TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-21-01

Object Localization in 3D Space for UAV Flight using a Monocular Camera

Marco Stephan · Batbayar Battseren · Wolfram Hardt

März 2021

Chemnitzer Informatik-Berichte

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Object Localization in 3D Space for UAV Flight using a Monocular Camera

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc. Computer Science

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Marco Stephan
Student ID: 404257
Date: 04.02.2021

Supervisors: Prof. Dr. Dr. h. c. Wolfram Hardt
Prof. Dr. Uranchimeg Tudevdagva
M.Sc. Batbayar Battseren

Abstract

In this master thesis, a localization algorithm capable of localizing generic objects in a three-dimensional space is developed. This algorithm is to be used in unmanned aerial systems which are integrated into unmanned aerial vehicles or UAVs. Therefore, the inputs of the object localization are limited to the flight data of the UAV, the orientation of a gimbal mounted on the UAV and the two-dimensional positions of objects being detected in images shot via a camera placed on the gimbal. A monocular camera is used for this. The developed object localization algorithm shall further satisfy certain efficiency and accuracy constraints to be used for real-time collision avoidance. In particular, it is to be integrated into the Automated Power Line Inspection (APOLI) project carried out by the Professorship of Computer Engineering at the Faculty of Computer Science of the Chemnitz University of Technology.

Keywords: 3D Object Localization, Monocular Vision, UAV, Triangulation, Multi-View

Table of Content

List of Figures

List of Tables

List of Abbreviations

UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aerial System
APOLI	Automated Power Line Inspection
RTL	Real-Time Object Localizer
FLC	Flight Controller
EXS	Expert System
CTH	Control Handler
MAL	MAVLink Abstraction Layer
CGC	Camera Gimbal Controller
FPD	Feature Point Detection
GNSS	Global Navigation Satellite System
NED	North-East-Down
SVD	Singular Value Decomposition
SLAM	Simultaneous Localization and Mapping
CSV	Comma-separated values
IFC	Indoor Flight Center
RMSE	Root Mean Square Error

1 Introduction

1.1 Motivation

Computer vision [5], occasionally abbreviated to CV, is a field of study which seeks to develop techniques to understand the content of images or videos on a higher level. Examples of challenges tackled in computer vision are the stitching problem [5], [6], in which multiple images are overlapped in a way a single seamlessly stitched panorama image is generated, the recognition of fingerprints [5], [7], the detection of faces [5], [8] or, more generally, the detection of objects [5], [9] in image streams. Another fundamental challenge faced in computer vision which is still heavily researched is the vision-based three-dimensional object localization problem [10]. There, the location and, if feasible, the shapes of objects shall be determined in a three-dimensional space using, among others, data from image streams. Generally speaking, such object localization information is mostly used to generate an environment map, i.e. a map containing all objects that have been localized. There are a number of real-world applications for which an environment map is useful. Firstly, this environment map may be used to measure the size of objects or areas in general. Furthermore, it can also be used for route planning.

Another application of the environment map is the collision avoidance problem. There, the environment map generated by the object localization algorithm is used to avoid the collision between a moving vehicle and the objects of the map. In particular, such a vehicle may be an unmanned aerial vehicle [1], or UAV, which are aircrafts operating without a human pilot. The autonomization of the systems UAVs are embedded in, the so-called unmanned aerial systems (UASs), also has been a focus of research for several years now. Autonomously operating UASs shall start, navigate and land completely on their own without human interference. Most often, UASs are designed to integrate specific use cases. Examples for such use cases are aerial observations, logistics or mapping of environments. For the navigation process, collision avoidance is of crucial importance. While a static environment map provided to the UAS prior to the flight is more efficiently used by the UAS, a dynamically generated environment map generated by an object localization algorithm on-the-fly allows for more flexibility. The application of three-dimensional object localization algorithms on UASs, however, leads to additional challenges. Firstly, a UAS has in most cases rather limited computational resources available. Therefore, the object localization algorithm used must not be too complex to solve, computationally. Furthermore, most simple object localization approaches require accurate knowledge of the camera’s position, its

orientation as well as the location of the object in the images. For UAVs, however, all of this information must be assumed to be afflicted by some noise. Therefore, the object localization algorithm used for UASs must be able to deal with such noise.

In particular, the Professorship of Computer Engineering at the Faculty of Computer Science of the Chemnitz University of Technology carries out a research project which aims to develop a UAS capable of autonomously inspecting power lines and detecting faults of them. This is the Automated Power Line Inspection [1]–[4] project, hereinafter referred to as APOLI. The UAS inspects power lines by navigating the UAV around the power poles and evaluating the video footage captured by a camera mounted on the UAV's gimbal. Obviously, it is of crucial importance that the UAV does not collide with the power lines themselves. In the APOLI project, this collision avoidance problem, however, has not been tackled yet and a procedure for this has yet to be implemented. As stated before, if an environment map is available, the collision avoidance problem is a much more tractable task to solve.

1.2 Thesis Objectives

The goal of this master thesis is to develop an object localization algorithm which is capable of determining the location as well as, if appropriate, the shape of objects. This algorithm, referred to as the Real-Time Object Localizer or simply RTL, shall run in real-time on a UAS while the UAV is flying. More specifically, it is to be integrated into the APOLI project. As the environment map generated by the RTL shall be used for collision avoidance, it has to satisfy certain accuracy and robustness constraints. Furthermore, the input of the RTL is limited so only the flight data of the UAV, the orientation of the gimbal as well as the two-dimensional object position in RGB images captured using a monocular camera system are used. No information about the model of the objects that are to be localized is known prior to the localization. The RTL shall further be implemented in C++. For demonstration purposes and easier evaluation, the implementation of the RTL shall also render the environment map in a three-dimensional space.

While Chapter 2 elaborates the required fundamental knowledge, some state-of-the-art three-dimensional object localization approaches are presented in Chapter 3. In particular, the triangulation, described in Chapter 3.2, as well as the ellipsoid approximation approach of Chapter 3.3, are further explained in detail. Subsequently, Chapter 4 elaborates the complete concept of the Real-Time Object Localizer. Its implementation in C++ is briefly described in Chapter 5. Then, the RTL is evaluated in Chapter 6 in regards to its runtime, the accuracy as well as its robustness. Finally,

Chapter 7 provides a conclusion of the developed object localization algorithm as well as an outlook.

2 Fundamentals

This chapter introduces some fundamental knowledge required for the state-of-the-art object localization approaches presented in Chapter 3 as well as the conceptualization of the RTL in Chapter 4. While Chapter 2.1 and 2.2 explain mathematical fundamentals, Chapter 2.3 elaborates the APOLI project in greater detail. Subsequently, some of the flight data of UAVs is described in Chapter 2.4. Finally, Chapter 2.5 presents the concept of monocular RGB camera systems and highlights some of their advantages in comparison to other camera systems that may be used for object localization.

2.1 Projective Space

The n-dimensional Euclidean space $\mathbb{R}^n$ can be used to describe any n-dimensional point $\boldsymbol{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$. As we live in a three-dimensional world, the three-dimensional Euclidean space $\mathbb{R}^3$ is of particular importance to us. However, to ease some equations, especially in the context of projections, the so-called projective space $\mathbb{P}^n \coloneqq \mathbb{R}^{n+1} \backslash \{\mathbf{0}\}$ [5] may be used. This space extends the Euclidean space by adding an additional projective coordinate, which is mostly referred to as w. Given any point $\tilde{\boldsymbol{x}} = (\widetilde{x_1}, \dots, \widetilde{x_n}, \tilde{w})^T \in \mathbb{P}^n$, $\tilde{\boldsymbol{x}}$ describes the point

$$\boldsymbol{x} = \left(\frac{\widetilde{x_1}}{\tilde{w}}, \dots, \frac{\widetilde{x_n}}{\tilde{w}}\right)^T \in \mathbb{R}^n$$

of the n-dimensional Euclidean space. While $\boldsymbol{x}$ uses cartesian coordinates, $\tilde{\boldsymbol{x}}$ uses so-called homogeneous coordinates. Of particular importance is the observation that all $a\tilde{\boldsymbol{x}} = (a\widetilde{x_1}, \dots, a\widetilde{x_n}, a\tilde{w})^T \in \mathbb{P}^n$ with $a \in \mathbb{R}\backslash\{0\}$ describe the same $\boldsymbol{x} \in \mathbb{R}^n$. Therefore, when converting a point from cartesian coordinates to homogeneous coordinates, w can be chosen arbitrarily. To ease some equations, it is mostly set to $w = 1$. If not stated otherwise, throughout this thesis a vector with the tilde sign atop of it represents a homogeneous vector while the absent of it expresses the corresponding vector of the Euclidean space.

The homogeneous representation $\tilde{\boldsymbol{x}} \in \mathbb{P}^3$ of a point $\boldsymbol{x} \in \mathbb{R}^3$ can be used to apply any transformation on $\boldsymbol{x}$. Such a transformation is expressed via a transformation matrix $T \in \mathbb{R}^{4\times4}$. To apply the transformation T on $\boldsymbol{x} \in \mathbb{R}^3$, the equation

$$\tilde{\boldsymbol{x}}' \coloneqq T\tilde{\boldsymbol{x}}$$

is used. While the transformation matrix T may be of any form, of particular importance are the Euclidean transformations [5]

$$T \coloneqq \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}.$$

Euclidean transformations are composed of two components. The translation vector $\boldsymbol{t} \in \mathbb{R}^3$ described the translation of a point, i.e. how it is displaced. Furthermore, the orthonormal rotation matrix $R \in \mathbb{R}^{3\times3}$ expressed the rotation of the point around the origin. The inverse Euclidean transformation T^{-1} of T can be derived as

$$T^{-1} = \begin{bmatrix} R^T & -R^T\boldsymbol{t} \\ \boldsymbol{0^T} & 1 \end{bmatrix}. \tag{2.1}$$

It should be noted that this equation results from the observation that the inverse rotation of any rotation matrix $R \in \mathbb{R}^{3\times3}$ is equal to its transpose, i.e.

$$R^{-1} = R^T.$$

2.2 Camera Matrix

The camera matrix $P \in \mathbb{R}^{3\times4}$ [5] of a given pinhole camera describes the projection of three-dimensional points onto the two-dimensional image plane. The point $\boldsymbol{x} \in \mathbb{R}^3$ is projected onto the image plane via

$$\widetilde{\boldsymbol{x}'} = P\widetilde{\boldsymbol{x}}.$$

$\widetilde{\boldsymbol{x}'}$ can then be converted to a point of the Euclidean space as usual. The camera matrix P consists of two distinct components. While the calibration matrix $K \in \mathbb{R}^{3\times3}$ describes the camera's intrinsics, the extrinsics are described via a rigid transformation matrix $E \in \mathbb{R}^{3\times4}$. E expresses the transformation from the world coordinate system to the three-dimensional camera coordinate system. It consists of the world's origin $\boldsymbol{t} \in \mathbb{R}^3$ in camera coordinates and the rotation $R \in \mathbb{R}^{3\times3}$. Knowing the camera's position $\boldsymbol{c} \in \mathbb{R}^3$ in world coordinates, $\boldsymbol{t}$ is calculated as $\boldsymbol{t} = -R\boldsymbol{c}$. The camera matrix P is therefore determined as

$$P \coloneqq KE$$

with $E \coloneqq [R|\boldsymbol{t}]$.

To check whether the point $\boldsymbol{x} \in \mathbb{R}^3$ lies in front or behind a camera view given by its camera matrix, the condition

$$(P\widetilde{\boldsymbol{x}})_3 > 0 \tag{2.2}$$

may be used [11]. $(P\widetilde{\boldsymbol{x}})_3$, i.e. the projective coordinate w of $\widetilde{\boldsymbol{x}'}$ is positive if $\boldsymbol{x}$ lies in front of the camera and negative if $\boldsymbol{x}$ lies behind the camera. Further, it is equal to zero if $\boldsymbol{x}$ lies in the so-called principle plane of the camera. This plane contains the camera's position $\boldsymbol{c}$ and its normal vector is given by the view direction. In this case, there exists no unique point of the image plane on which $\boldsymbol{x}$ projects onto. Note that these conditions can only be used if the left 3×3 submatrix of P, i.e. KR, has a positive determinant [11]. Otherwise the conditions are reversed. Throughout this thesis and without loss of generality, the determinant of the left 3×3 submatrix of a camera matrix is always considered positive. If this submatrix has a negative determinant, then the camera

matrix P can simply be multiplied by minus one. This does not, however, influence the projection itself as all coordinates of $\widetilde{\boldsymbol{x}}' = P\widetilde{\boldsymbol{x}}$, including the projective coordinate w, are negated.

For some algorithms, an inverse projection, and therefore an inverse camera matrix, is needed. For this, P must be inversible and extended to a 4×4 matrix. This is achieved by adding non-diagonal elements of value zero and a diagonal element of value one, i.e.

$$\tilde{P} \coloneqq \tilde{K}\tilde{E} = \begin{bmatrix} K & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix},$$

whereby $\tilde{P}$ denotes the extended camera matrix. It is applied in a similar manner as the conventual camera matrix P. Let $\widetilde{\boldsymbol{y}} \in \mathbb{P}^3$ be a point of the projection space. $\widetilde{\boldsymbol{y}}' = \tilde{P}\widetilde{\boldsymbol{y}}$ is the projection of $\widetilde{\boldsymbol{y}}$ onto the image plane. However,

$$\widetilde{\boldsymbol{y}}' = (\tilde{y}_1', \tilde{y}_2', \tilde{y}_3', \tilde{y}_4')^T \tag{2.3}$$

is not a point of the two-dimensional projective space because it is composed of four elements instead of three. The existence of $\tilde{y}_4'$ is a direct result of the extension of P to $\tilde{P}$. That being said, in this case it can simply be dropped. The resulting point $(\tilde{y}_1', \tilde{y}_2', \tilde{y}_3')$ can be handled as a point of the projection space, meaning that it corresponds to

$$\boldsymbol{y}' = \left(\frac{\tilde{y}_1'}{\tilde{y}_3'}, \frac{\tilde{y}_2'}{\tilde{y}_3'} \right)^T \in \mathbb{R}^2.$$

The inverse camera matrix $\tilde{P}^{-1}$ can be used to determine all points $\boldsymbol{x} \in \mathbb{R}^3$ that project onto a given image plane point $\boldsymbol{x}' \in \mathbb{R}^2$. For this,

$$\widetilde{\boldsymbol{x}} = \tilde{P}^{-1}\widetilde{\boldsymbol{x}}' \tag{2.4}$$

may be used. Again, $\widetilde{\boldsymbol{x}}' = (x_1', x_2', 1, d)^T$ is composed of four elements. However, while the first three elements are known, d is unknown. This is a result of the information loss deriving from the reduction of the dimension by one. More specifically, it is the dropped y_4' from (2.3) divided by y_3'. Using any $d \neq 0$ for (2.4) leads to a valid $\boldsymbol{x} \in \mathbb{R}^3$, that projects onto the given $\boldsymbol{x}' \in \mathbb{R}^2$.

2.3 APOLI

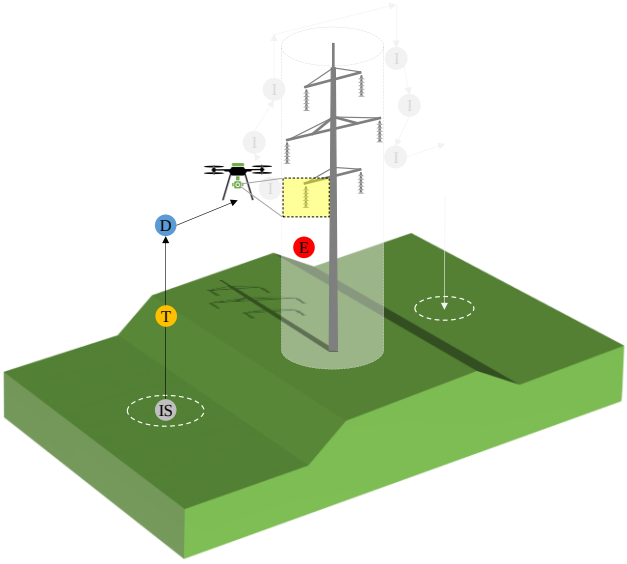


Figure 2.1: States of Insulator Inspection of APOLI [2].

