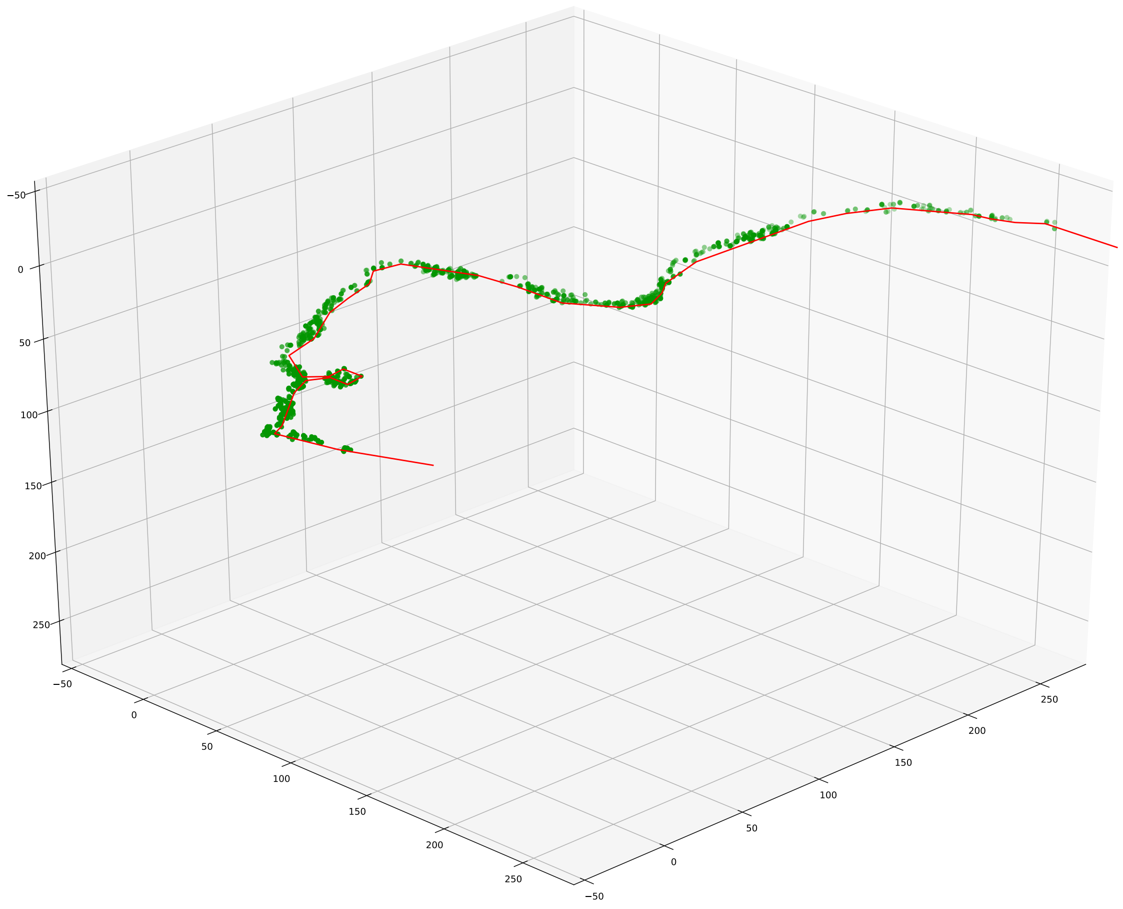


Figure 6.12: Results achieved with the best performing parameters shown in table 6.4. The red line denotes the ground truth.

speed	convergence	band width	mahalanobis dist.	downscaling factor
3 m/s	0.5	50 px	3.75	2

Table 6.4: Configuration with the best performing parameters.

The results obtained by using these parameters can be seen in fig. 6.12, table 6.5 and fig. 6.13. Comparing them with the initial results (see section 6.1), shows a significant improvement. While the 3D plots look relatively comparable, the number of landmarks has doubled while the average distance to ground truth has been reduced by almost 20%.

Nr. of Landmarks	dist < 10m	10m < dist < 20m	dist > 20m	avg. dist to GT
1631	1631	0	0	2.82 m

Table 6.5: Results achieved with the best performing parameters

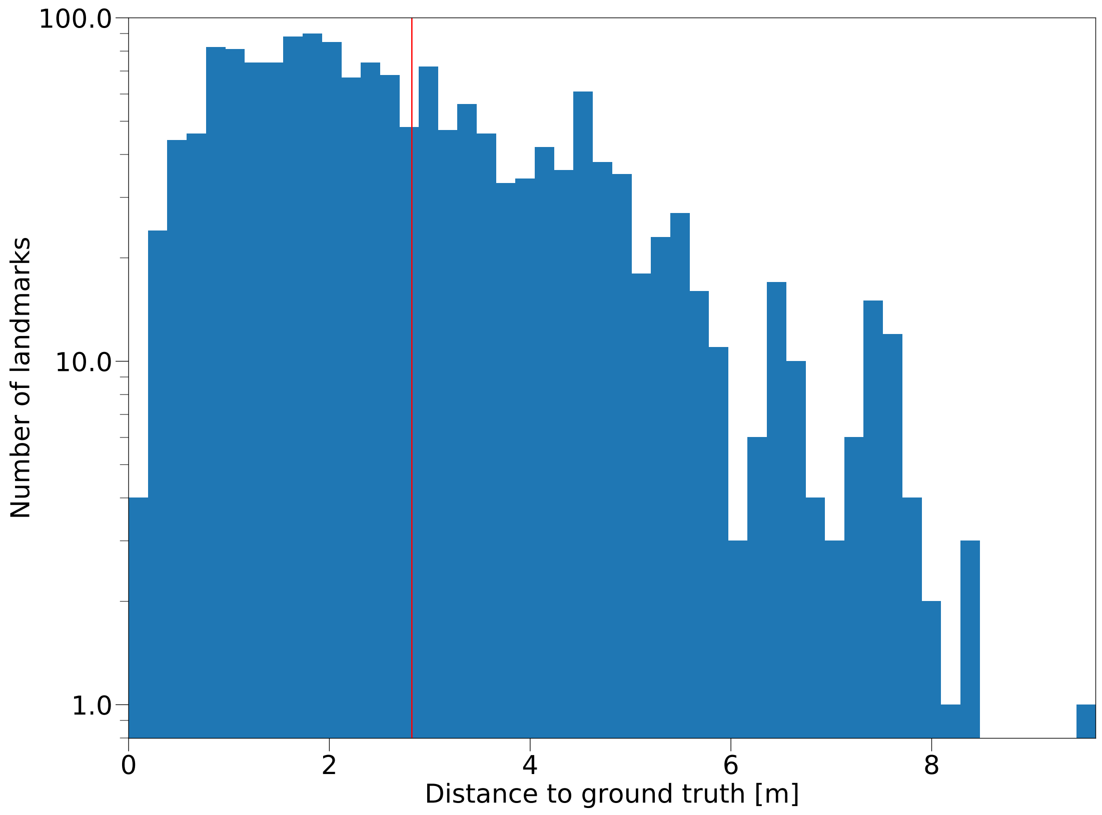


Figure 6.13: This histogram shows the distribution of the landmarks according to their minimal distance to ground truth. The red line indicates the average value. To make outliers visible, y-axis is in log scale.

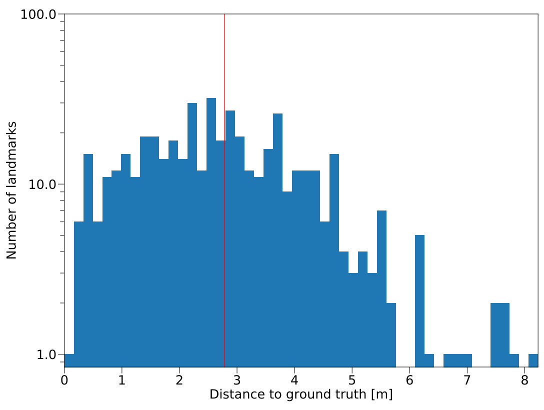
6.9 Cliff mission

The tests described above have all been performed on the coastline mission as described in section 4.2.4. Since the area to be localized is water, all landmarks are located in the same plane. This does not demonstrate the 3D capability of the developed algorithm. Furthermore, all parameters have been tuned on that example. It is therefore possible, that the parameters do not fit the algorithm in general, but the mission used to evaluate them.

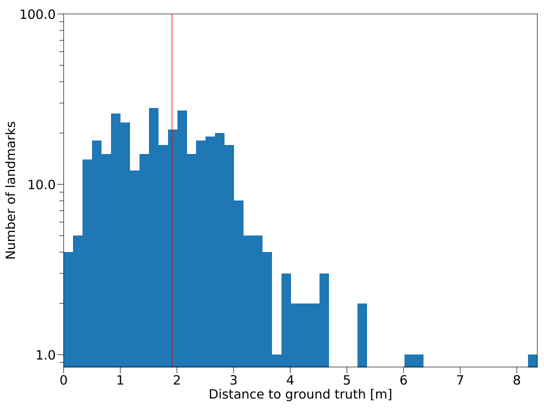
For this reason, the performance of the algorithm is evaluated using a second mission with the initial and the best performing parameters. The mission used is the cliff mission described in section 4.2.4. As can be seen in fig. 6.15, the 3D shape of the cliff is captured well with both settings. The best performing parameters once again lead to a shorter average distance to the ground truth. This happens at the cost of a lower number of converged landmarks. It is therefore a trade-off between quality and quantity of the resulting data.

	Landmarks	$d < 10m$	$10m < d < 20m$	$d > 20m$	avg. d to GT
Initial	468	468	0	0	2.78 m
Best performing	354	354	0	0	1.91 m