The Automated Power Line Inspection (APOLI) [1]–[4] project has been briefly introduced in Chapter 1.1. This chapter, however, elaborates the project in greater detail. In particular, a UAS is developed which autonomously inspects power lines and detects damages of them such as

- electrical short damages,
- cracks and glass breakages as well as
- transmission line damages.

While these tasks can also be performed without the involvement of a UAV, objects such as insulators of power poles are hard to reach by humans in respect to safety and cost. Therefore, an appropriate UAS can ease the inspection process drastically.

2.3.1 Hardware Setup

The UAS of APOLI runs on a single-board computer, an ODROID-XU4 [12], which is directly mounted on the UAV. The ODROID computer features a Samsung Exynos 5422 Octa-Core processor that consists of four Cortex®-A15 2.1GHz and four Cortex®-A7 1.4GHz processor cores. Additionally, the XU4 possesses 2GB of main memory. To control the UAV, a dedicated flight controller (FLC), which is either a 3DR Pixhawk 1 [13] or a Pixhawk 4 [14], is being used. It handles given flight commands and provides flight data in an easy-to-use format. The firmware flashed on the FLC is ArduCopter 4.0 [15].

For the navigation, a FLIR Blackfly S USB3 BFS-U3-31S4C-C [16] camera is being used. It captures RGB images at 3.2 megapixels with a framerate of up to 55Hz. To

retrieve the image stream, a USB 3 interface is provided by the camera. In addition to the navigation camera, the UAV also features an inspection camera that captures images at a higher resolution. More specifically, this camera is a FLIR Blackfly S GigE BFS-PGE-120S4C-CS [17] which captures RGB images at 12 megapixels. However, it features a lower framerate of only up to 8.5Hz. For image retrieval, a Gigabit Ethernet port is available. To achieve stable and UAV movement independent image streams of both cameras, they are mounted on a gimbal. The HD Air Studio InfinityMR-S [18] gimbal is being used for this.

2.3.2 Software Setup

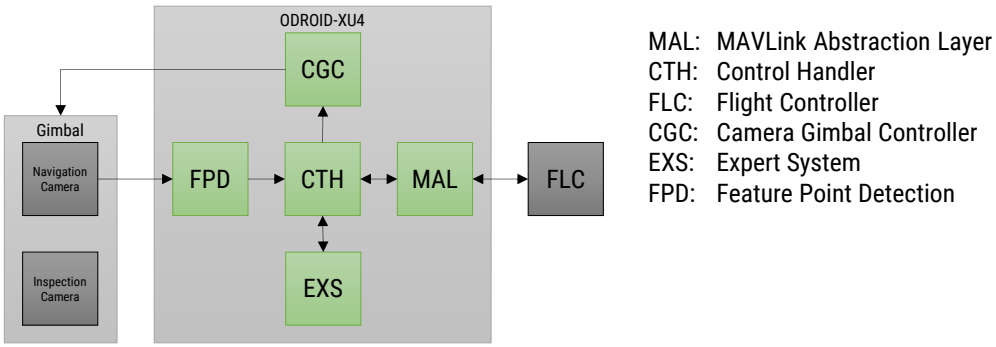


Figure 2.2: Snippet of the Structure of the APOLI Project.

Figure 2.2 shows a snippet of the APOLI project's software structure. As can be seen, it is composed of multiple individual software components. The EXS is the central decision-making component. It issues flight commands and aligns the gimbal. These commands are sent to the CTH, which forwards them to their corresponding component. The MAL sends flight commands to the FLC. This component also receives the UAV's flight data. The CGC simply sends gimbal align commands to the gimbal and the FPD detects features of objects in the image stream of the navigation camera.

2.4 UAV Flight Data

In this chapter, some of the flight data provided by a UAV is elaborated. This data is necessary for building accurate camera matrices for the cameras mounted on the UAV. In particular, Chapter 2.4.1 describes the data obtained from a Global Navigation Satellite System [19] sensor while Chapter 2.4.2 explains the principle axes of a UAV which are used to determine the orientation of the UAV.

2.4.1 Global Navigation Satellite System

A Global Navigation Satellite System, also referred to as a GNSS, is a system consisting of multiple satellites which can be used to accurately determine the position

of a GNSS sensor on the earth. Some examples [19] for existing Global Navigation Satellite Systems are the Global Positioning System (GPS), developed by the United States of America, the GLONASS, which also stands for Global Navigation Satellite System and is operated by the Russian Federation, as well as Galileo, which is being maintained by the European Union.

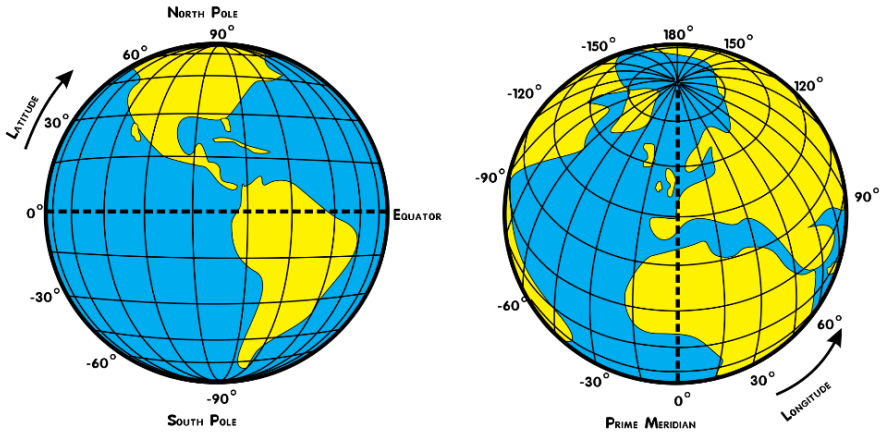


Figure 2.3: Geographic Latitude and Longitude of the Earth [20].

In this thesis, the data provided by a GNSS sensor mounted on a UAV is considered to be the latitude, the longitude and the altitude of the UAV in respect to the earth. The latitude is a coordinate that specified the north-south position. It ranges from -90°, being the geographic south pole, to 90° which represents the geographic north pole. Analogously, the longitude represents the east-west position. If it is 0°, the UAV lies on the same line of longitude as Greenwich. This line of longitude is also called the prime meridian. Going westwards yields negative longitude values down to -180° while the positions to the east of the prime meridian have positive longitude values of up to 180°. The line of longitude which may possess a longitude of 180° or -180° is called the antimeridian and traverses the easternmost part of the Russian Federation. Lastly, the altitude simply represents the elevation of the UAV above the mean see level in meters.

2.4.2 Principle Axes of a UAV

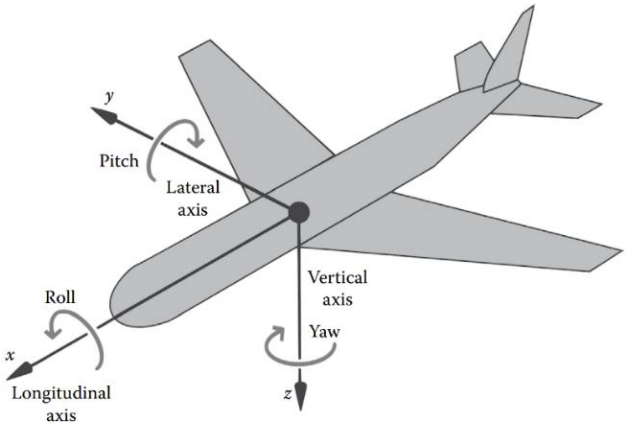


Figure 2.4: Principle Axes of an Aircraft [21].

To express the orientation of any UAV, the principle axes used for aircrafts may be used [21], [22]. These axes are the roll axis, the pitch axis and the yaw axis. They build an orthonormal coordinate system as illustrated in Figure 2.4. Furthermore, a reference system must be defined. In the north-east-down (NED) reference system the origin lies at the center of the aircraft, the roll axis points in the north direction, the pitch axis shows eastwards and the yaw axis is perpendicular to both axis and is directed downwards pointing to the center of the earth. As a result, the NED reference system is right-handed. The principle axis of an aircraft itself are, as it may be oriented arbitrarily, defined differently. The roll axis of an aircraft shows in the heading direction while the pitch axis is directed “to the right”. Lastly, the yaw axis is again perpendicular to both axis and points downwards with respect to the aircraft.

To determine the rotation from the NED reference system to the UAV system, three rotation angles are necessary. These parameters determine the rotation angles around their corresponding axis. In the following, $\phi \in \mathbb{R}$ is the roll rotation, $\theta \in \mathbb{R}$ denotes the pitch rotation and $\psi \in \mathbb{R}$ represents the yaw rotation. Any point $\boldsymbol{x_{NED}}$ of the NED system is first rotated around the z-axis, i.e. the yaw axis, then around the y-axis and lastly around the x-axis. Therefore, the complete rotation matrix $R \in \mathbb{R}^{3\times3}$ is derived via

$$R_{xyz} \coloneqq R_{x,\phi} R_{y,\theta} R_{z,\psi}$$
$$= \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta) \\ \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix},$$

whereby $R_{x,\phi}, R_{y,\theta}, R_{z,\psi} \in \mathbb{R}^{3\times3}$ denote the rotation matrices to rotate around the corresponding axis. More specifically, rotating any point $\boldsymbol{x_{NED}}$ of the NED system using R results in the corresponding point $\boldsymbol{x_{UAV}}$ of the UAV system, i.e.

$$\boldsymbol{x_{UAV}} = R\boldsymbol{x_{NED}}.$$

To rotate from $\boldsymbol{x_{UAV}}$ to $\boldsymbol{x_{NED}}$ the inverse rotation matrix of R may be used. This yields

$$\boldsymbol{x_{NED}} = R^T \boldsymbol{x_{UAV}}.$$

2.5 Monocular RGB Cameras

As written in Chapter 1.2, the Real-Time Object Localizer shall use, among others, the image data provided by a monocular RGB camera system [5], [23]. In this context, RGB refers to the three base colors red, green and yellow of the additive RGB color space. Consequently, RGB cameras are camera systems which capture RGB images, i.e. images consisting of three channels each corresponding to one of these base colors. Additionally, a monocular camera system is a camera system consisting only of a single camera.

Monocular camera systems stand in strong contrast to multi-camera systems [24], i.e. camera setups consisting of multiple cameras. While the problem of object localization becomes easier to solve if a multi-camera system is used, such camera systems may also lead to practicality problems. Firstly, a multi-camera system consumes more space. While this may not be a problem if these systems are used on the ground, UAVs only have a very limited amount of space available. Furthermore, the additional cameras lead to supplemental weight and, therefore, to additional power consumption of the UAV. Last but not least, a multi-view camera system has a significantly higher cost than a monocular camera system.

Besides RGB cameras, one can also use camera systems which provide additional information in the images captured, e.g. RGB-D cameras [25]. Here, the added D stands for depth and its channel contains the distance between the camera and the three-dimensional point captured by the corresponding pixel. This additional distance information can be used for further object localization algorithms or to improve existing algorithms. However, the range of the distance is, for most RGB-D cameras, rather limited. For these reasons, and also because the APOLI's UAV already possesses two monocular RGB camera systems, i.e. the navigation and the inspection camera, the object detection input of the RTL is generated via a monocular RGB camera system. More specifically, the navigation camera is used for this.

3 State-of-the-Art of Object Localization

In this chapter, some state-of-the-art object localization algorithms are elaborated. Approaches, which may be used in a UAS that possesses a monocular RGB camera, provides GNSS data and also features some form of object detection, are further explained in greater detail, i.e. algorithms which may be used in the APOLI project.

Chapter 3.1 elaborates a single-view object localization approach. This means that a single camera view, i.e. a monocular camera system, is sufficient to localize objects in the 3D space. Subsequently, Chapter 3.2.1 to 3.2.2 introduce so-called stereo-view approaches. These algorithms use input data captured from two different viewpoints. Normally, a monocular camera does not provide such information. However, as the camera is mounted on a moving vehicle, this movement can be used to capture images from different viewpoints, effectively creating a multi-camera system. That being said, this requires that the pose of the objects that are to be localized must not change. The following chapters 3.2.3 to 3.4 present multi-view algorithms. These localization approaches operate on any number of views greater than a specific threshold. Finally, a brief summary of all discussed object localization approaches is given in chapter 3.5.

3.1 Depth Estimation using the Vertical Position of the Bounding Rectangle

Figure 3.1: Depth Estimation using the Vertical Position of the Bounding Rectangle. The yellow Line is located at the Vertical Position of the Vanishing Point [23].

In [23] Joglekar et al. proposed a single-view approach to estimate the distance of a detected object from a monocular camera. If the distance of an object from the camera is known, its location in the 3D environment can easily be approximated. In their paper, Joglekar et al. use the image's geometry to directly estimate the depth. More specifically, their algorithm is being provided with the camera's calibration as well as a

bounding rectangle around an object in the image. It further assumes that the object is placed on a flat ground plane, e.g. a road, and that the camera is mounted so that the optical axis is parallel to the ground plane. The algorithm then approximates the depth of the object by abusing the perspective properties of the pinhole camera model. The higher the distance of an object placed on a flat ground from the camera, the more does the lower boundary of the bounding rectangle converge to the vertical position of the vanishing point.

This approach, however, is not feasible for most UASs. Firstly, the algorithm assumes a flat ground plane to which the UAV has a constant distance to. This assumption cannot be made for most UASs. Even if a flat ground plane is present, the UAV then cannot ascend or descend in any way and, depending on the depth estimation tolerance, must precisely keep its altitude. Moreover, the depth estimation algorithm proposed by Joglekar et al. requires a fixed pitch angle of the camera used. While this can be achieved using a gimbal, it limits the usability of the camera in the corresponding UAS.

3.2 Triangulation of a Single Point

Triangulation [26] is the reconstruction of a point $\boldsymbol{x} \in \mathbb{R}^3$ which has been captured from n different viewpoints via the image points $\boldsymbol{u_1}, \dots, \boldsymbol{u_n} \in \mathbb{R}^2$. These viewpoints are denoted using their corresponding camera matrices $P_1 = K_1[R_1|\boldsymbol{t_1}], \dots, P_n = K_n[R_n|\boldsymbol{t_n}] \in \mathbb{R}^{3\times4}$. Triangulation then aims to find an $\boldsymbol{x'} \in \mathbb{R}^3$, given $\boldsymbol{u_1}, \dots, \boldsymbol{u_n}$ and $P_1, \dots, P_n$, which best explains $\boldsymbol{x}$. The following subchapters elaborate five different triangulation algorithms. More specifically, these algorithms are:

- Noiseless Triangulation (Chapter 3.2.1),
- Midpoint Triangulation (Chapter 3.2.2),
- Generalized Midpoint Triangulation (Chapter 3.2.3),
- L_2 Triangulation (Chapter 3.2.4) and
- L_∞ Triangulation (Chapter 3.2.5).

The Noiseless Triangulation and the Midpoint Triangulation are two stereo-view triangulation algorithms meaning that only the data provided by two camera views are being used. Subsequently, the Generalized Midpoint, the L_2 and the L_∞ Triangulation formulate triangulation approaches which utilize any number of views greater than one. While the Noiseless Triangulation assumes the absence of noise, all other triangulation approaches can also be applied on noise afflicted data.

3.2.1 Noiseless Triangulation

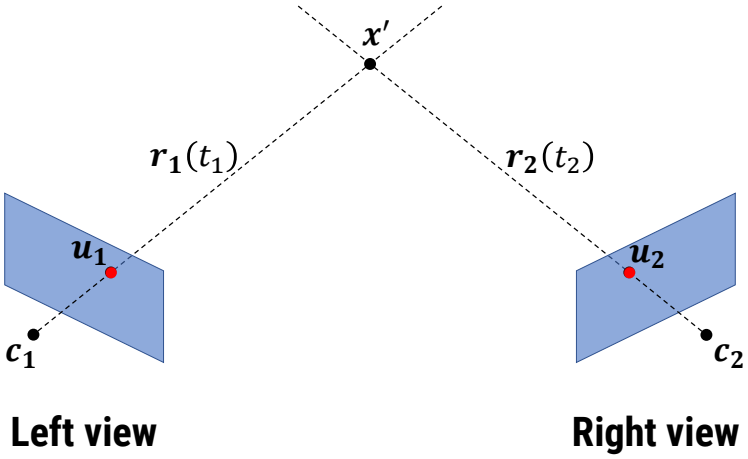


Figure 3.2: Noiseless Triangulation of a Single Point.