Table 6.6: Results achieved with the best performing parameters

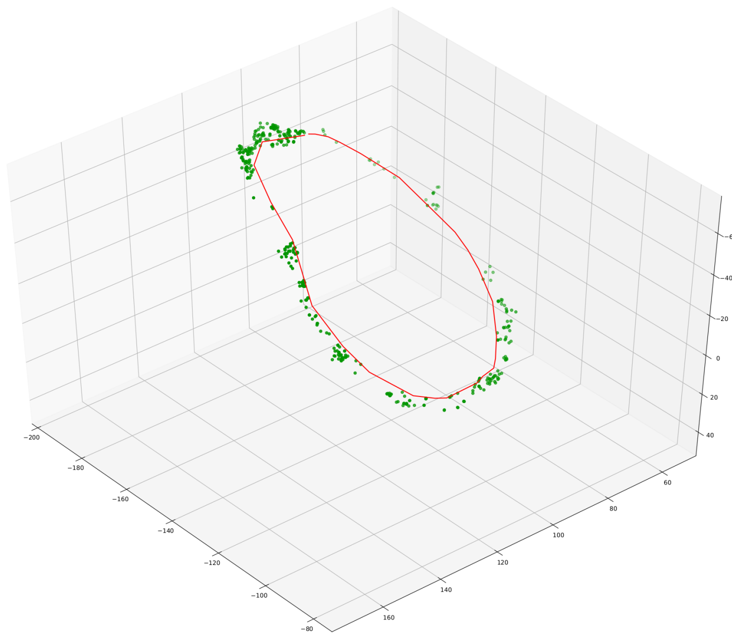


(a) Initial parameters

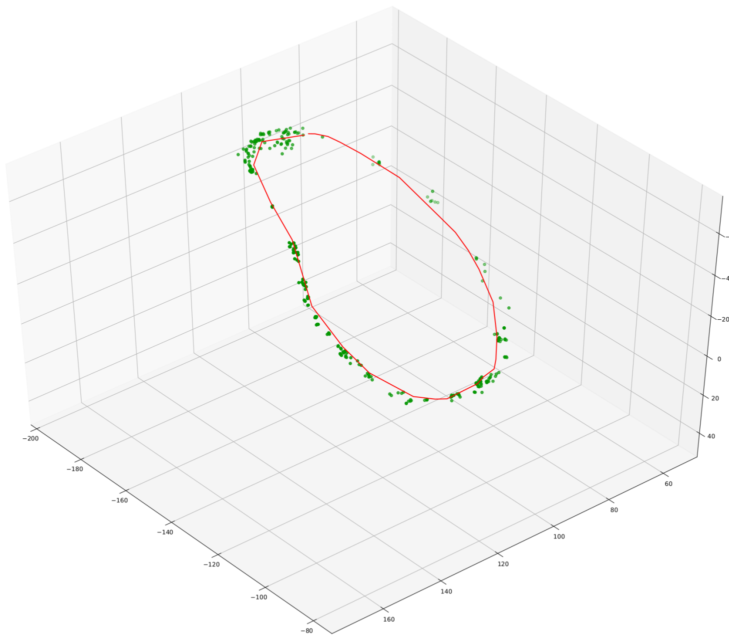


(b) Best performing parameters

Figure 6.14: Comparison of the histograms. y-axis in log scale.



(a) Initial parameters



(b) Best performing parameters

Figure 6.15: Comparison between the results from initial and best performing parameters applied to the cliff mission.The red line marks the manually generated ground truth. Both results capture the 3D shape of the are very well.

6.10 Influence of static errors

All tests so far have been performed with random, normally distributed errors added to both gimbal pitch and gimbal yaw measurements (see section 6.2.1). However, the normal distributions used were centred at the actual measurement. It is therefore not surprising, that the Kalman filters could easily deal with the introduced errors. As will be shown in section 6.11, static errors which move the center of the distribution away from the actual value also pose a real problem. Tests are therefore performed to evaluate the influence of static offsets on both the measured gimbal pitch and gimbal yaw angles. The parameters used are the best performing parameters, mentioned in section 6.8. As in all tests before, the standard deviations for the distribution of gimbal yaw and gimbal pitch are 0.5°each.

6.10.1 Yaw offset

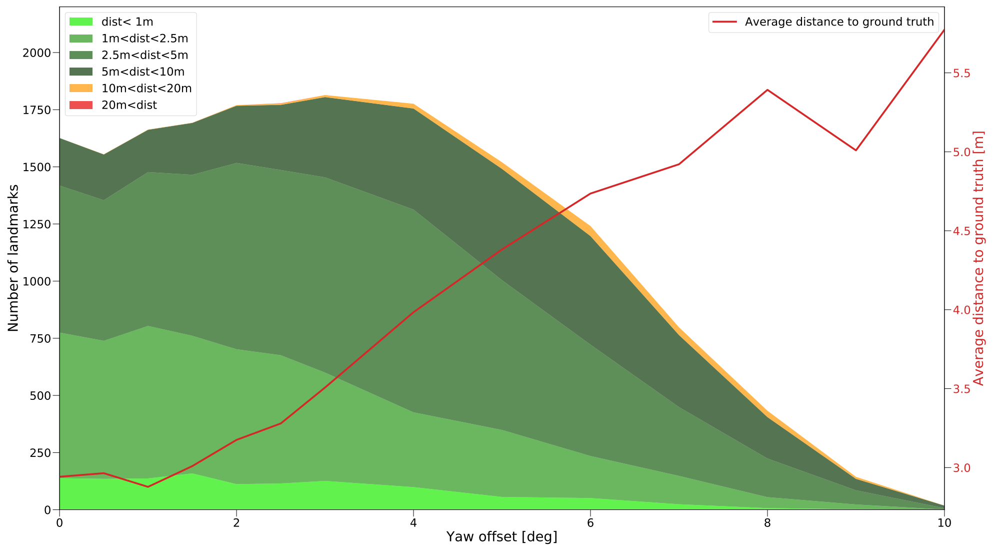


Figure 6.16: With increasing yaw offset the number of landmarks that fully converge diminishes. While the average distance to the ground truth increases steadily, it stays fairly low even with extreme offsets.

The effects of applying a static yaw offset can be seen in fig. 6.16. While average distance from the ground truth increases steadily with increasing offset, there seems to be little impact below 2 degrees. With higher values above 4 degrees, the number of converged landmarks decreases sharply. While extreme 10 degrees of offset eliminate almost all landmarks, the remaining few ones are still relatively close to the ground truth.

The 3D plot shown in fig. 6.16, displays the results achieved with an offset of 5 degrees. It is visually almost indistinguishable from the one taken without any offset (fig. 6.12). This shows that the algorithm can compensate for yaw offsets relatively well. In the case of extreme offsets above 5 degrees, landmarks are rejected instead of being estimated in a wrong position. This reduces the amount of available data but still maintains the quality of the result.

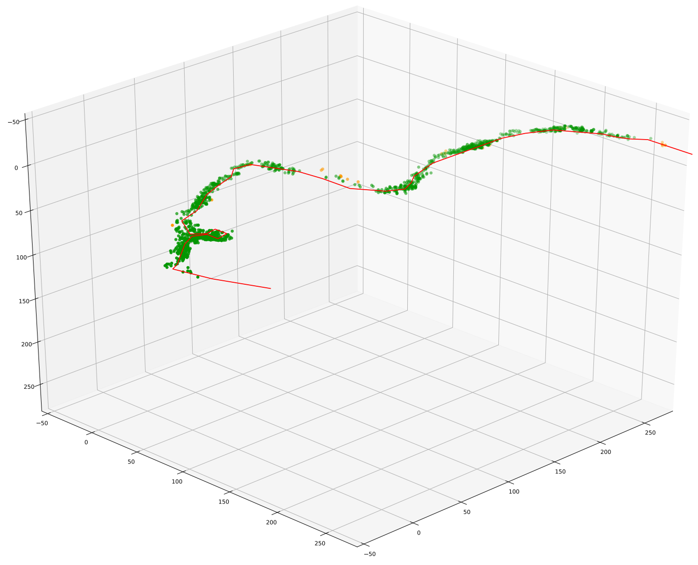


Figure 6.17: A yaw offset of 5 degrees does not affect the results in a visible way (compare fig. 6.12 for no offset).

6.10.2 Pitch offset

The effects of applying a pitch offset are much more severe. In fig. 6.18 it can be seen that while the number of converging landmarks stays relatively high, the average distance to the ground truth greatly increases with increasing pitch offset. When visually inspecting the results in fig. 6.19, it can be observed that the major error generated is along the z-axis. The localization in x- and y-axis is not affected by the addition of pitch offset. The average distance to the ground truth increases linearly with increasing pitch offset. While the effects are noticeable for all values tested, small offsets below 2 degrees still produce acceptable results.

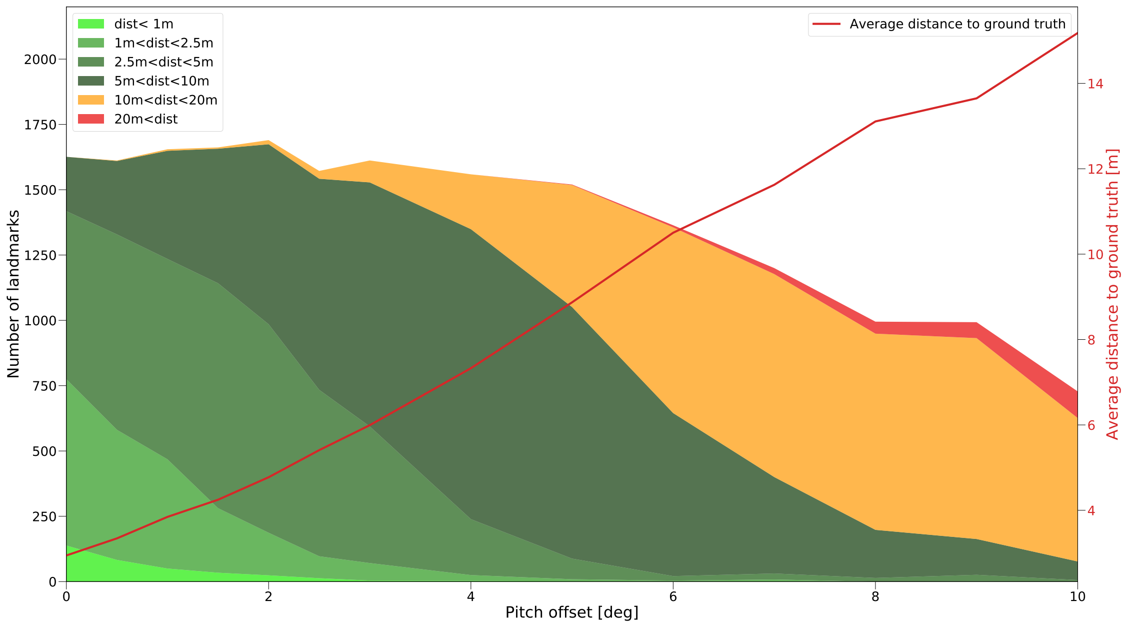


Figure 6.18: With increasing pitch offset the average distance from ground truth increases greatly. As can be seen in fig. 6.19, this increase is mostly along z axis.

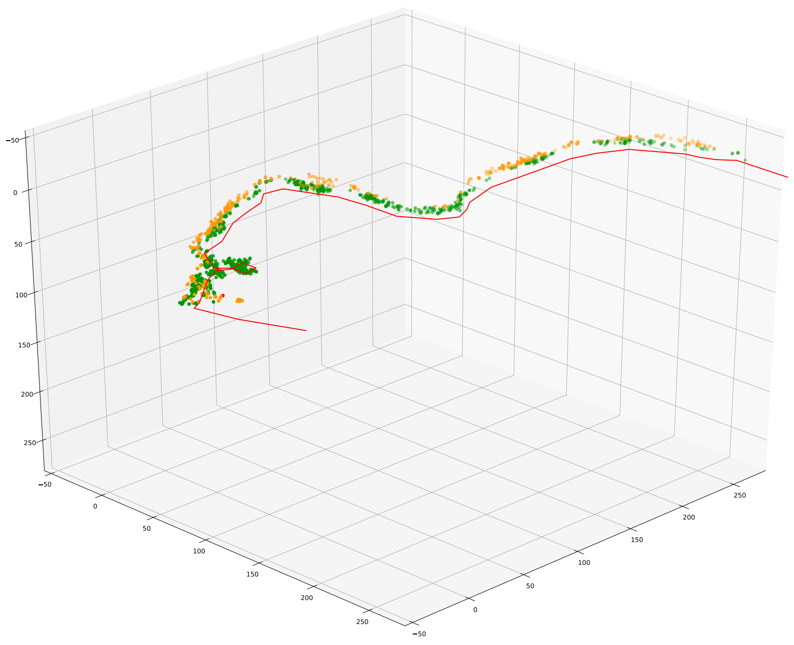


Figure 6.19: With a pitch offset of 5 degrees, it can be clearly seen, that the results are offset along the z-axis.

Real World Test

6.10.3 Used Hardware

The test was conducted with a DJI Matrice 300 provided by Robotto. A DJI D-Real Time Kinematic (RTK) ground station was used to provide RTK-GPS. The camera used was the wide RGB lens of a DJI Zenmuse H20T.

6.11 Previous method test

This test was performed to evaluate the accuracy of the previous method as described in 2.1. It predates most of the work in this thesis and played a vital role in the decision to find a new method, instead of improving the old one.