In a noiseless environment, $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are the precise projections of $\boldsymbol{x}$ in their corresponding image plane. Using their camera matrices, they can be expressed as

$$\widetilde{\boldsymbol{u_1}} = P_1\widetilde{\boldsymbol{x}}$$

and

$$\widetilde{\boldsymbol{u_2}} = P_2\widetilde{\boldsymbol{x}}.$$

The problem of finding the corresponding $\boldsymbol{x}'$ can be solved in a trivial manner as $\boldsymbol{x}$ is the intersection point of two rays. While the first ray starts at $\boldsymbol{c_1} \coloneqq -R^T\boldsymbol{t_1} \in \mathbb{R}^3$ and traverses $\boldsymbol{v_1} \in \mathbb{R}^3$, where

$$\widetilde{\boldsymbol{v_1}} = \widetilde{P_1}^{-1}\widetilde{\boldsymbol{u_1}}$$

for any $d > 0$, the second ray begins at $\boldsymbol{c_2} \in \mathbb{R}^3$ and passes through $\boldsymbol{v_2} \in \mathbb{R}^3$ analogously. These rays can therefore be described as

$$\boldsymbol{r_1}(t_1) \coloneqq \boldsymbol{c_1} + t_1(\boldsymbol{v_1} - \boldsymbol{c_1}),$$
$$\boldsymbol{r_2}(t_2) \coloneqq \boldsymbol{c_2} + t_2(\boldsymbol{v_2} - \boldsymbol{c_2})$$

with $t_1, t_2 \geq 0$. Furthermore, because $\boldsymbol{x} \in \mathbb{R}^3$ lies on both of these rays, the system of linear equations

$$\boldsymbol{c_1} + t_1(\boldsymbol{v_1} - \boldsymbol{c_1}) = \boldsymbol{c_2} + t_2(\boldsymbol{v_2} - \boldsymbol{c_2})$$

with t_1 and t_2 as unknowns, is obtained. Solving it yields the solution for t_1 and t_2 and, therefore, also for the triangulated $\boldsymbol{x}'$ as

$$\boldsymbol{x}' \coloneqq \boldsymbol{c_1} + \frac{\|(\boldsymbol{c_2} - \boldsymbol{c_1}) \times (\boldsymbol{v_2} - \boldsymbol{c_2})\|}{\|(\boldsymbol{v_1} - \boldsymbol{c_1}) \times (\boldsymbol{v_2} - \boldsymbol{c_2})\|}(\boldsymbol{v_1} - \boldsymbol{c_1}) \tag{3.1}$$

or equally

$$\boldsymbol{x}' \coloneqq \boldsymbol{c_2} + \frac{\|(\boldsymbol{c_1} - \boldsymbol{c_2}) \times (\boldsymbol{v_1} - \boldsymbol{c_1})\|}{\|(\boldsymbol{v_1} - \boldsymbol{c_1}) \times (\boldsymbol{v_2} - \boldsymbol{c_2})\|}(\boldsymbol{v_2} - \boldsymbol{c_2}), \tag{3.2}$$

whereby

$$\|\boldsymbol{a}\| \coloneqq \sqrt{\sum_j a_j^2}$$

denotes the L_2 norm, also Euclidean norm, of a vector $\boldsymbol{a}$ and $\boldsymbol{b} \times \boldsymbol{c}$ stands for the cross-product between two vectors $\boldsymbol{b}$ and $\boldsymbol{c}$. It should be noted, however, that if $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are parallel, there is no unique solution for $\boldsymbol{x}'$. In this case, the denominators are equal to zero, as the cross-product of two parallel vectors results in the zero vector. Because both rays have fixed initial points $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ and also traverse the same $\boldsymbol{x}$, there exist only two cases in which they are parallel. If $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are parallel and if both of them contain the other ray's initial point, then $\boldsymbol{x}$ lies on the line segment $\boldsymbol{l}$ connecting $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$. In that event, both rays completely contain $\boldsymbol{l}$. Therefore, every point $\boldsymbol{x}'$ that lies on $\boldsymbol{l}$ is a valid reprojection of $\boldsymbol{x}$. If both rays $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are parallel but it does not yield true that both of them contain each other's initial point, a separate scenario is achieved. In this case, only one of the rays contain the other ray's initial point. Without loss of generality, let that ray be $\boldsymbol{r_1}$. Furthermore, $\boldsymbol{r_1}$ does not only contain the initial point $\boldsymbol{c_2}$ of $\boldsymbol{r_2}$ but also, because both rays traverse the same $\boldsymbol{x}$, $\boldsymbol{r_2}$ as a whole. Again, any point $\boldsymbol{x}'$ of $\boldsymbol{r_2}$ is a valid reprojection of $\boldsymbol{x}$. As stated before, $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are parallel if, and only if,

$$\|(\boldsymbol{v_1} - \boldsymbol{c_1}) \times (\boldsymbol{v_2} - \boldsymbol{c_2})\| = 0 \tag{3.3}$$

holds true. Therefore, a test for this case can easily be performed.

This triangulation assumes the absence of noise. If any of the values $\boldsymbol{u_1}, \boldsymbol{u_2}, P_1$ or P_2 are affected by noise, the rays are not guaranteed to intersect anymore. In this case, (3.1) and (3.2) cannot be used. However, the data provided by any UAS must be assumed to be afflicted by some noise. This includes both the points $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ as well as the camera matrices P_1 and P_2. In particular, the rigid transformation matrices E_1 and E_2 of the camera matrices are obtained using, among other information, GNSS sensor data which is by no means noiseless. Even in the absence of noise, for real images the coordinates of $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are integers. Therefore, the digitalization further amplifies this issue. For these reasons, the Noiseless Triangulation is not feasible for application in systems dealing with noise afflicted data. In particular, this triangulation cannot be used in UASs.

3.2.2 *Midpoint Triangulation*

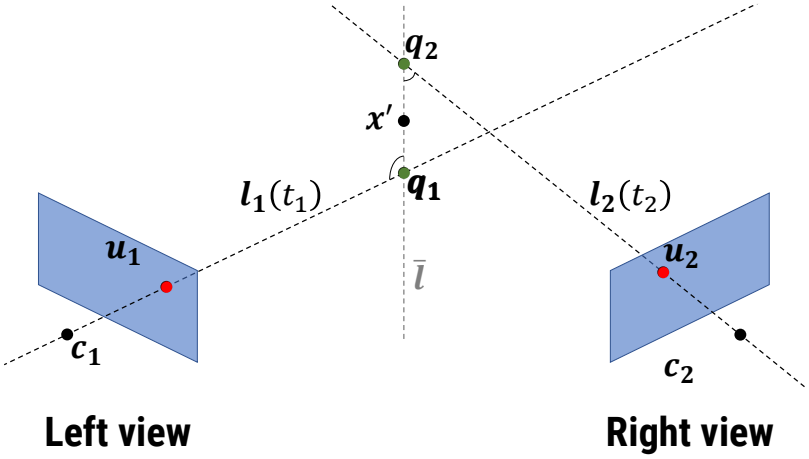


Figure 3.3: Midpoint Triangulation of a Single Point.

P. A. Beardsley et al. briefly describe in [27] an approach able to deal with the noise issue, the so-called Midpoint Triangulation. Here $\boldsymbol{x}'$ is not determined as the point of intersection between the rays $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ but rather as the midpoint of the common perpendicular to them. Therefore, $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ do not have to intersect anymore. The Midpoint Triangulation further extends the rays to lines. This is because if $\boldsymbol{x}$ lies close to the camera position $\boldsymbol{c_1}$, noise can easily translate the captured image point $\boldsymbol{u_2}$ of the second view in a way that the resulting ray $\boldsymbol{r_2}$ lies completely lies "behind" the first camera and vise-versa. In such cases, the rays $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ do not have a common perpendicular. Therefore, using lines instead of rays is more feasible. To prevent confusions, the lines, which are extensions of the rays $\boldsymbol{r_1}(t_2)$ and $\boldsymbol{r_2}(t_2)$, are referred to as $\boldsymbol{l_1}(t_1)$ and $\boldsymbol{l_2}(t_2)$ respectively. Furthermore, it should be noted that $\boldsymbol{x}'$ must be uniquely identifiable meaning that there has to be exactly one point $\boldsymbol{x}' \in \mathbb{R}^3$ which has the same distance to both lines $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$. This uniqueness is satisfied if, and only if, $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$ are nonparallel. Again, this occurs exactly if (3.3) yields false.

To determine $\boldsymbol{x}'$, the lines $\boldsymbol{l_1}(t_1)$ and $\boldsymbol{l_2}(t_2)$ are required. As they are the extensions of $\boldsymbol{r_1}(t_1)$ and $\boldsymbol{r_2}(t_2)$ respectively, they can be obtained in an almost identical way, i.e.

$$\boldsymbol{l_1}(t_1) \coloneqq \boldsymbol{c_1} + t_1(\boldsymbol{v_1} - \boldsymbol{c_1}), \tag{3.4}$$

$$\boldsymbol{l_2}(t_2) \coloneqq \boldsymbol{c_2} + t_2(\boldsymbol{v_2} - \boldsymbol{c_2}), \tag{3.5}$$

whereby $t_1, t_2 \in \mathbb{R}$. The common perpendicular $\bar{\boldsymbol{l}}$ to $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$ is, except if $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$ are parallel which is to be avoided, uniquely determined. Further, $\bar{\boldsymbol{l}}$ traverses through the point of each line, which has minimal distance to the other line. In the following, these points are referred to as $\boldsymbol{q_1} \in \mathbb{R}^3$, which lies on $\boldsymbol{l_1}$, and $\boldsymbol{q_2} \in \mathbb{R}^3$, which is located on $\boldsymbol{l_2}$. Additionally, let $\boldsymbol{v}(t_1, t_2)$ be a vector which translates from $\boldsymbol{l_2}(t_2)$ to $\boldsymbol{l_1}(t_1)$. Its equation is therefore determined as

$$\boldsymbol{v}(t_1, t_2) \coloneqq \boldsymbol{l_1}(t_1) - \boldsymbol{l_2}(t_2).$$

It is perpendicular to both lines $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$ exactly when it translates from $\boldsymbol{q_2}$ to $\boldsymbol{q_1}$. Therefore, t_1 and t_2 can be obtained by solving a system of linear equations given as

$$\langle \boldsymbol{v}(t_1, t_2), \boldsymbol{v_1} - \boldsymbol{c_1} \rangle = 0,$$
$$\langle \boldsymbol{v}(t_1, t_2), \boldsymbol{v_2} - \boldsymbol{c_2} \rangle = 0,$$

whereby $\langle *,* \rangle$ denotes the dot product. Solving this system, the parameters $t_1$and t_2 are determined as:

$$t_1 = \frac{\|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2 \langle \boldsymbol{c_1} - \boldsymbol{c_2}\,, \boldsymbol{v_1} - \boldsymbol{c_1} \rangle - \langle \boldsymbol{c_1} - \boldsymbol{c_2}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle \langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle}{2\langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2},$$
$$t_2 = \frac{\|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \langle \boldsymbol{c_2} - \boldsymbol{c_1}\,, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \langle \boldsymbol{c_2} - \boldsymbol{c_1}, \boldsymbol{v_1} - \boldsymbol{c_1} \rangle \langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle}{2\langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2}.$$

Utilizing (3.4) and (3.5), t_1 and t_2 can then be used to calculate the point of each line that intersects with $\bar{\boldsymbol{l}}$, i.e. $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$. They are expressed through

$$\boldsymbol{q_1} \coloneqq \boldsymbol{c_1} + \frac{\|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2 \langle \boldsymbol{c_1} - \boldsymbol{c_2}\,, \boldsymbol{v_1} - \boldsymbol{c_1} \rangle - \langle \boldsymbol{c_1} - \boldsymbol{c_2}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle \langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle}{2\langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2} (\boldsymbol{v_1} - \boldsymbol{c_1})$$

and

$$\boldsymbol{q_2} \coloneqq \boldsymbol{c_2} + \frac{\|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \langle \boldsymbol{c_2} - \boldsymbol{c_1}\,, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \langle \boldsymbol{c_2} - \boldsymbol{c_1}, \boldsymbol{v_1} - \boldsymbol{c_1} \rangle \langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle}{2\langle \boldsymbol{v_1} - \boldsymbol{c_1}, \boldsymbol{v_2} - \boldsymbol{c_2} \rangle - \|\boldsymbol{v_1} - \boldsymbol{c_1}\|^2 \|\boldsymbol{v_2} - \boldsymbol{c_2}\|^2} (\boldsymbol{v_2} - \boldsymbol{c_2}).$$

As $\boldsymbol{x'}$ corresponds to the midpoint of the line segment between $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$, it can now be obtained via

$$\boldsymbol{x'} \coloneqq \frac{1}{2}(\boldsymbol{q_1} + \boldsymbol{q_2}).$$

In principle, the Midpoint Triangulation algorithm can be used by any UAS which implements some form of object detection algorithm. All necessary triangulation input data can easily be obtained. While the image points $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are simply the center of the object in both image frames, the camera matrices P_1 and P_2 can be constructed using the UAV's flight data. Furthermore, the Midpoint Triangulation is able to deal with noise afflicted on the cameras' positions $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ particularly well as the midpoint approach averages this noise. That being said, R. I. Hartley and P. Sturm state in [26] that when noise is afflicted on projective input data, i.e. the image points $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ as well as the cameras' orientations, the Midpoint Triangulation approach does not yield good results. This is because this triangulation reconstructs a Euclidean space from the camera views and, therefore, lacks to recognize the projective nature of pinhole cameras. Small noise in the values of $\boldsymbol{u_l}$ can lead to great errors in the calculation of $\boldsymbol{x'}$. This is especially true if $\boldsymbol{x}$ is far away from the corresponding camera position $\boldsymbol{c_l}$. Therefore, the usability of the Midpoint Triangulation greatly depends on the kind of noise which may be afflicted on the input data.

3.2.3 Generalized Midpoint Triangulation

While the previous two chapters elaborated triangulation approaches that use two views only, i.e. stereo-view approaches, the triangulation algorithm presented in this chapter utilizes any number of views $n \geq 2$. In fact, this algorithm, briefly described by S. Ramalingam et al. in [28], is a generalization of the Midpoint Triangulation elaborated in Chapter 3.2.2. For this reason, it will be referred to as the Generalized Midpoint Triangulation throughout the thesis.