6.11.1 Setup

The test was conducted on a soccer field close to Aalborg University. The RTK base station and 2 additional markers are placed on the field. They serve as ground reference points as shown in Figure 6.20. The RTK base station serves as the origin for the local coordinate system. The ground truth for the Center and Goal markers was established by using RTK-GPS. These markers were then photographed from 8 predefined positions as seen in Figure 6.21, at an altitude of 10 m, 30 m and 50 m resulting in 72 data points as seen in Figure 6.22. Furthermore, the area of the test was mapped using automatic mapping missions created with DJI tools. These images were then used to create a complete 3D reconstruction of the area using Pix4D. This includes a self-calibration of the camera, which is performed during bundle adjustment. The intrinsic matrix gained from this calibration was used in further steps.

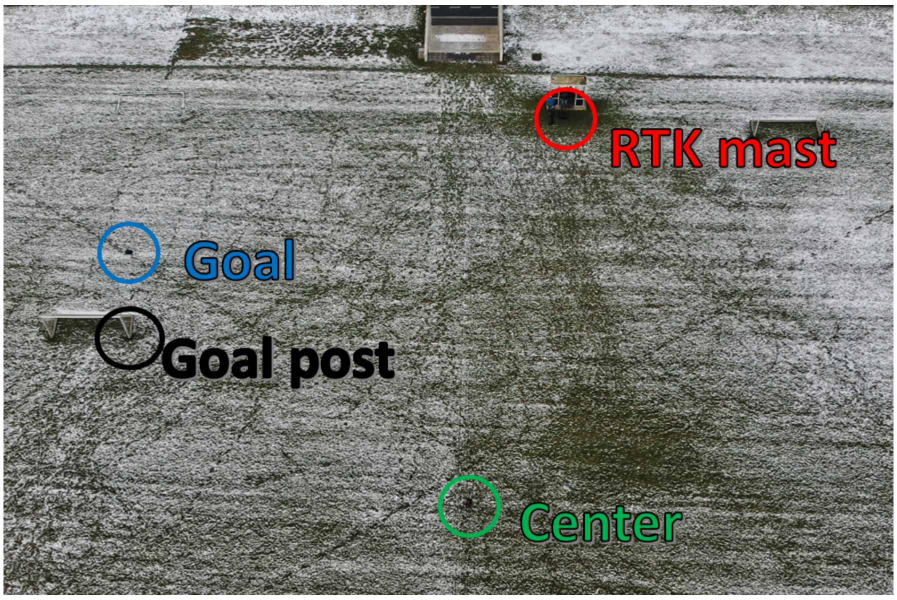


Figure 6.20: The ground reference points used during testing where, the RTK mast is denoted by red, marker in the center of the field is denoted by green, marker near the goal is denoted as blue and common reference point on the goal post is denoted by grey.

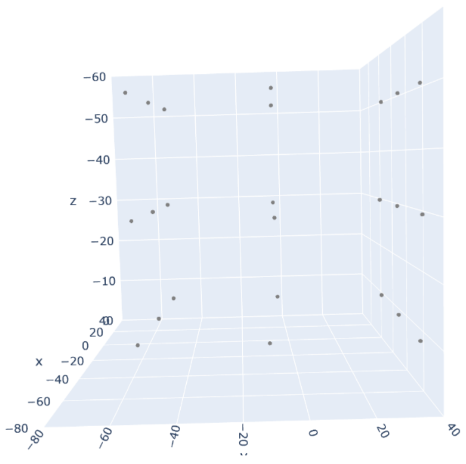


Figure 6.21: Camera positions at 10 m, 30m and 50m in NED frame.

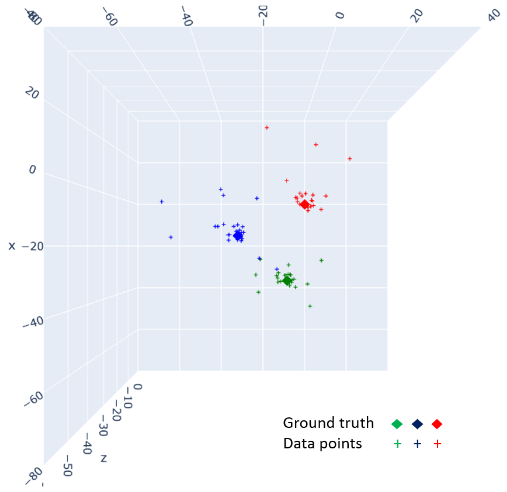


Figure 6.22: Data points generated from different camera positions

6.11.2 Procedure

The markers are placed at their respective locations and their GPS locations were recorded by measuring the drone GPS at that location, displayed in Table 6.7. A mission plan is developed to fly the drone around the football field at different altitudes - 10 m, 30 m, and 50 m. At each height, the drone is stopped at 8 different locations. The gimbal is then manually controlled to focus on each individual marker placed to capture images with metadata information. After acquiring the image dataset, the pixel locations of every marker visible in the images are recorded manually. They are stored along with the camera intrinsics and the drone location at the time of capture.

Marker	ID	Lat	Lon	x	y	relative altitude
RTK ground station	R	57.0134748	9.9742011	0.00	0.00	0.00
Backpack "Center"	M	57.0131517	9.9740606	-36.55	-8.45	0.00
Backpack "Goal"	G	57.0133455	9.9736700	-14.97	-32.12	0.00