Using more than two views may seem redundant at first glance as two views are, in general, sufficient to triangulate a $\boldsymbol{x}'$. However, the additional information can be used to smoothen out noise and, therefore, improve the triangulation result $\boldsymbol{x}'$. Again, for each view the input consists of an image point $\boldsymbol{u_l}$ as well as the corresponding camera matrix P_l. Given this input, the camera positions $\boldsymbol{c_l}$ and an arbitrary line point $\boldsymbol{v_l}$ can be constructed as discussed before. Given the n lines $\boldsymbol{l_1}, \dots, \boldsymbol{l_n}$ as

$$\boldsymbol{l_l}(t_l) \coloneqq \boldsymbol{c_l} + t\boldsymbol{d_l}$$

with $t_l \in \mathbb{R}$ and

$$\boldsymbol{d_l} \coloneqq \frac{1}{\|\boldsymbol{v}_l - \boldsymbol{c}_l\|}(\boldsymbol{v}_l - \boldsymbol{c}_l), \tag{3.6}$$

the Generalized Midpoint Triangulation determines the $\boldsymbol{x}'$ which is closest on average to all lines, i.e.

$$\boldsymbol{x}' \coloneqq \underset{\boldsymbol{x}'}{\operatorname{argmin}} \sum_{l=1}^{n} \mathrm{d}(\boldsymbol{x}', \boldsymbol{l_l})^2 = \underset{\boldsymbol{x}'}{\operatorname{argmin}} \min_{\boldsymbol{t}} \sum_{l=1}^{n} \|\boldsymbol{x}' - (\boldsymbol{c_l} + t_l\boldsymbol{d_l})\|^2. \tag{3.7}$$

Here, $\mathrm{d}(*,*)$ denotes the Euclidean distance between a point and a line. Because (3.7) represents a linear least squares problem, it can be solved using the pseudo-inverse which yields

$$\begin{pmatrix} \boldsymbol{x}' \\ t_1 \\ \vdots \\ t_n \end{pmatrix} = M^{-1} \begin{bmatrix} I & \cdots & I \\ -\boldsymbol{d_1}^T & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -\boldsymbol{d_n}^T \end{bmatrix} \begin{pmatrix} \boldsymbol{c_1} \\ \vdots \\ \boldsymbol{c_n} \end{pmatrix} \tag{3.8}$$

with $I \in \mathbb{R}^{3\times3}$ as the identity matrix and

$$M \coloneqq \begin{bmatrix} nI & -\boldsymbol{d_1} & \cdots & -\boldsymbol{d_n} \\ -\boldsymbol{d_1}^T & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{d_n}^T & 0 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{(3+n)\times(3+n)}.$$

Because of the sparse structure of M, its inversion can be performed efficiently. For this, S. Ramalingam et al. even derive a closed-form solution as

$$M^{-1} = \begin{bmatrix} \frac{1}{n}(I + DD^TA) & AD \\ D^TA & I + D^TAD \end{bmatrix}, \tag{3.9}$$

whereby $D \coloneqq [\boldsymbol{d_1}| \dots |\boldsymbol{d_n}] \in \mathbb{R}^{3\times n}$ and $A \coloneqq (nI - DD^T)^{-1} \in \mathbb{R}^{3\times 3}$. Applying (3.9) in (3.8) yields a closed-form solution for $\boldsymbol{x}'$ via

$$\boldsymbol{x}' \coloneqq \frac{1}{n}(I + DD^T A)\sum_{i=1}^{n} \boldsymbol{c_i} - A\sum_{i=1}^{n}\langle \boldsymbol{c_i}, \boldsymbol{d_i}\rangle \boldsymbol{d_i}. \tag{3.10}$$

It should be noted that (3.10) can only be used to determine $\boldsymbol{x}'$ if $\boldsymbol{x}'$ is uniquely determined via (3.7), i.e. if there is exactly one point which is closest on average to all lines. If there are multiple points $\boldsymbol{x}'$ which minimize (3.7) then A cannot be determined because the matrix $nI - DD^T$ is singular and, therefore, cannot be inverted. For the stereo-view case, A is singular exactly if both lines are parallel, i.e. if

$$\|\boldsymbol{d_1} \times \boldsymbol{d_2}\| = 0$$

yields true.

In comparison to the Midpoint Triangulation, the Generalized Midpoint Triangulation reduces the effect of noise in the input data $\boldsymbol{u_1}, \dots, \boldsymbol{u_n}$ and $P_1, \dots, P_n$ on the triangulation result $\boldsymbol{x}'$. This is because it utilizes any number of views greater than one and, therefore, is able to smoothen out such noise. Furthermore, such like the Midpoint Triangulation itself, it determines $\boldsymbol{x}'$ through a closed-form solution. This $\boldsymbol{x}'$ can be calculated efficiently as only basic algebraic operations are being used. However, the Generalized Midpoint Triangulation still does not recognize the projective properties of pinhole cameras. Instead, it operates in the Euclidean space using lines which have been constructed from the projective space. Therefore, it suffers from the same issue as the Midpoint Triangulation. While it deals with noise in the cameras' positions $\boldsymbol{c_l}$ in an optimal way, assuming a Gaussian noise model, noise afflicted on the image points $\boldsymbol{u_l}$ and the cameras' orientations is not dealt well with. That being said, the Generalized Midpoint Triangulation is, in general, more suitable for application in noise afflicted environments and, therefore, also in UASs as it deals with noise more efficiently. In particular, it may be used in the APOLI project.

3.2.4 L_2 Triangulation

In the previous chapters, triangulation algorithms were presented which operate in the Euclidean space. This, however, is not a feasible triangulation approach if only the image points $\boldsymbol{u_l}$ of images captured using pinhole cameras are afflicted by noise as the projective properties of such cameras are not considered. Instead of working in the three-dimensional Euclidean space, the L_2 Triangulation [5], [29], [30] described in this chapter operates on the two-dimensional Euclidean space of the image planes. Furthermore, it is a multi-view triangulation algorithm.

The L_2 Triangulation assumes noiseless camera matrices P_l and image points $\boldsymbol{u_l}$ which are subjects to a Gaussian noise model. This means that the image points $\boldsymbol{u_l}$ are likely to be in the correct area and that their ground-truth correspondences are of small distance to them. The goal of the L_2 Triangulation is to find an $\boldsymbol{x'}$ that has the lowest reprojection error, i.e. finding an $\boldsymbol{x'}$ which minimizes the least squares error

$$\boldsymbol{x'} \coloneqq \underset{\boldsymbol{x'}}{\operatorname{argmin}} \sum_{l=1}^{n} \left|\left|\boldsymbol{u_l} - \boldsymbol{u_l'}\right|\right|^2 \tag{3.11}$$

with

$$\widetilde{\boldsymbol{u_l'}} = P_l \widetilde{\boldsymbol{x'}}.$$

It is worth mentioning that (3.11) is effectively minimizing the L_2 norm, hence the name of the triangulation, of a vector $\boldsymbol{f}(\boldsymbol{x'}) \coloneqq (\|\boldsymbol{u_1} - \boldsymbol{u_1'}\|, \dots, \|\boldsymbol{u_n} - \boldsymbol{u_n'}\|)^T \in \mathbb{R}^n$, i.e. (3.11) can be reformulated as

$$\boldsymbol{x'} \coloneqq \underset{\boldsymbol{x'}}{\operatorname{argmin}} \|\boldsymbol{f}(\boldsymbol{x'})\| = \underset{\boldsymbol{x'}}{\operatorname{argmin}} \left\| \begin{pmatrix} \|\boldsymbol{u_1} - \boldsymbol{u_1'}\| \\ \vdots \\ \|\boldsymbol{u_n} - \boldsymbol{u_n'}\| \end{pmatrix} \right\|. \tag{3.12}$$

Further, it can be seen that (3.11) represents a non-linear least squares problem when rewritten as

$$\boldsymbol{x'} \coloneqq \underset{\boldsymbol{x'}}{\operatorname{argmin}} \sum_{l=1}^{n} \left(\boldsymbol{u_l} - \frac{1}{P_{l_3} \widetilde{\boldsymbol{x'}}} \begin{pmatrix} P_{l_1} \widetilde{\boldsymbol{x'}} \\ P_{l_2} \widetilde{\boldsymbol{x'}} \end{pmatrix} \right)^2,$$

whereby P_{l_i} denotes the i-th row of the camera matrix P_l. Therefore, it is not solvable in a trivial manner. For this reason, the solution of (3.11) can only be obtained using more complex methods. The most common approach to solve this problem is the Levenberg-Marquardt Method [31] which is an iterative technique to solve non-linear least squares problems.

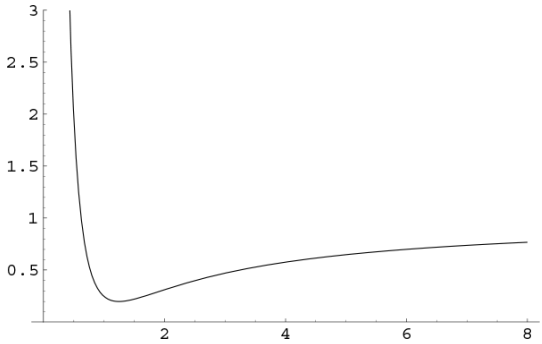


Figure 3.4: Graph of some Function $f_l(x'(t))^2$ using a parameterized x' [32].

However, the Levenberg-Marquardt Method only converges to local minima. If the formulated problem possesses multiple minima, then the found minimum may not be the global minimum. Unfortunately, (3.11) is a problem for which multiple minima may

exist. This can be shown in two separate ways. Firstly, R. I. Hartley and P. Sturm discuss in [26] the stereo-view case of the L_2 Triangulation. There, the minimalization problem (3.11) is reformulated to finding the roots of a polynomial of degree six. Such a polynomial has up to three local minima [32]. Consequently, even the simplest case in which only two camera views are being utilized is already subject to the local minima issue. It should be noted that such a reformulation to a polynomial cannot be performed if more than two views are used as stated in [32]. A second derivation of the possible existence of multiple minima is elaborated in [32]. There, Hartley and Schaffalitzky consider the equivalent problem formulation of (3.12), or more specifically

$$\boldsymbol{x}' = \underset{\boldsymbol{x}'}{\operatorname{argmin}} \|\boldsymbol{f}(\boldsymbol{x}')\|^2 = \underset{\boldsymbol{x}'}{\operatorname{argmin}} \sum_{l=1}^{n} f_l(\boldsymbol{x}')^2$$

with $f_l(\boldsymbol{x}') \coloneqq \|\boldsymbol{u_l} - \boldsymbol{u_l'}\|$. They further consider a parametrization for $\boldsymbol{x}'$ as a point on a line in front of a camera l, i.e. $\boldsymbol{x}'(t) = \boldsymbol{x_0'} + t\boldsymbol{d}'$. Using this parametrization, the cost function's component $f_l\big(\boldsymbol{x}'(t)\big)^2$ can be expressed through

$$f_l\big(\boldsymbol{x}'(t)\big)^2 = \frac{a + bt + ct^2}{(d + et)^2} \tag{3.13}$$

for $a, b, c, d, e \in \mathbb{R}$. A derivation of this expression can be found in [32]. From geometric intuition, it follows that $f_l\big(\boldsymbol{x}'(t)\big)^2$ must have a single minimum, as all considered values of $\boldsymbol{x}'$ lie in front of camera l. However, upon further inspection it can be seen that (3.13) is not always a convex function, i.e. there may exist line segments on the graph of $f_l\big(\boldsymbol{x}'(t)\big)^2$ between any two $f_l\big(\boldsymbol{x}'(t_1)\big)^2$ and $f_l\big(\boldsymbol{x}'(t_2)\big)^2$ which lie below the graph. An illustration of this non-convexity can be seen in Figure 3.4 where the graph of some function $f_l\big(\boldsymbol{x}'(t)\big)^2$ is visualized. As the functions $f_l\big(\boldsymbol{x}'(t)\big)^2$ are not guaranteed to be convex, their sum is also not guaranteed to be convex. For this reason, it may possess multiple minima.

Another problem to consider is the chirality [11], [32]. It states that the triangulated point $\boldsymbol{x}'$ shall lie inside the so-called chirality domain. This domain is defined as the intersection of all half-spaces bound by the camera views' principle planes, i.e. the convex domain in which each point is in front of all camera views. This observation is obvious as $\boldsymbol{x}'$ should be visible from all camera views because the camera matrices $P_1, \dots, P_n$ are assumed to be noiseless. However, the projective space does not formulate this chirality condition. Instead, it also projects points that lie behind a camera onto the image plane. A condition to check if $\boldsymbol{x}'$ lies inside the chirality domain can be, using (2.2), formulated as

$$\forall l: \widetilde{u_{l_3}'} > 0 \tag{3.14}$$

with $\widetilde{\boldsymbol{u}_{\boldsymbol{l}}'} = P_l \widetilde{\boldsymbol{x}'}$.

In principle, the L_2 Triangulation is able to find the optimal solution $\boldsymbol{x}'$ for multiple views assuming noiseless camera matrices P_l and image points $\boldsymbol{c_l}$ afflicted by Gaussian noise. For this, any algorithm solving a non-linear least squares problem may be used, e.g. the Levenberg-Marquardt Method. However, these algorithms more often than not find local minima of the L_2 Triangulation's minimization problem. Therefore, the found minimum does not have to match the global minimum of the problem and, therefore, a non-optimal solution may be found. For the Levenberg-Marquardt Method, the minimum that is found highly depends on the initial point $\boldsymbol{x'}^{(\boldsymbol{0})} \in \mathbb{R}^3$. Therefore, it may have to be executed multiple times, each with a different initial point $\boldsymbol{x'}^{(\boldsymbol{0})}$, in order to retrieve the optimal solution. Again, the L_2 Triangulation may be used in any UAS implementing some object detection algorithm. However, as stated before, the camera matrices are assumed to be noiseless using this algorithm. For most UASs, this does not yield true as both, the cameras' positions as well as their orientation must be assumed to be afflicted by noise. Furthermore, this triangulation also requires the solution of a problem of high complexity, i.e. a non-linear least squares problem. The time required to find the optimal solution may be too high to achieve real-time capability which is an essential criterion for the integration into the APOLI project. Therefore, a computationally more feasible approach is more desirable when dealing with noise afflicted image points.

3.2.5 L_∞ Triangulation

In 2004, R. I. Hartley and F. Schaffalitzky [32] introduced a multi-view triangulation approach which aims to tackle the local minima issue of the L_2 Triangulation. More specifically, it formulates a cost function that has exactly one local, and therefore global, minimum in the chirality domain. For this reason, solving the resulting minimization problem is simplified drastically and the risk of converging into local minima is eliminated.

Instead of finding the $\boldsymbol{x}'$ that minimizes the L_2 norm of $\boldsymbol{f}(\boldsymbol{x}') = (\|\boldsymbol{u_1} - \boldsymbol{u_1'}\|, \dots, \|\boldsymbol{u_n} - \boldsymbol{u_n'}\|)^T$, as it was done in the L_2 Triangulation, the L_∞ Triangulation instead finds the $\boldsymbol{x}'$ which minimizes the L_∞ norm of $\boldsymbol{f}(\boldsymbol{x}')$, i.e.

$$\boldsymbol{x}' = \underset{\boldsymbol{x'}}{\operatorname{argmin}} \|\boldsymbol{f}(\boldsymbol{x}')\|_\infty = \underset{\boldsymbol{x'}}{\operatorname{argmin}} \left\| \begin{pmatrix} \|\boldsymbol{u_1} - \boldsymbol{u_1'}\| \\ \vdots \\ \|\boldsymbol{u_n} - \boldsymbol{u_n'}\| \end{pmatrix} \right\|_\infty, \tag{3.15}$$

whereby $\|\boldsymbol{a}\|_\infty$ denotes the L_∞ norm of a vector $\boldsymbol{a}$. It is defined as the maximum absolute component of $\boldsymbol{a}$, i.e. $\|\boldsymbol{a}\|_\infty := \max_l |a_l|$. The minimization problem of (3.15) can, therefore, be expressed as

$$\boldsymbol{x}' = \underset{\boldsymbol{x}'}{\operatorname{argmin}} \max_l \|\boldsymbol{u_l} - \boldsymbol{u_l'}\|.$$

To prove that the cost function $\|\boldsymbol{f}(\boldsymbol{x}')\|_\infty$ has a single minimum in the chirality domain, Hartley and Schaffalitzky [32], again, consider the parametrization of $\boldsymbol{x}'$ as a point on a line $\boldsymbol{x}'(t)$, i.e. $\boldsymbol{x}'(t) = \boldsymbol{x_0'} + t\boldsymbol{d}'$. Here, the largest domain T is chosen for t so $\boldsymbol{x}'(t)$ lies completely inside the chirality domain. Lines $\boldsymbol{x}'(t)$ for which the domain T is empty are not being considered. As the chirality domain is convex, T is an interval. Additionally, as stated in Chapter 3.2.4, each function $f_l(\boldsymbol{x}'(t))$ has a single minimum for $t \in T$. Contrary to the assumption that $\|\boldsymbol{f}(\boldsymbol{x}'(t))\|_\infty$ only possesses one minimum in T, suppose that there are two minima at $t_1 \in T$ and $t_2 \in T$ and, without loss of generality,

$$\|\boldsymbol{f}(\boldsymbol{x}'(t_1))\|_\infty > \|\boldsymbol{f}(\boldsymbol{x}'(t_2))\|_\infty$$

as well as $t_1 < t_2$. Then there has to be a $t' \in T$ with $t' > t_1$ for which $\|\boldsymbol{f}(\boldsymbol{x}'(t'))\|_\infty > \|\boldsymbol{f}(\boldsymbol{x}'(t_1))\|_\infty$ holds true. Further, there is a component $f_l(\boldsymbol{x}'(t))$ of the cost function $\|\boldsymbol{f}(\boldsymbol{x}'(t))\|_\infty$ for which $f_l(\boldsymbol{x}'(t')) = \|\boldsymbol{f}(\boldsymbol{x}'(t'))\|_\infty$.It follows

$$f_l(\boldsymbol{x}'(t')) > f_l(\boldsymbol{x}'(t_1))$$

and

$$f_l(\boldsymbol{x}'(t')) > f_l(\boldsymbol{x}'(t_2)).$$

Therefore, the function $f_l(\boldsymbol{x}'(t'))$ must have a minimum each in $[t_1; t') \subset T$ and $(t'; t_2] \subset T$ which, however, contradicts the observation that each $f_l(\boldsymbol{x}'(t))$ possesses a single one minimum in T. Consequently, $\|\boldsymbol{f}(\boldsymbol{x}'(t))\|_\infty$ has exactly one minimum along any line in the chirality domain. Further, $\|\boldsymbol{f}(\boldsymbol{x}')\|_\infty$ possesses a single minimum in the chirality domain itself. This is trivial to demonstrate as if there were at least two minima in the chirality domain, $\|\boldsymbol{f}(\boldsymbol{x}'(t))\|_\infty$, whereby $\boldsymbol{x}'(t)$ is a line joining two of these minima, itself has two minima.

To solve the minimization problem of (3.15), a simple iterative algorithm is elaborated in [32]. Firstly, an initial estimate for $\boldsymbol{x}'$ must be determined, i.e. $\boldsymbol{x}'^{(\boldsymbol{0})}$. For example, it may be found by solving the linear programming problem formulated by (3.14). After an initial point has been chosen, the iteration step m is initialized to 0 and a random direction $\boldsymbol{v}^{(\boldsymbol{0})}$ is determined. Using that, a line

$$\boldsymbol{l}^{(m)}(t) \coloneqq \boldsymbol{x'}^{(m)} + t\boldsymbol{v}^{(m)} \tag{3.16}$$

can be formulated. The parameter domain of t is chosen as large as possible while still guaranteeing that $\boldsymbol{l}^{(m)}$ lies inside the chirality domain for all t. Now, knowing that $\|\boldsymbol{f}(\boldsymbol{x'})\|_\infty$ only has one minimum in the chirality domain and, therefore, only one minimum on $\boldsymbol{l}^{(m)}$, any line search algorithm may be used to determine the $t^{(m)}$ for which

$$\left\|\boldsymbol{f}\left(\boldsymbol{l}^{(m)}\left(t^{(m)}\right)\right)\right\|_\infty$$

is minimal. In [32], Hartley and Schaffalitzky chose the Fibonacci Line Search [33] for this. After the minimum has been determined by estimating the optimal line parameter $t^{(m)}$, the current estimate of $\boldsymbol{x'}^{(m)}$ can be improved upon using

$$\boldsymbol{x'}^{(m+1)} \coloneqq \boldsymbol{l}^{(m)}\left(t^{(m)}\right).$$

This concludes the iteration step and m is incremented by one. Subsequently, a new random direction $\boldsymbol{v}^{(m)}$ is chosen and the next iteration step starts again at (3.16). This algorithm terminates after a set number of iteration steps. Finally, an approximation for the triangulation solution of (3.15) is determined as

$$\boldsymbol{x'} \coloneqq \boldsymbol{x'}^{(m)}.$$

In comparison to the L_2 Triangulation, the L_∞ Triangulation eliminates the possible existence of multiple minima in the chirality domain. Therefore, the risk of finding a non-optimal solution for $\boldsymbol{x'}$ is eliminated and the problem is much easier to solve. It should be noted that the found solution $\boldsymbol{x'}$ is only optimal in regards to (3.15) and most likely does not represent a solution of (3.11). However, it is obvious that outliers in the input data, i.e. $\boldsymbol{u_l}$ which are heavily affected by noise, have a greater impact on the result of the L_∞ Triangulation than on the result of the L_2 Triangulation. Nevertheless, those outliers may be removed prior to the triangulation using algorithms like RANSAC [34]. Because of the lowered complexity, the L_∞ Triangulation may be preferred over the L_2 Triangulation for usage in most UASs and especially the APOLI project. That being said, the camera matrices P_l are still assumed to be noiseless for the L_∞ Triangulation which is not realistic for most UASs.

3.3 Multi-View Ellipsoid Approximation using Bounding Rectangles

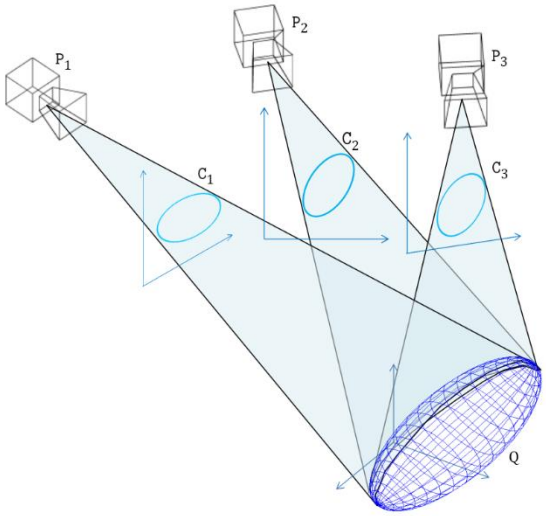


Figure 3.5: Multi-View Ellipsoid Approximation using Bounding Rectangles. The Figure is based on an Image from [35].

In 2018, Rubino et al. [35] presented another multi-view object localization approach. The basic concept of this algorithm is visualized in Figure 3.5. Their algorithm utilizes the bounding rectangles around an object in captured camera images as well as the corresponding camera matrices. More specifically, it approximates objects as ellipsoids. By reformulating the problem to use an ellipsoid as well as ellipses, instead of the bounding rectangles directly, it becomes mathematically much more feasible and, therefore, easier to solve. In the following, the ellipsoid that describes the object to localize precisely is denoted as $Q \in \mathbb{R}^{4\times4}$, i.e. given in a quadric matrix form [36]. Note that in this form Q is symmetrical, i.e. $Q^T = Q$. In an optimal noiseless environment, the ellipses that result from projecting Q onto the image planes perfectly fit inside the bounding rectangles of the corresponding view. The projection of Q onto the image plane of view l is referred to as the ellipse $C_l \in \mathbb{R}^{3\times3}$ given as the matrix representation of a conic section. Again, C_l is a symmetric matrix. Because the ellipsoid Q is unknown, the ellipses C_l are constructed using the bounding rectangles. Furthermore, the camera matrices are given as $P_1 = K_1[R_1|\boldsymbol{t_1}], \dots, P_n = K_n[R_n|\boldsymbol{t_n}] \in \mathbb{R}^{3\times4}$. The goal of the following algorithms is to find an ellipsoid Q', given the ellipses C_l as well as the camera matrices P_l, which best explains Q. Obviously, this approximation highly depends on the quality of the bounding rectangles. While the Noiseless Ellipsoid Approximation, elaborated in Chapter 3.3.1 assumes the absence of noise, all other algorithms aim to reduce the influence of noise in the input data on the localization result Q'. All described algorithms require at least three different views, i.e. $n \geq 3$.

3.3.1 Noiseless Ellipsoid Approximation

If the input data is noiseless, each C_l perfectly represents the projection of Q onto the image plane of view l. In this case, the Noiseless Ellipsoid Approximation, presented in [35] and elaborated in this chapter, can be used to determine Q'. To check whether an image point $\boldsymbol{u} \in \mathbb{R}^2$ lies on the ellipse C_l, the condition

$$\tilde{\boldsymbol{u}}^T C_l \tilde{\boldsymbol{u}} = 0$$

may be used [36]. Analogously, a three-dimensional point $\boldsymbol{x} \in \mathbb{R}^3$ lies on the surface of the ellipsoid Q if, and only if,

$$\tilde{\boldsymbol{x}}^T Q \tilde{\boldsymbol{x}} = 0$$

yields true. From these conditions, it is obvious that both matrices Q and C_l are scale-invariant.

Nevertheless, the projection between the ellipsoid Q and the ellipses C_l is not trivial to derive in the Euclidean space. Therefore, Rubino et al. [35] reformulate the projection into the so-called dual space. In this space an ellipsoid is expressed by the envelope of all the planes tangent to it while an ellipse is represented by the envelope of all the lines tangent to it. To convert the ellipsoid Q and the ellipses C_l into this space,

$$\hat{Q} \coloneqq \mathrm{adj}(Q),$$
$$\hat{C}_l \coloneqq \mathrm{adj}(C_l)$$

may be used, whereby

$$\mathrm{adj}(A) \coloneqq \mathrm{cof}(A)^T = \left(\left((-1)^{o+p} A_{o,p} \right)_{1 \le o,p \le n} \right)^T$$

denotes the adjoint matrix [36] of $A \in \mathbb{R}^{n \times n}$. Here, $A_{o,p}$ is the (o,p)-minor of A, i.e. the determinant of a submatrix formed by deleting the o-th column and the p-th row from A. The projection between the ellipsoid and the ellipses can now be expressed via

$$s_l \hat{C}_l = P_l \hat{Q} P_l^T. \tag{3.17}$$

The $s_l \in \mathbb{R}$ is the scale factor which derives from the scale-invariance of Q and C_l. (3.17) shall now be solved to find $\hat{Q}$. For this, the linear system is reformulated to

$$s_l \hat{\boldsymbol{c}}_{\boldsymbol{l}} = G_l \hat{\boldsymbol{q}} \tag{3.18}$$

with $\hat{\boldsymbol{q}} \coloneqq \mathrm{vecs}(\hat{Q}) \in \mathbb{R}^{10}$ as the vectorization of the lower triangle part of $\hat{Q}$ and $\hat{\boldsymbol{c}}_{\boldsymbol{l}} \coloneqq \mathrm{vecs}(\hat{C}_l) \in \mathbb{R}^6$ as an analogous vectorization of $\hat{C}_l$. Furthermore, $G_l \in \mathbb{R}^{6 \times 10}$ is determined as

$$G_l = A(P_l \otimes P_l) B,$$

whereby $P_l \otimes P_l$ denotes the Kronecker product of P_l with itself. Additionally, $A \in \mathbb{R}^{6 \times 9}$ and $B \in \mathbb{R}^{16 \times 10}$ are two arbitrary matrices for which $\mathrm{vecs}(X) = A\,\mathrm{vec}(X)$ and $\mathrm{vecs}(Y) = B\,\mathrm{vec}(Y)$ for any symmetrical matrices $X \in \mathbb{R}^{9 \times 9}$ and $Y \in \mathbb{R}^{16 \times 16}$ hold true. Here, the $\mathrm{vec}(*)$ operator vectorizes all elements of a matrix. A closed-form solution for G_l only utilizing the values of the camera matrix P_l is provided in [35].

By applying all n views on (3.18), the system

$$M\boldsymbol{w} = \mathbf{0} \tag{3.19}$$

is derived. The matrix M and the vector $\boldsymbol{w}$ are built as

$$M \coloneqq \begin{bmatrix} G_1 & -\widehat{\boldsymbol{c_1}} & \mathbf{0} & \cdots & \mathbf{0} \\ G_2 & \mathbf{0} & -\widehat{\boldsymbol{c_2}} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_n & \mathbf{0} & \mathbf{0} & \cdots & -\widehat{\boldsymbol{c_n}} \end{bmatrix} \in \mathbb{R}^{6n\times(10+n)}, \tag{3.20}$$

$$\boldsymbol{w} \coloneqq \begin{pmatrix} \widehat{\boldsymbol{q}} \\ s_1 \\ \vdots \\ s_n \end{pmatrix} \in \mathbb{R}^{10+n}. \tag{3.21}$$

The solution of (3.19) can be found by minimizing

$$\boldsymbol{w}' \coloneqq \underset{\boldsymbol{w}}{\operatorname{argmin}} \|M\boldsymbol{w}\|^2$$

under the constraint

$$\|\boldsymbol{w}'\|^2 = 1, \tag{3.22}$$

whereby (3.22) is used to avoid the trivial solution of $\boldsymbol{w}' = \mathbf{0}$. This minimization problem can be solved by applying a Singular Value Decomposition (SVD) [37] on M. The solution $\boldsymbol{w}'$ is the right singular vector associated with the minimum singular value. Using (3.21), the first ten values of $\boldsymbol{w}'$ represent $\widehat{\boldsymbol{q}}'$ which is further used to determine the ellipsoid Q' of the Euclidean space via

$$Q' \coloneqq \operatorname{adj}^{-1}\left(\operatorname{vecs}^{-1}\left(\widehat{\boldsymbol{q}}'\right)\right). \tag{3.23}$$

The inverse adjoint operation $\operatorname{adj}^{-1}(A)$ of a matrix $A \in \mathbb{R}^{n\times n}$ can be defined as

$$\operatorname{adj}^{-1}(A) \coloneqq \left(\frac{1}{\sqrt[n-1]{|A|}}A\right)^{-1}.$$

The Noiseless Ellipsoid Approximation provides a closed-form solution for Q' formulated in (3.23) which can be determined efficiently. By expressing the quadric matrix form Q' of the ellipsoid as

$$Q' = \begin{bmatrix} Q_1' & \boldsymbol{Q_2'} \\ Q_3' & Q_4' \end{bmatrix} \in \mathbb{R}^{4\times4}$$

with $Q_1' \in \mathbb{R}^{3\times3}, \boldsymbol{Q_2'} \in \mathbb{R}^3, Q_3' \in \mathbb{R}^{1\times3}$ and $Q_4' \in \mathbb{R}$, its information can easily be extracted [35], [36], [38]. While the ellipsoid's center $\boldsymbol{c_{Q'}}$ is calculated as

$$\boldsymbol{c_{Q'}} = -{Q_1'}^{-1}\boldsymbol{Q_2'}, \tag{3.24}$$

its principle axes are determined as

$$\boldsymbol{p_{Q'}}_{\boldsymbol{l}} \coloneqq \sqrt{-\frac{|Q'|}{|Q_1'|}\frac{1}{\lambda_m}}\boldsymbol{v_l} \in \mathbb{R}^3$$

with $\lambda_l \in \mathbb{R}$ as the l-th eigenvalue and $\boldsymbol{v_l} \in \mathbb{R}^3$ as the l-th eigenvector of Q_1'. However, the Noiseless Ellipsoid Approximation assumes an environment in which the provided camera matrices P_l and the ellipses C_l are noiseless. While this algorithm in theory also works if this data is afflicted by noise, the solution Q' that results from such noise may not be a valid ellipsoid. Instead, Q' could represent a nearly degenerated ellipsoid or even a different quadric, e.g. a hyperboloid. To test whether Q' represents a valid ellipsoid,

$$\forall l \in \{1,2,3\}: \lambda_l > 0 \vee \forall l \in \{1,2,3\}: \lambda_l < 0 \tag{3.25}$$

may be used [39]. For this reason, the Noiseless Ellipsoid Approximation is only feasible if the provided data can be assumed to be noiseless, which is not realistic in real-world application. Also, the coordinates of the bounding rectangles are, when using real images, integers which introduces additional noise. Therefore, this approach is not feasible for application in a UAS where the camera matrices and the bounding rectangles must be assumed to be affected by noise. Instead, an algorithm that handles noise in the data in a better way is more desirable.

3.3.2 LfD Ellipsoid Approximation

In [35], Rubino et al. also present a modification of the Noiseless Ellipsoid Approximation that is able to deal with higher noise while still leading to a closed-form solution for Q'. In their paper, Rubino et al. call this modified algorithm the Localisation from Detection algorithm. Throughout this thesis, it will simply be referred to as the LfD Ellipsoid Approximation. The LfD Ellipsoid Approximation utilizes the observation that high diversities in the magnitude of the elements of M, as formulates in (3.21), is a potential source of its ill-conditioning when dealing with noise. An ill-conditioned matrix M is easily influenced by noise in the camera matrices P_l and the ellipses C_l in a way in which the solution $\boldsymbol{w}'$ results in an invalid ellipsoid Q'.