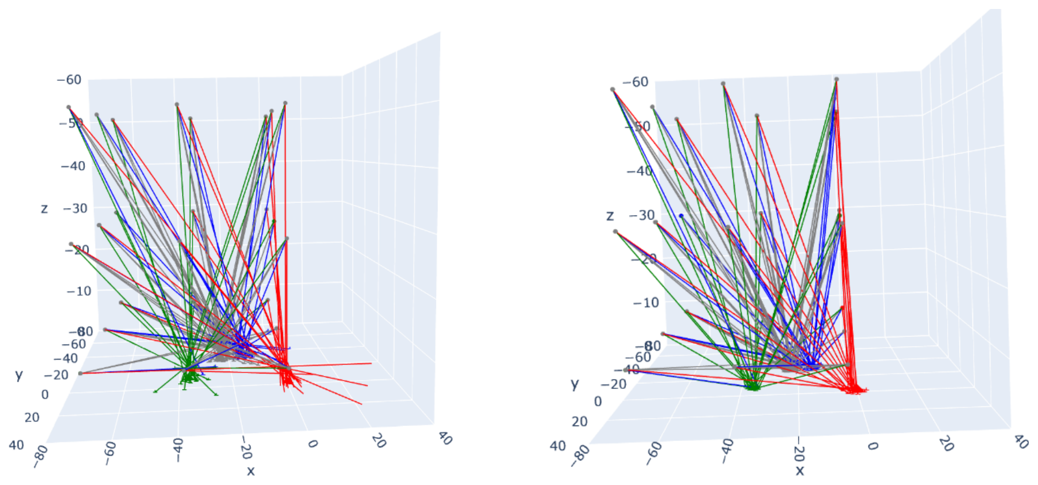
Table 6.7: Markers placed on the field.

This data was then used to calculate the location of these markers through the method explained in section 6.11 and is compared with their respective ground truths recorded (see Figure 6.23a). This prediction was observed to be very inaccurate, especially at shallow pitch angles. It was hypothesised that, these might be static errors due to bad calibration of the gimbal angles. Because of the nonlinear effect of the pitch angle on the results, errors would be disproportionally amplified at shallow angles.

	Roll	Pitch	Yaw
Calculated offset	0.60°	-2.16°	-0.34°

Table 6.8: Offset corrections calculated by least squares optimisation.

In order to correct this, least square optimization was used to calculate the static errors in the gimbal angles by using the RTK position as the reference. The result of this optimisation produced static offset corrections for each axis which can be seen in table 6.8. These can then be used to re-calculate the positions of the other markers as shown in Figure 6.23b.



(a) Using previous method *(b) After least square optimization*

Figure 6.23: Prediction of marker locations at different camera positions.

6.11.3 Results

These original results along with the optimised results can be seen in Table 6.9. The initial results are bad, especially when considering the low altitude and flat terrain. When least square optimization was used to adjust the gimbal angles, the results were greatly improved. This proves the initial hypothesis, that the errors generated were, at least partly, due to static errors in the gimbal angles. It is therefore necessary to perform an initial calibration around the RTK mast to calculate the static errors in the gimbal angles at the the beginning of a flight mission.

While the accuracy reached with least square optimization is satisfactory, it still has to be taken into account that this test was performed on a flat soccer field. Any kind of height deviation would negatively impact the results. This is one of the main reasons, why this approach was not followed up on.

Static error will also occur in the new method developed later during this thesis. Even though it uses Kalman filtering to eliminate normally distributed errors, the mean of this normal distribution still has to be accurate.

	original data		calibration on RTK data	
Marker	mean error	std deviation	mean error	std deviation
RTK ground station	8.811	10.727	2.154	1.250
10 m	17.701	14.737	3.111	1.440
30 m	4.299	1.679	1.531	0.961
50 m	3.870	1.858	1.742	0.645
Backpack "Center"	11.051	12.584	2.031	1.372
10 m	21.589	15.985	2.212	1.802
30 m	5.117	3.422	1.832	1.246
50 m	5.128	4.012	2.027	1.051
Backpack "Goal"	6.901	5.651	1.693	0.919
10 m	14.211	4.043	1.866	1.086
30 m	3.591	1.492	1.422	1.065
50 m	3.400	1.215	1.779	0.661
Goalpost "Post"	9.295	10.795	2.213	1.948
10 m	19.090	14.433	3.346	2.760
30 m	5.105	2.847	1.575	1.052
50 m	4.215	2.574	1.754	1.129

Table 6.9: Mean errors and standard deviation of original calculated data points and the optimised data points

7 Discussion

The idea proposed by the authors was successfully developed and it produced very promising results. The analysis of the algorithm helped the authors to calculate the best parameters for the algorithm. Using these parameters, a clear point cloud boundary of the area was produced with almost no outliers. To develop this algorithm, the authors made a few simplifications. These influenced on how the algorithm is to be developed and how it can be transferred to a real-time implementation.

One of the major points to be addressed is how the algorithm has been implemented in the simulation environment. As previously mentioned in Section 4.2.4, the algorithm is not run directly on the simulation. The simulation was only run only to collect the necessary data to run the algorithm offline. AirSim by itself takes a lot computation power, since it simulates high visual fidelity. When added to the implementation of algorithm, the entire process slows down heavily. Therefore, a decision was made to run the two processes separately at a decent speed, rather than running both simultaneously at a slow speed. Moreover, it allows the authors to test the algorithm on the stored dataset again and again, without the need of rerunning the simulation. Thereby saving a lot of time.

The second point to be discussed is the transfer of the algorithm to an actual drone. To facilitate the transfer, some of the basic requirements are:

- Hardware Requirements - The processor available must be capable of running image segmentation, feature detection and matching simultaneously.
- Area detection and segmentation algorithm - In the current implementation, the segmented image is received from simulation. Therefore, an additional area detection and segmentation algorithm is required based on the type of area selected.
- Path planning - The main focus of the thesis is area localization. The path planned for the drone in the simulation was calculated assuming a prior knowledge of the ground truth. In an actual implementation, the path would have to be calculated dynamically based on the situation.
- SIFT GPU implementation - The slowest part of the algorithm is the feature detection and matching. This part can be optimized by replacing it with

a GPU implementation of SIFT like cudaSift [25] and popsift [26]. SIFT GPU implementation was previously attempted by the authors, but was later dismissed because it failed to function optimally with the structure of the algorithm. GPU SIFT optimizes the task of feature detection and matching, assuming, that features are matched between two images. However in the algorithm, the feature matching is performed between the LoL and an image. This requires the transfer of information from GPU to CPU and back, which is a very time consuming process. To incorporate GPU SIFT into the structure of the algorithm, the entire handling of the LoL would have to be done on GPU. This is not the objective of the thesis and was therefore dismissed.