To deal with this issue, Rubino et al. [35] acknowledge that the ellipses C_l themselves are a source of the element magnitude diversity of M. Therefore, they introduce a precondition step in which they calculate the normalized ellipses $\hat{\hat{C}}_l$ that, when transformed using an affine transformation H_l, result in the ellipses $\widehat{C}_l$. Here, an ellipse is said to be normalized when it is centered in its corresponding image plane at $(0,0)^T$ and if its axis lengths are normalized. In particular, this yields

$$\widehat{C}_l \coloneqq H_l \hat{\hat{C}}_l H_l^T$$

with

$$H_l \coloneqq \begin{bmatrix} h_l & 0 & c_{l_1} \\ 0 & h_l & c_{l_2} \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3},$$

$$\hat{\dot{C}}_l \coloneqq \begin{bmatrix} \hat{\dot{c}}_{l_{11}} & \hat{\dot{c}}_{l_{12}} & 0 \\ \hat{\dot{c}}_{l_{12}} & \hat{\dot{c}}_{l_{22}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times3}.$$

$c_{l_1} \in \mathbb{R}$ and $c_{l_2} \in \mathbb{R}$ are the coordinates of the center of the ellipse C_l in its image plane. Further, $h_l \in \mathbb{R}$ is defined as

$$h_l \coloneqq \sqrt{s_{l_1}^2 + s_{l_2}^2}$$

with $s_{l_1} \in \mathbb{R}$ and $s_{l_2} \in \mathbb{R}$ as the semi axes of C_l, i.e. half of its axis lengths. Lastly, $\hat{\dot{c}}_{l_{11}}, \hat{\dot{c}}_{l_{12}}, \hat{\dot{c}}_{l_{22}} \in \mathbb{R}$ are defined so $\dot{C}_l$ has normalized axis lengths.

In addition to this, the quadric representation of the resulting ellipsoid Q' is subject to a similar problem as the matrix M itself. This means that the translation values of Q' neglect the shape parts of Q' if its translation values are sufficiently large. To deal with this issue, a second precondition step is being introduced using a transformation matrix T. It is chosen so the resulting ellipsoid of the LfD Ellipsoid Approximation lies around the origin $(0,0,0)^T$. To retrieve the translation offset, the solution of the Noiseless Ellipsoid Approximation Q' is used. More specifically, T describes the translation of the origin to the center of Q'. Therefore, T is, using $\boldsymbol{c}_{\boldsymbol{Q}'}$ as calculated in (3.24), determined as

$$T \coloneqq \begin{bmatrix} I & \boldsymbol{c}_{\boldsymbol{Q}'} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{3.26}$$

with $I \in \mathbb{R}^{3\times3}$ as the identity matrix. Alternatively, the center $\boldsymbol{c}_{\boldsymbol{Q}'}$ can be directly calculated using the dual space representation of Q', i.e. $\widehat{Q'}$, via

$$\boldsymbol{c}_{\boldsymbol{Q}'} \coloneqq \frac{1}{\widehat{Q'}_{4,4}} \begin{pmatrix} \widehat{Q'}_{1,4} \\ \widehat{Q'}_{2,4} \\ \widehat{Q'}_{3,4} \end{pmatrix}.$$

This eliminates the need to invert the upper left 3×3 sub-matrix of Q' which may not be invertible. Additionally, there is no need to convert $\widehat{Q'}$ back into the Euclidean space.

Using these transformation matrices, the LfD Ellipsoid Approximation derives its solution is a similar way as the Noiseless Ellipsoid Approximation. The projection between the ellipsoid in the dual space $\hat{Q}$ and the normalized ellipses in the dual space $\hat{\dot{C}}_l$ is now determined as

$$s_l \hat{\dot{C}}_l = H_l^{-1} P_l T \hat{Q} T^T P_l^T H_l^{-T}$$

with the pseudo-centered ellipsoid $\hat{Q}$ determined through $\hat{Q} \coloneqq T^{-1} \hat{Q} T^{-T}$. When substituting $H_l^{-1} P_l T$ to new camera matrices $\dot{P}_l$, these matrices can be used to build the matrix $\dot{G}_l$ in the same manner as G_l. Again, the system

$$\dot{M}\dot{\boldsymbol{w}} = \mathbf{0} \tag{3.27}$$

is derived and the minimization problem

$$\dot{\boldsymbol{w}}' \coloneqq \underset{\dot{\boldsymbol{w}}}{\operatorname{argmin}} \left\| \dot{M}\dot{\boldsymbol{w}} \right\|^2 \tag{3.28}$$

under the constraint

$$\|\dot{\boldsymbol{w}}'\|^2 = 1$$

with

$$\dot{M} \coloneqq \begin{bmatrix} \dot{G}_1 & -\widehat{\dot{\boldsymbol{c}}_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \dot{G}_2 & \mathbf{0} & -\widehat{\dot{\boldsymbol{c}}_2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{G}_n & \mathbf{0} & \mathbf{0} & \cdots & -\widehat{\dot{\boldsymbol{c}}_n} \end{bmatrix} \in \mathbb{R}^{6n \times (10+n)}$$

follows. Solving it by applying an SVD yields $\dot{\boldsymbol{w}}'$. Once more, its first ten elements represent $\widehat{\dot{\boldsymbol{q}}'}$. Using it, the pseudo-centered ellipsoid $\widehat{\dot{Q}'}$ is calculated as

$$\widehat{\dot{Q}'} \coloneqq \operatorname{vecs}^{-1}\left(\widehat{\dot{\boldsymbol{q}}'}\right).$$

Applying the inverse transformation of T yields the solution Q'.

$$Q' \coloneqq \operatorname{adj}^{-1}\left(T\widehat{\dot{Q}'}T^T\right).$$

The LfD Ellipsoid Approximation adds further transformations to the minimization problem (3.28) while avoiding to introduce complex constraints. Again, Q' can be obtained as a closed-form solution. Furthermore, the transformations help reducing the ill-conditioning problem of M resulting in the modified matrix $\dot{M}$. Therefore, the resulting quadric Q' is more likely to be a valid ellipsoid. For this reason, this algorithm is more feasible if the provided input data may be afflicted by noise. In particular, it is to be preferred for usage in UASs and, therefore, the APOLI project. However, it is still by no means guaranteed that Q' is an ellipsoid. Again, this can be tested using (3.25).

3.3.3 LfDC Ellipsoid Approximation

In [40], Gay and Bue further modify the LfD Ellipsoid Approximation by introducing additional constraints. They call their approach the Localisation from Detection with Constraints algorithm. Throughout this thesis, it is simply referred to as the LfDC Ellipsoid Approximation. It should be noted that in [40] Gay and Bue do not explicitly state that the introduced modifications are applied to the LfD Ellipsoid Approximation in contrast to the simpler Noiseless Ellipsoid Approximation. However, this is implied by the name of their modified algorithm as well as the modifications themselves.

In particular, Gay and Bue use the observation that in a noiseless environment the projections of the center of the ellipsoid Q, i.e. $\boldsymbol{c}_{\boldsymbol{Q}} \in \mathbb{R}^3$, match the center $\boldsymbol{c}_{\boldsymbol{C}_i} \in \mathbb{R}^2$ of

any ellipse C_l precisely. In the Euclidean space, this observation can be formulated easily as

$$\widetilde{\boldsymbol{c}_{\boldsymbol{C}_l}} = P_l \widetilde{\boldsymbol{c}_{\boldsymbol{Q}}} \ \forall l.$$

To derive the mathematical modification used in the LfDC Ellipsoid Approximation, Gay and Bue further consider the vectorized dual space representations of Q and C_l, i.e. $\widehat{\boldsymbol{q}}$ and $\widehat{\boldsymbol{c}_{\boldsymbol{l}}}$. These are composed of

$$\widehat{\boldsymbol{q}} = \begin{pmatrix} {c_Q}_1^2 - a_1 \\ {c_Q}_1 {c_Q}_2 - a_2 \\ {c_Q}_1 {c_Q}_3 - a_3 \\ {c_Q}_1 \\ {c_Q}_2^2 - a_4 \\ {c_Q}_2 {c_Q}_3 - a_5 \\ {c_Q}_2 \\ {c_Q}_3^2 - a_6 \\ {c_Q}_3 \\ 1 \end{pmatrix} \text{ and } \widehat{\boldsymbol{c}_{\boldsymbol{l}}} = \begin{pmatrix} {c_{C_l}}_1^2 - b_{l_1} \\ {c_{C_l}}_1 {c_{C_l}}_2 - b_{l_2} \\ {c_{C_l}}_1 \\ {c_{C_l}}_1^2 - b_{l_3} \\ {c_{C_l}}_2 \\ 1 \end{pmatrix}. \tag{3.29}$$

Again, $\boldsymbol{c_Q}$ is the center of Q while $\boldsymbol{c_{C_l}}$ refers to C_l's center. The orientation as well as the scaling of these objects are encoded in the values of $a_m \in \mathbb{R}$ and $b_{l_m} \in \mathbb{R}$ respectively. Upon further inspection of (3.29), it can be seen that in certain rows $\boldsymbol{c_Q}$ and $\boldsymbol{c_{C_l}}$ appear independently of the orientation and scaling. Using this observation, an additional linear system per view can be derived as

$$s_l' \boldsymbol{c_{C_l}} = {G_l}' \widehat{\boldsymbol{q}}, \tag{3.30}$$

whereby

$${G_l}' := \begin{bmatrix} 0 & 0 & 0 & {P_l}_{1,1} & 0 & 0 & {P_l}_{1,2} & 0 & {P_l}_{1,3} & {P_l}_{1,4} \\ 0 & 0 & 0 & {P_l}_{2,1} & 0 & 0 & {P_l}_{2,2} & 0 & {P_l}_{2,3} & {P_l}_{2,4} \end{bmatrix} \in \mathbb{R}^{2\times 10}$$

ensures the projection of the ellipsoid's center $\boldsymbol{c_Q}$ onto the ellipse's center $\boldsymbol{c_{C_l}}$. Here, ${P_l}_{m,n}$ denotes the element of P_l at the m-th column and the n-th row. Note that the scaling factor $s_l' \in \mathbb{R}$ is usually unequal to $s_l \in \mathbb{R}$ as formulated in the Noiseless Ellipsoid Approximation. While s_l is used in the projection of the ellipsoid $\hat{Q}$ onto the ellipse $\widehat{C_l}$, i.e. a dual space projection, s_l' compensates for the missing projection component $\widetilde{{c_C}_{l_3}}$ of the Euclidean space projection. One could now modify the system (3.19) of the Noiseless Ellipsoid Approximation by extending M. However, this would not only lead to an increased number of rows but also additional columns. That being said, (3.30) can be simplified if all of the ellipses' centers are located at the origin $(0,0)^T$ of their corresponding image plane, i.e. when considering the LfD Ellipsoid Approximation as a base. It follows

$$s_l' \boldsymbol{c_{\dot{C}_l}} = \boldsymbol{0} = {\dot{G}_l}' \widehat{\boldsymbol{q}}, \tag{3.31}$$

whereby $\dot{G}_l{}'$ is now computed using the substituted camera matrix $\dot{P}_l$:

$$\dot{G}_l{}' \coloneqq \begin{bmatrix} 0 & 0 & 0 & \dot{P}_{l_{1,1}} & 0 & 0 & \dot{P}_{l_{1,2}} & 0 & \dot{P}_{l_{1,3}} & \dot{P}_{l_{1,4}} \\ 0 & 0 & 0 & \dot{P}_{l_{2,1}} & 0 & 0 & \dot{P}_{l_{2,2}} & 0 & \dot{P}_{l_{2,3}} & \dot{P}_{l_{2,4}} \end{bmatrix} \in \mathbb{R}^{2\times 10}.$$

Using (3.31), the system (3.27) of the LfD Ellipsoid Approximation can be modified, resulting in

$$\dot{M}'\dot{\boldsymbol{w}} = \mathbf{0} \tag{3.32}$$

with

$$\dot{M}' \coloneqq \begin{bmatrix} \dot{G}_1 & -\widehat{\dot{\boldsymbol{c}}_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \dot{G}_1{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \dot{G}_2 & \mathbf{0} & -\widehat{\dot{\boldsymbol{c}}_2} & \cdots & \mathbf{0} \\ \dot{G}_2{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{G}_n & \mathbf{0} & \mathbf{0} & \cdots & -\widehat{\dot{\boldsymbol{c}}_n} \\ \dot{G}_n{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{8n\times(10+n)}.$$

As before, (3.32) can be solved by applying an SVD on $\dot{M}'$ which yields the solution $\dot{\boldsymbol{w}}'$ of the corresponding minimization problem. The localization solution Q' is then derived via

$$Q' \coloneqq \mathrm{adj}^{-1}\left(T\,\mathrm{vecs}^{-1}\left(\widehat{\dot{\boldsymbol{q}}'}\right)T^T\right).$$

The LfDC Ellipsoid Approximation elaborated in this chapter is a simple modification of the LfD Ellipsoid Approximation. Again, the solution Q' is obtained as a closed-form solution and can, therefore, be determined efficiently. In particular, the LfDC Ellipsoid Approximation only adds two rows per view to the matrix $\dot{M}$. This yields the matrix $\dot{M}'$ from which the solution is obtained from by applying an SVD. Fortunately, these rows can be determined efficiently themselves, too, as the center of the ellipses, i.e. $\boldsymbol{c}_{\dot{C}_l}$, are by definition $(0,0)^T$. Furthermore, the $\dot{G}_l{}'$ that are used in these rows consist of individual components of the camera matrices $\dot{P}_l$ which are already required for other rows of $\dot{M}'$ and, therefore, determined already. However, the LfDC Ellipsoid Approximation leads to additional computational costs as an SVD is applied on the enlarged matrix $\dot{M}'$ instead of $\dot{M}$. Gay and Bue also state in [40] that their approach tends to produce more spherical shaped ellipsoids. Furthermore, they evaluated their approach in comparison to the LfD Ellipsoid Approximation and found better approximation results for the dataset they used. In respect to its usability on UASs, both the LfD and the LfDC Ellipsoid Approximation are feasible. However, the higher computational cost of the LfDC Ellipsoid Approximation has to be weighed against the higher accuracy of the approximated ellipsoids. It is worth noting that the LfDC Ellipsoid

Approximation does not, just like the other two algorithms, guarantee valid ellipsoids to be generated.

3.4 Simultaneous Localization and Mapping

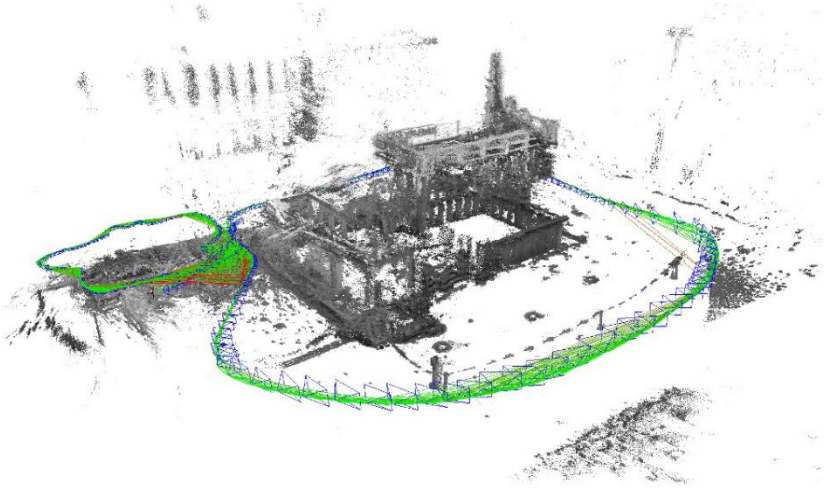


Figure 3.6: Point Cloud of an Environment generated using LSD-SLAM [41].

Simultaneous Localization and Mapping (SLAM) [28], [41], [42] follows a more complex approach than the localization algorithms discussed previously. Instead of only approximating the localization of an object, the SLAM problem is formulated as simultaneously localizing a system in its environment and creating a map of the environment itself at runtime. This problem is of complex nature and still heavily researched on. In particular, it is of high interest for the field of mobile robotics where the developed robot shall navigate in unknown environments autonomously. In the following, the visual SLAM approach [43] is discussed in more detail. This means that the input data is limited to visual information only, i.e. image streams generated by a camera system. While some state-of-the-art visual SLAM algorithms detect features in individual images and match them between images resulting in so-called keypoints, other SLAM algorithms may follow more direct approaches to match regions between different images, e.g. LSD-SLAM [41]. After regions of images have been matched, either via keypoints or more direct approaches, the system's localization as well as the environment map are approximated. Again, there are different approaches for this, e.g. using a Kalman Filter or the Bundle Adjustment [44] approach. However, the principle result of most visual SLAM algorithms is the same. They estimate the pose of the system as well as generate a map of the environment as a point cloud. Such a generated point cloud can be seen in Figure 3.6.

However, the SLAM approach is not feasible as an object localization algorithm in the context of this thesis. This is because the point cloud generated by most visual SLAM

algorithms lacks semantic information. While it does provide knowledge about the position of points in the three-dimensional Euclidean space which belong to objects, information about the correlation between these points is not generated. In particular, the localization of whole objects cannot be derived easily and reliably using this point cloud only. Instead, prior knowledge about the objects' models is required. As formulated in Chapter 1.2, however, information about the objects' models is not provided.

3.5 Summary

Triangulation Algorithm	*Localization Output*	*Feasible for noise-afflicted Input*	*Utilized Camera Views*	*Computational Complexity*
Noiseless	Point	No	Stereo-view	Very low
Midpoint	Point	Yes	Stereo-view	Very low
Generalized Midpoint	Point	Yes	Multi-view	Low
L_2	Point	Yes	Multi-view	Very high
L_∞	Point	Yes	Multi-view	High

Table 3.1: Some Criteria for the presented Triangulation Algorithms.

Ellipsoid Approximation Algorithm	*Localization Output*	*Feasible for noise-afflicted Input*	*Utilized Camera Views*	*Computational Complexity*
Noiseless	Ellipsoid	No	Multi-view	Medium
LfD	Ellipsoid	Yes	Multi-view	Medium to high
LfDC	Ellipsoid	Yes	Multi-view	Medium to high

Table 3.2: Some Criteria for the presented Ellipsoid Approximation Algorithms.

This chapter provides a brief summary of all state-of-the-art object localization algorithms which have been presented in the previous chapters. Again, the focus is laid on their applicability on UASs and in particular in regards to the APOLI project. For starters, the presented depth estimation algorithm as well as the SLAM approach, which have been discussed in Chapter 3.1 and 3.4 respectively, are not feasible for object localization on UASs. While the depth estimation algorithm puts too many restrictions on the allowed UAV movement and the orientation of the mounted camera, the SLAM approach requires prior knowledge about the models of objects because it does not generate any semantic information.

Table 3.1 and Table 3.2 list some criteria for the presented triangulation and ellipsoid approximation algorithms respectively. In particular, these criteria consist of the kind of generated localization output, whether the algorithm is feasible for noise-afflicted input data, the number of camera views utilized for the algorithm and, lastly, the algorithm's computational complexity. As discussed earlier, while the triangulation approach only approximates the position of an object as a single point, the ellipsoid approximation

estimates an ellipsoid of the object. From this, its position as well as an approximation of the object's shape can be derived. Additionally, only the Noiseless Triangulation as well as the Noiseless Ellipsoid Approximation are infeasible when dealing with noise afflicted input data. Note that this infeasibility is defined slightly different for these algorithms. While the problem solved by the Noiseless Triangulation, i.e. finding the point of intersection of two rays, simply does not make sense when dealing with noise afflicted data, the Noiseless Ellipsoid Approximation still attempts to solve a problem being well defined even if the input data is afflicted by noise. However, noise in the input data has a higher influence on the validity of the object localization than in comparison to the other ellipsoid approximation algorithms, i.e. the LfD and LfDC Ellipsoid Approximation. Furthermore, multi-view algorithms are to be preferred to stereo-view algorithms as noise-afflicted input data can be dealt with more efficiently.

4 Conceptualization

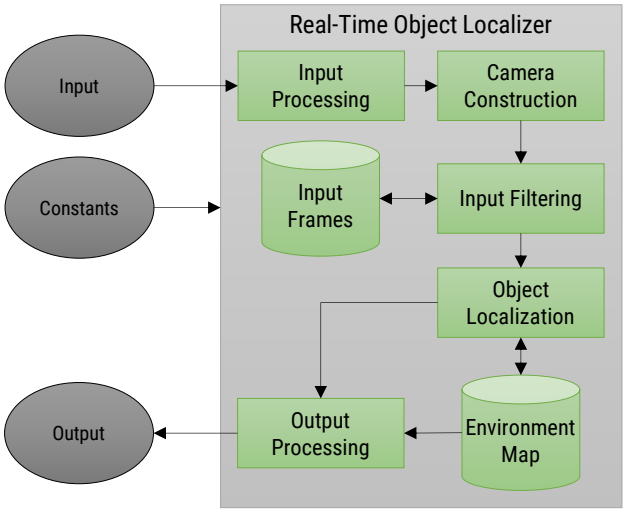


Figure 4.1: Concept of the Real-Time Object Localizer.

This chapter elaborates the concept of the developed object localization algorithm, i.e. the Real-Time Object Localizer (RTL), in detail. For this, some of the fundamentals explained in Chapter 2 and selected state-of-the-art object localization algorithms from Chapter 3 are being used. An overview of the RTL's concept is illustrated in Figure 4.1.

In Chapter 4.1, some requirements of the RTL are formulated. For this, its application in the APOLI project, which has been presented in Chapter 2.3, is considered. Subsequently, Chapter 4.2 elaborates the input and output data of a single frame as well as the constant data provided by the user beforehand. The following chapters 4.3 to 4.7 describe the individual steps performed by the RTL to localize objects. In the first step, the Input Processing which is explained in Chapter 4.3, the RTL performs some processing on the input data provided. More specifically, the input data is slightly modified to improve the accuracy of the object localization. The next step, elaborated in detail in Chapter 4.4, constructs the camera, i.e. its parameters, corresponding to the adjusted input. The calculated camera parameters as well as the input itself are then passed to the Input Filtering, presented in Chapter 4.5. In this step, a so-called Input Frame is generated. In its simplest form, it contains all of the input data as well as the camera parameters. This Input Frame is then added into a database of Input Frames. Subsequently, a set of Input Frames is selected from the database and passed to the next, most-essential step, the Object Localization. The Object Localization step utilizes the provided Input Frames to localize objects in a three-dimensional space. The localized objects are then added to the environment map

which may also be used for the localization process itself. Depending on the number of Fundamental Points per object in each of the Input Frames, the Object Localization is further split into two approaches. These approaches are the triangulation and the ellipsoid approximation. This step is described in detail in Chapter 4.6. As a last step, the Output Processing, which is elaborated in Chapter 4.7, uses the localization data in the environment map to generate the output of the RTL. The exact output generated by the Output Processing is explained in Chapter 4.2.3.

4.1 Requirements

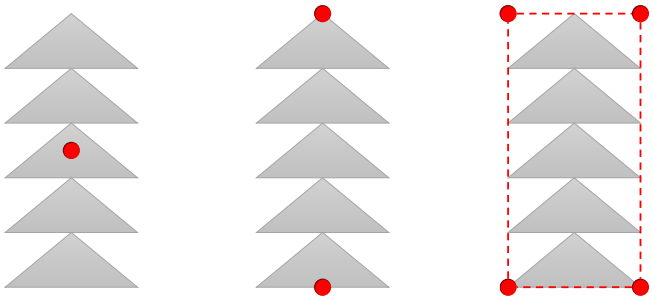


Figure 4.2: The Three Types of Fundament Points provided by the FPD. The Fundamental Points are highlighted in Red.

The RTL must satisfy certain requirements to be integratable into the APOLI project. It shall use some of the UAV's flight data, provided by the MAL, its gimbal orientation, provided by the CGC, the 2D location of a detected object and its object label in the camera image, provided by the FPD, as well as some predetermined constant data, e.g. the calibration of the camera. No further information is to be used to locate objects in the 3D space. The exact format of the 2D location data provided by the FPD varies depending on the FPD's configuration. In total, there are three configurations and, in all cases, the FPD provides points in or around the detected object. These points are hereinafter referred to as Fundamental Points. In its first configuration, as seen in Figure 4.2(a), the FPD presents a single Fundamental Points located at the center of the object. Figure 4.2(b) illustrates the second configuration in which two Fundamental Points are given, one above and one below the object. The last configuration is shown in Figure 4.2(c). There, the FPD provides four Fundamental Points around the object, essentially forming an axis-aligned bounding rectangle around it.

The application of the RTL for UAV flights leads to additional, more general, requirements. Firstly, its localization accuracy and robustness must be feasible for UAV flights. If they do not satisfy certain thresholds, objects may be located at incorrect locations which could lead to collisions. Furthermore, the algorithm must run in real-

time on the given hardware of the APOLI project, the ODROID-XU4 board. A localization frequency of at least 10Hz is desirable.

4.2 Input, Constants and Output

4.2.1 Input

Category	*Data*	*Notation*	*Units*
UAV flight data	UAV position	$\boldsymbol{c_{UAV}}$	m
	UAV movement speed	$\boldsymbol{v_{UAV}}$	m/s
	UAV positional timestamp	$t_{UAV_{Position}}$	
	UAV orientation	$\boldsymbol{\theta_{UAV}}$	rad
	UAV angular speed	$\boldsymbol{\omega_{UAV}}$	rad/s
	UAV orientational timestamp	$t_{UAV_{Orientation}}$	
	Origin of UAV Local NED	$\boldsymbol{o}$	GNSS
Gimbal orientation	Gimbal orientation	$\boldsymbol{\theta_{Gimbal}}$	rad
	Gimbal angular speed	$\boldsymbol{\omega_{Gimbal}}$	rad/s
	Gimbal orientational timestamp	t_{Gimbal}	
Object detection	Detected objects' labels		
	Detected objects' FPs	$\boldsymbol{u_l}$	
	Object detection timestamp	$t_{Object\ Detection}$	

Table 4.1: Input of the Real-Time Object Localizer.

The input of the RTL, which is listed in Table 4.1, can be split into three categories. Firstly, there is the UAV flight data input. This input contains the UAV's position $\boldsymbol{c_{UAV}} \in \mathbb{R}^3$ in a three-dimensional Euclidean coordinate system, which will be referred to as the UAV Local NED coordinate system, its movement speed $\boldsymbol{v_{UAV}} \in \mathbb{R}^3$ in the same system as well as a timestamp $t_{UAV_{Position}}$ specifying the point in time at which this data was valid. The UAV flight data input also consists of the orientation of the UAV $\boldsymbol{\theta_{UAV}} \in \mathbb{R}^3$ as roll, pitch and yaw angles in respect to the NED system at the UAV's position, as described in Chapter 2.4.2, paired with its angular speed $\boldsymbol{\omega_{UAV}} \in \mathbb{R}^3$ in respect to the same NED coordinate system and, again, a timestamp $t_{UAV_{Orientation}}$ corresponding to this orientation information. Lastly, the origin

$$\boldsymbol{o} = \left(latitute_{Origin}, longitude_{Origin}, altitude_{Origin}\right)^T \in \mathbb{R}^3$$

of the UAV Local NED system in GNSS coordinates is also included in the UAV flight data input. The second input type of the RTL is the gimbal orientation input. This input is comprised of the orientation $\boldsymbol{\theta_{Gimbal}} \in \mathbb{R}^3$ of the gimbal, again as roll, pitch and yaw angles, its angular speed $\boldsymbol{\omega_{Gimbal}} \in \mathbb{R}^3$ and a timestamp t_{Gimbal} indicating the time at which this data was valid. Lastly, the object detection input of the RTL includes

information about all detected objects in the camera image. This information consists of the objects' label as well as their Fundamental Points $\boldsymbol{u_l} \in \mathbb{R}^2$, as elaborated in Chapter 4.1. Again, it also contains a timestamp. This timestamp $t_{Object\ Detection}$ represents the time at which the image has been captured.

It should be noted that, currently, there exists no official release of the FLC's firmware, namely ArduCopter [15], which defines a constant UAV Local NED coordinate system. Instead, ArduCopter uses the so-called home position as the point of origin for that system. This home position is set to the position of the UAV every time it is being armed. However, it may also be changed midflight which results in different UAV Local NED coordinate systems being used in the same flight session. Consequently, invalid input data is being provided in that case. Nevertheless, the ArduCopter development team has already made effort to change the UAV Local NED coordinate system to be defined only once while booting, i.e. statically. These changes are expected to be integrated in a stable release in the near future. That being said, because the gathering of valid input is out-of-scope of the RTL, the RTL's input is assumed to be valid at all times.

4.2.2 Constants

The Real-Time Object Localizer uses a variety of constants which have to be provided by the user before it is being started. While some of them control the accuracy and performance of the RTL, others are directly used in calculations. In addition to this, there also exist constant values which influence the RTL's control flow on a higher level. Because of the high number of constants that exist, the particular constants used in a step of the RTL are elaborated in the corresponding chapter of that step.

4.2.3 Output

Category	*Data*	*Units*
Localized objects	Localized objects' labels	
	Localized objects' positions	m
	Localized objects' principle axes (1)	m
	Localized objects' distances to UAV	m
	Flags whether objects were localized	
	Flags whether objects' localizations failed	
Localized object evaluations (2)	Localized objects' distances to GT position	m
	Localized objects' overlap between Q' and Q (1)	
	Localized objects' overlap between $\overline{Q'}$ and Q (1)	
Localized object pairs	Localized object pairs' object labels	
	Distance between object pairs' objects	m

Table 4.2: Output of the Real-Time Object Localizer. (1) denotes Output Data which is only generated if the Ellipsoid Approximation Localization Approach is used by the Real-Time Object Localizer. Output marked with (2) is only generated if Ground-Truth Evaluation is desired.

Table 4.2 illustrates the output data of the RTL. Again, it can be split into three categories. Note that the exact format of the output generated by the RTL depends on its configuration, i.e. some of its constants. The first category of the RTL's output is the localized objects output. This output data describes information about each of the localized objects. In general, for each object the object's label, its position in the UAV Local NED system as well as the L_2 distance to the UAV is being generated. If the RTL uses the ellipsoid approximation localization approach, the localized objects output data also contains the principle axes of the approximated ellipsoid in the UAV Local NED system. Regardless of the localization approach used, a Boolean flag indicating whether the object has been localized with the provided input data as well as a second Boolean flag being signaled if the object could have been localized using the input but the localization algorithm failed to produce valid results are also provided.

The localized object evaluations output data is the second output category of the RTL. This output contains some ground-truth evaluation performed on the localized objects and it is only generated if desired, i.e. if a specific constant is set and if ground-truth information about the localized objects is provided via constants. For each localized object for which ground-truth data is present, this output consists of the L_2 distance between the object's position and the ground-truth object position. If the ellipsoid approximation approach is being used, the overlap between the approximated ellipsoid Q' and the ground-truth ellipsoid Q as well as the overlap between the translated

approximated ellipsoid $\overline{Q'}$ and Q are also components of this output. Here, the overlap between two ellipsoids $Q_1$and Q_2 is defined as

$$o(Q_1, Q_2) \coloneqq \frac{V(Q_1 \cap Q_2)}{V(Q_1 \cup Q_2)} \tag{4.1}$$

with $V(*)$ as the volume. The translated ellipsoid $\overline{Q'}$ is Q' which has been translated so it is centered at the ground-truth center, i.e. $\boldsymbol{c_Q}$. Throughout this thesis, the overlap between $\overline{Q'}$ and Q may also be referred to as the translated overlap.

Finally, the localized object pairs output expresses information about all pairs of localized objects. In particular, this output consists of the two object labels corresponding to each object pair as well as the L_2 distances between the objects' positions.

4.3 Input Processing

The very first step performed by the Real-Time Object Localizer is the Input Processing. Here, some simple processing is applied on the provided input data. While the Input Processing is an optional step, evaluations have been performed validating that it increases the overall accuracy of the object localization. Furthermore, it leads to nearly no performance overhead. For these reasons, the Input Processing step, elaborated in this chapter, is a fixed component of the RTL's pipeline.