In Chapter 6, the results of the algorithm were tested for various boundary conditions, with addition of noise and the best parameters were calculated for optimal results. The algorithm is robust and provides satisfactory results even for unoptimised parameters. The algorithm performs well for downscaled images and is able to detect and match features consistently up to a downscaling factor of 8. While low drone speeds lead to better results, even when tested at the drone's maximum speed, the algorithm performed well. Increasing the band width, also increases the amount of low quality landmarks detected. This is to be expected, as the region of feature matching increases, more landmarks will be detected. This indicates that, the algorithm is performing well in finding landmarks and localising them. However, not all the landmarks are useful in calculating the boundary of the area. By varying the size of the band width, the boundary calculated would be affected significantly as seen from fig. 7.1. Therefore, a balance of the number of landmarks detected and the accuracy of the boundary is necessary while selecting the band width around the target area.

The main parameter to remove outliers is the Mahalanobis distance. With an increase in the mahalanobis distance threshold, the number of outliers also increases. The total number of landmarks detected remains almost the same after a certain value. At high mahalanobis distance thresholds, the feature matcher adds outliers into the LoL. This leads to further mismatches and the estimation worsens. These outliers, would certainly be impactful during the calculation of the boundary of the target area. However,from a visual point of view, even with high mahalanobis distance the final results can still be visualized satisfactorily.

To have a better and faster convergence of the position estimate, the initial estimate chosen was taken as the 3D reprojection of the pixel values at an assumed ground plane. This improved the speed of convergence, although the effect was not documented through tests.

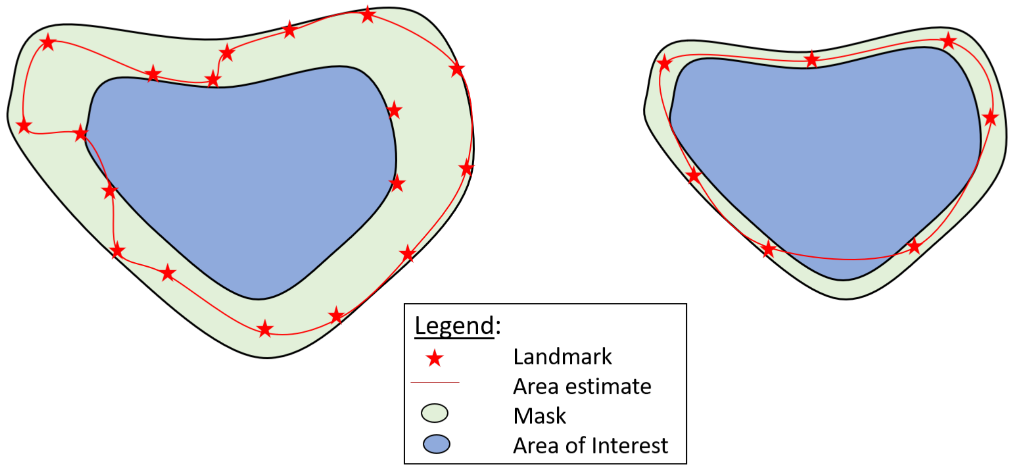


Figure 7.1: The figure of the left shows the result with a high offset mask and the figure on the right shows the result of lower offset mask.

Furthermore, due to the accurately segmented images acquired from the simulation, the effect of an faulty or noisy segmentation on the final results was also not tested.

Boundaries of an area are best represented as a 3D line. Therefore, it would be desirable to calculate a 3D line from the generated pointcloud. 3D curve fitting is still a topic of research and can be a separate project by itself. It is therefore not implemented.

The final tuned parameters provided a very accurate result for these specific landscapes. This might not be true for any area, and the parameters might have to be tuned again for different circumstances.

8 Conclusion

The algorithm developed successfully localizes the borders of 3D areas in simulation. The results obtained were tuned and the parameters that produce the best result were calculated. The final results are accurate and produce almost no outliers. While most of the tests were performed on an example mission without altitude changes, a second mission with big height differences was also tested and the algorithm performed equally well.

The set research and development goals were followed to achieve the final result. The algorithm performed well in all categories defined in the initial requirements:

1. The feature detector used for the project is SIFT, which is one the best feature detector algorithms. It is able to detect features consistently throughout the mission.
2. The final average distance obtained after using the tuned parameters was 2.82m, which satisfies the set requirement.
3. All the tests described above were conducted with the addition of standard deviation in position and gimbal angles. The Kalman filter corrects for the noise in the measurement and provides a filtered output.
4. The algorithm was tested for multiple camera resolutions. Although the number of total landmarks decreased with reduced image resolution, the outliers still remained to a minimum. Moreover, the average distance was below 5m even when images downscaled by factor 8 were used. Hence, this requirement is also satisfied.

The testing of the algorithm on a real dataset was delayed due to infrequent availability of the drone. The authors therefore directed their full attention on the simulation implementation. This resulted in an algorithm which performs exceptionally well in the simulation environment.

9 Future Works

While the algorithm is working well in simulation, there is a further work to be done until it can be deployed on a drone:

- The most important next step is to test the algorithm on a dataset captured on a real drone. This will show if the feature tracking also works as well on real images and if the data provided by the drone is accurate enough to provide good localization.
- Using real life data, it has to be established for which use cases the algorithm is actually suitable. In regard to the use for detecting wildfires, which was the original inspiration, it would be interesting to conduct tests using thermal images. These show fire very clearly but are not obscured by smoke.
- If this gives promising results, the algorithm will have to be further optimized so it eventually can be run directly on a drone. This will require rewriting a lot of the code so it can be executed on a GPU.
- The output of the algorithm is currently a point cloud. While humans can easily see the border of the area from the pointcloud, a solution should be found to extract the border as a 3D line. This would allow further processing by algorithms such as the path planner of the drone.

Bibliography

[1] S. Van Riel, “Exploring the use of 3d gis as an analytical tool in archaeological excavation practice,” Ph.D. dissertation, 2016, doi: 10.13140/RG.2.1.4738.2643.

[2] “Opencv reference manual,” OpenCV, accessed: 31.05.21. [Online]. Available: https://docs.opencv.org/4.4.0/d9/d0c/group__calib3d.html

[3] B. Berner, “Internship at robotto - drone based autonomous fire detection,” 2020, (Only accessible when logged in to AAU library), https://projekter.aau.dk/projekter/en/studentthesis/internship-at-robotto--drone-based-autonomous-fire-detection(17e3313b-7108-4785-8b04-d0c37616dee0).html).

[4] L. Merino, F. Caballero, J. R. Martinez-de Dios, I. Maza, and A. Ollero, “An unmanned aircraft system for automatic forest fire monitoring and measurement,” *Journal of Intelligent and Robotic Systems*, vol. 65, pp. 533–548, 2012, doi: 10.1007/s10846-011-9560-x.

[5] A. Dena and N. Aouf, “Unscented kalman filter for vision based target localisation with a quadrotor,” in *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2017, pp. 453–458, doi: 10.5220/0006474404530458.

[6] H. Hosseinpoor, F. Samadzadegan, and F. Dadrass Javan, “PRICISE TARGET GEOLOCATION AND TRACKING BASED ON UAV VIDEO IMAGERY,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B6, pp. 243–249, 2016, doi: 10.5194/isprs-archives-XLI-B6-243-2016.

[7] J. Li, C. W. Chen, and T. Cheng, “Estimation and tracking of a moving target by unmanned aerial vehicles,” in *2019 American Control Conference (ACC)*, 2019, pp. 3944–3949, doi: 10.23919/ACC.2019.8815101.

[8] X. Wang, J. Liu, and Q. Zhou, “Real-time multi-target localization from unmanned aerial vehicles,” *Sensors*, vol. 17, no. 1, p. 33, 2017, doi: 10.3390/s17010033.

[9] S. Ponda, R. Kolacinski, and E. Frazzoli, “Trajectory optimization for target localization using small unmanned aerial vehicles,” in *In AIAA guidance, navigation, and control conference, p. 6015.*, 2009, doi: 10.2514/6.2009-6015.

[10] D. Wang, D. Huang, C. Xu, and W. Han, “A closed-form method for simultaneous target localization and UAV trajectory optimization,” *Applied Sciences*, vol. 11, no. 1, p. 114, 2021, doi: 10.3390/app11010114.

[11] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[12] E. A. Wan and R. Van Der Merwe, “The unscented kalman filter for nonlinear estimation,” in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2000, pp. 153–158, doi: 10.1109/ASSPCC.2000.882463.

[13] R. Van Der Merwe, “Sigma-point kalman filters for probabilistic inference in dynamic state-space models,” Ph.D. dissertation, OGI School of Science & Engineering at OHSU, 2004.

[14] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” in *European conference on computer vision.* Springer Berlin Heidelberg, 2006, vol. 3951, pp. 404–417, doi: 10.1007/11744023_32.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571, doi: 10.1109/ICCV.2011.6126544.

[16] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “Kaze features,” in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Springer Berlin Heidelberg, 2012, pp. 214–227, doi: https://doi.org/10.1007/978-3-642-33783-3_16.

[17] S. A. K. Tareen and Z. Saleem, “A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK,” in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2018, pp. 1–10, doi: 10.1109/ICOMET.2018.8346440.

[18] I. Rey Otero and M. Delbracio, “Anatomy of the SIFT Method,” *Image Processing On Line*, vol. 4, pp. 370–396, 2014, doi: https://doi.org/10.5201/ipol.2014.82.

[19] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and Service Robotics.Springer Proceedings in Advanced Robotics*, M. Hutter and R. Siegwart, Eds., vol. 5. Cham: Springer, 2018, pp. 621–635, doi: https://doi.org/10.1007/978-3-319-67361-5_40.

[20] Epic Games, “Landscape mountains.” [Online]. Available: www.unrealengine.com/marketplace/en-US/learn/landscape-mountains

[21] National Imagery and Mapping Agency, “Department of defense world geodetic system 1984: its definition and relationships with local geodetic systems,” National Imagery and Mapping Agency, Redmond, WA, USA, Tech. Rep. TR8350.2, Jan. 2000, accessed: 17.12.20. [Online]. Available: http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html

[22] R. Labbe, “Filterpy,” *GitHub repository*, 2014. [Online]. Available: https://github.com/rlabbe/filterpy

[23] “Reprint of: Mahalanobis, p.c. (1936) "on the generalised distance in statistics.",” *Sankhya A*, vol. 80, no. 1, pp. 1–7, 2018, doi: https://doi.org/10.1007/s13171-019-00164-5.

[24] N. J. Higham, “Computing a nearest symmetric positive semidefinite matrix,” *Linear Algebra and its Applications*, vol. 103, pp. 103–118, 1988, doi: https://doi.org/10.1016/0024-3795(88)90223-6.

[25] M. Björkman, N. Bergström, and D. Kragic, “Detecting, segmenting and tracking unknown objects using multi-label mrf inference,” *Computer Vision and Image Understanding*, vol. 118, pp. 111–127, 2014, doi: https://doi.org/10.1016/j.cviu.2013.10.007.

[26] C. Griwodz, L. Calvet, and P. Halvorsen, “Popsift: A faithful sift implementation for real-time applications,” in *Proceedings of the 9th ACM Multimedia Systems Conference*, ser. MMSys ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 415–420, doi: https://doi.org/10.1145/3204949.3208136.