In particular, the Input Processing approximates input data being valid at the exact same timestamp. For this, the speed input data, i.e. the movement speed of the UAV $\boldsymbol{v_{UAV}}$, its angular orientation speed $\boldsymbol{\omega_{UAV}}$ and the gimbal's angular speed $\boldsymbol{\omega_{Gimbal}}$, is being used. As there is no speed data of the captured image, all input data is approximated at the timestamp the image has been captured at, i.e. $t_{Object\ Detection}$. The time differences in seconds between the corresponding input data's timestamp and $t_{Object\ Detection}$ are denoted as $\Delta t_{UAV_{Position}}, \Delta t_{UAV_{Orientation}}, \Delta t_{Gimbal} \in \mathbb{R}$ for the UAV positional data, its orientational input data and the gimbal's orientational data respectively. As the time differences Δt can be assumed to be close to zero, linear models may be used to approximate $\boldsymbol{c'_{UAV}}$, $\boldsymbol{\theta'_{UAV}}$ and $\boldsymbol{\theta'_{Gimbal}}$, each at the image's timestamp. It yields

$$\boldsymbol{c'_{UAV}} \coloneqq \boldsymbol{c_{UAV}} + \Delta t_{UAV_{Position}} \boldsymbol{v_{UAV}},$$
$$\boldsymbol{\theta'_{UAV}} \coloneqq \boldsymbol{\theta_{UAV}} + \Delta t_{UAV_{Orientation}} \boldsymbol{\omega_{UAV}}$$

and

$$\boldsymbol{\theta'_{Gimbal}} \coloneqq \boldsymbol{\theta_{Gimbal}} + \Delta t_{Gimbal} \boldsymbol{\omega_{Gimbal}}.$$

This newly approximated input data, alongside the unmodified input data provided, is used as the input of the following steps of the RTL. There, the approximated $\boldsymbol{c'_{UAV}}$, $\boldsymbol{\theta'_{UAV}}$

and $\boldsymbol{\theta'_{Gimbal}}$ are, for simplicity reasons, referred to as $\boldsymbol{c_{UAV}}$, $\boldsymbol{\theta_{UAV}}$ and $\boldsymbol{\theta_{Gimbal}}$ respectively.

4.4 Construction of the Camera

In this chapter, the calculation of all camera parameters corresponding to the provided input is elaborated. These parameters consist of the camera matrix, the inverse camera matrix as well as the camera's position in the UAV Local NED coordinate system coordinate system. The construction of these parameters is necessary because they are not given in the input itself but required for the object localization algorithms discussed later on in Chapter 4.6.

4.4.1 Coordinate Systems

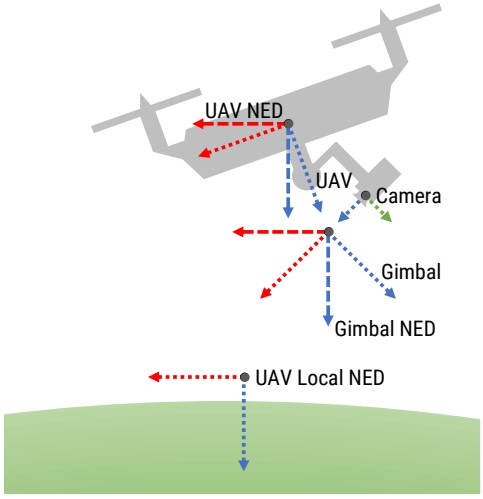


Figure 4.3: Coordinate Systems used for the Camera Construction.

For the construction of the camera's parameters, the so-called Camera coordinate system is required. This is a three-dimensional Euclidean coordinate system. As it is not provided by the input directly, it must be constructed using this input. Additionally, other coordinate systems are introduced to ease the computation of the Camera coordinate systems. All of these systems can be seen in Figure 4.3. There, the x-axis is denoted as a red arrow, the y-axis is colored green and the z-axis is blue. Furthermore, north is located to the left of the figure. In the following, each of the coordinate systems are described and transformations between them are provided. There, a Euclidean transformation matrix $T_A^B \in \mathbb{R}^{4\times4}$ denotes the transformation from coordinate system A to coordinate system B. Furthermore, the units of the following coordinate systems are meters and all of them are right-handed coordinate systems.

The first coordinate system is the UAV Local NED system which is a Euclidean three-dimensional coordinate system in which the UAV's position is provided. This system is used as the world coordinate system. Its origin may be placed arbitrarily but it is usually defined at the position where the FLC has been started at. Additionally, because it is an NED system, its x-axis shows in the north direction, the y-axis points to the east and the z-axis is perpendicular to both axis and shows to the center of the earth.

The next coordinate system used is the so-called UAV NED coordinate system. This system is also oriented as an NED frame. However, its origin is placed at the position of the UAV. Therefore, its axes may not match the axes of the UAV Local NED system. To calculate the transformation between the UAV Local NED coordinate system and the UAV NED system, this has to be taken into account. For this, the center of the earth $\boldsymbol{c_{earth}}$ as well as the north direction vector from the earth's center $\boldsymbol{d_{north}}$, each in the UAV Local NED system, must be determined. The calculation of $\boldsymbol{c_{earth}}$ is trivial. By definition, the z-axis of the UAV Local NED system points to the center of the earth. Therefore, $\boldsymbol{c_{earth}}$ is determined as

$$\boldsymbol{c_{earth}} \coloneqq \begin{pmatrix} 0 \\ 0 \\ r_{earth} + altitude_{Origin} \end{pmatrix}$$

with r_{earth} as the radius of the earth in meters and $altitude_{Origin}$ representing the altitude of the origin above the mean sea level given via $\boldsymbol{o}$ of the UAV flight data input. The north direction can also be calculated in a simple manner. For this, the vector pointing to the center of the earth in the UAV Local NED coordinate system, referred to as $\boldsymbol{v_{earth}}$, is rotated around the y-axis, i.e. the east direction. The angle of rotation is the origin's latitude plus 90°. It follows

$$\widetilde{\boldsymbol{d_{north}}} \coloneqq \widetilde{R_{y,\alpha}}\widetilde{\boldsymbol{v_{earth}}} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ r_{earth} + altitude_{Origin} \\ 0 \end{pmatrix}$$

whereby $\widetilde{R_{y,\alpha}}$ is the 4×4 rotation matrix to rotate around the y-axis using the angle $\alpha \coloneqq latitute_{Origin} + 90°$. The length of $\boldsymbol{d_{north}}$ is irrelevant as long as it points in the north direction. Given $\boldsymbol{c_{earth}}$ and $\boldsymbol{d_{north}}$, the transformation matrix $T^{UAV\_NED}_{UAV\_Local\_NED}$ can be determined as

$$T^{UAV\_NED}_{UAV\_Local\_NED} \coloneqq \begin{bmatrix} \boldsymbol{r_x} & \boldsymbol{r_y} & \boldsymbol{r_z} & \boldsymbol{c_{UAV}} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}. \tag{4.2}$$

Here, $\boldsymbol{c_{UAV}}$ is the translation part of the inverse transformation matrix which is simply the UAV's position in the UAV Local NED coordinate system. $\boldsymbol{r_z}$ denotes the transformation of the z-axis from the UAV NED system to the UAV Local NED system. Because it points to the center of the earth, it is determined via

$$\boldsymbol{r_z} \coloneqq \frac{1}{\|\boldsymbol{c_{earth}} - \boldsymbol{c_{UAV}}\|} (\boldsymbol{c_{earth}} - \boldsymbol{c_{UAV}}).$$

Next, the $\boldsymbol{r_y}$ vector is defined to point eastwards. Therefore, it is perpendicular to the z-axis as well as the north direction vector $\boldsymbol{d_{north}}$, i.e.

$$\boldsymbol{r_y} \coloneqq \frac{1}{\|\boldsymbol{r_z} \times \boldsymbol{d_{north}}\|} (\boldsymbol{r_z} \times \boldsymbol{d_{north}}).$$

Lastly, $\boldsymbol{r_x}$ is simple determined as the cross-product between $\boldsymbol{r_y}$ and $\boldsymbol{r_z}$, i.e. $\boldsymbol{r_x} \coloneqq \boldsymbol{r_y} \times \boldsymbol{r_z}$. It should be noted that the matrix inversion in (4.2) can be performed efficiently using (2.1) as it is a Euclidean transformation matrix.

The UAV coordinate system has the same origin as the UAV NED system. However, it is oriented so their axes match the principle axes of the UAV as described in Chapter 2.4.2. Therefore, it is determined as

$$T_{UAV\_Local\_NED}^{UAV} \coloneqq T_{UAV\_NED}^{UAV} T_{UAV\_Local\_NED}^{UAV\_NED},$$

whereby $T_{UAV\_NED}^{UAV}$ is the extension of the 3×3 rotation matrix of Chapter 2.4.2 to a 4×4 rotation matrix, i.e. by adding non-diagonal elements of value zero and a diagonal element of value one. The rotation angles $\boldsymbol{\theta_{UAV}}$, i.e. the roll, pitch and yaw angles, are provided as input.

Next, the Gimbal NED coordinate system is located at the point of rotation of the gimbal on which the camera is mounted on. Because this point is unequal to the UAV's position most of the time, an offset is required between the UAV's position and the gimbal's origin. This is realized as the constant vector $\boldsymbol{t_{UAV}^{Gimbal}} \in \mathbb{R}^3$ which denotes the position of the gimbal's point of rotation in the UAV coordinate system. The transformation matrix $T_{UAV\_Local\_NED}^{Gimbal\_NED}$ is obtained in a similar fashion as in (4.2). However, a different translation component is being used for the inverse transformation matrix. This translation $\boldsymbol{c_{Gimbal\_NED}}$ is the position of the gimbal's point of rotation in the UAV Local NED system. It is obtained via

$$\boldsymbol{c_{\widetilde{Gimbal\_NED}}} \coloneqq {T_{UAV\_Local\_NED}^{UAV}}^{-1} \boldsymbol{t_{UAV}^{\widetilde{Gimbal}}}.$$

The Gimbal coordinate system reorients the Gimbal NED coordinate system so its axes are aligned with the principle axes of the gimbal. As seen before, its transformation matrix is therefore defined as

$$T_{UAV\_Local\_NED}^{Gimbal} \coloneqq T_{Gimbal\_NED}^{Gimbal} T_{UAV\_Local\_NED}^{Gimbal\_NED}.$$

Again, $T_{Gimbal\_NED}^{Gimbal}$ is the 4×4 rotation matrix using the roll, pitch and yaw angles of the gimbal, i.e. $\boldsymbol{\theta_{Gimbal}}$.

Finally, we arrive at the Camera coordinate system. As the camera may be mounted at an arbitrary position on the gimbal, another constant offset vector $\boldsymbol{t}_{\boldsymbol{Gimbal}}^{\boldsymbol{Camera}} \in \mathbb{R}^3$ is being used. This vector is the camera's position in the Gimbal coordinate system. The orientation of the Camera coordinate system, however, is fundamentally different from the other systems. Here, the x-axis points to the right of the camera matching the direction of the x-axis in captured images. Furthermore, the y-axis points downwards to also match the y-axis of images. Lastly, z points in the view direction of the camera to achieve a right-handed system. The transformation from the UAV Local NED system to the Camera system is, therefore, derived as

$$T_{UAV\_Local\_NED}^{Camera} := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\boldsymbol{t}_{\boldsymbol{Gimbal}}^{\boldsymbol{Camera}} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} T_{UAV\_Local\_NED}^{Gimbal}.$$

4.4.2 Camera Parameters

Given $T_{UAV\_Local\_NED}^{Camera}$, the camera parameters can be calculated easily. The first camera parameter is the camera matrix $P \in \mathbb{R}^{3\times4}$. It is an essential component of most localization algorithms. As described in Chapter 2.2, it describes the projection of any three-dimensional point $\boldsymbol{x} \in \mathbb{R}^3$ onto the image plane of the corresponding camera. It consists of the camera calibration matrix $K \in \mathbb{R}^{3\times3}$ as well as the Euclidean transformation matrix $E \in \mathbb{R}^{3\times4}$, i.e. $P := KE$. Because K is a matrix which does not change over time, it is provided as a constant value. E, on the other hand, depends on the input provided. That being said, it is simply the upper 3×4 submatrix of $T_{UAV\_Local\_NED}^{Camera}$. The second camera parameter is the inverse camera matrix $\tilde{P}^{-1}$ as described in Chapter 2.2. This matrix may be used to determine all three-dimensional points $\boldsymbol{x} \in \mathbb{R}^3$ which project onto a given image point $\boldsymbol{u} \in \mathbb{R}^2$. To calculate $\tilde{P}^{-1}$, the camera matrix has to be extended to a 4×4 matrix resulting in $\tilde{P}$ and inverted afterwards. Lastly, the position of the camera $\boldsymbol{c}_{\boldsymbol{camera}} \in \mathbb{R}^3$ may also be required. This vector is simply given by the first three values of the fourth column of the inverse camera matrix $\tilde{P}^{-1}$.

4.5 Input Filtering

All object localization algorithms elaborated in Chapter 4.6 are multi-view localization algorithm. Therefore, they require data from multiple viewpoints to localize objects. However, the input of only one viewpoint is provided at a time. For this reason, past inputs have to be stored in a database. More precisely, so-called Input Frames are generated and then stored in the Input Frame database. While these Input Frames may consist of the input data and their corresponding camera parameters in their

simplest form, more advanced structures may be used to increase the performance of further steps. For example, data can be pre-processed for the corresponding object localization algorithm used. Obviously, one cannot use all past Input Frames for the object localization as this leads to an infinitely increasing cost, both in terms of memory and performance. Therefore, the size of the database must be finite. Furthermore, an adequate selection process shall be used to select a set of Input Frames from the database. The algorithm which manages the database and also makes an Input Frame selection is hereinafter referred to as an Input Filtering algorithm. The following two chapters 4.5.1 and 4.5.2 elaborate two concrete Input Filtering algorithms.

4.5.1 Linear Input Filter

The most naïve Input Filtering algorithm is the so-called Linear Input Filter. This algorithm simply stores and selects the most recent Input Frames. The exact number of Input Frames stored in the database may be chosen arbitrarily. Here, the database size is provided via a constant. It should be mentioned, however, that this size must not be below the minimum number of Input Frames required by the object localization algorithm used. More specifically, while the triangulation approach, presented in Chapter 4.6.1, only requires two Input Frames, the ellipsoid approximation, elaborated in Chapter 4.6.2, needs at least three Input Frames to localize objects.

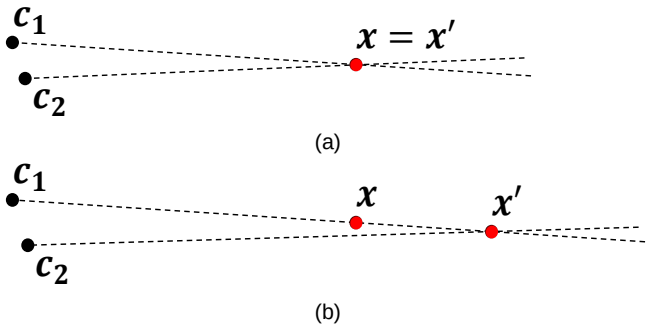


Figure 4.4: Influence of Noise on the Object Localization Result.

The advantages of the Linear Input Filter are clear. It leads to nearly no overhead and is easy to implement. However, it suffers from one crucial issue. When using consecutive Input Frames, the chances are high that they were generated from camera positions that are close to one another. Furthermore, if the camera is not close to one of the objects to localize, the direction vectors between the camera and that object are also similar between Input Frames. Here, such a direction vector is defined as the normalized vector pointing from the camera to the center of the object, as detected by its Fundamental Points. Furthermore, two direction vectors are more similar the lower

the distance between these vectors. More specifically, the similarity $\mathrm{sim}(\boldsymbol{d_1}, \boldsymbol{d_2})$ between two direction vectors is defined as

$$\mathrm{sim}(\boldsymbol{d_1}, \boldsymbol{d_2}) \coloneqq -\min\{\|\boldsymbol{d_1} - \boldsymbol{d_2}\|, \|\boldsymbol{d_1} + \boldsymbol{d_2}\|\}.$$

In a noiseless environment, the similarity between two direction vectors would not yield an issue. However, because all of the input data must be assumed to be afflicted by some kind of noise, this similarity between the direction vectors is problematic. More specifically, the influence of noise is amplified drastically if the direction vectors are similar to one another which may result in tremendously inaccurate object localizations. This issue can be seen in Figure 4.4 where two two-dimensional triangulations are visualized, each using two consecutive Input Frames. In Figure 4.4(a) noiseless Input Frames have been provided. There, the triangulation result $\boldsymbol{x'}$ perfectly matches the ground-truth object position $\boldsymbol{x}$. In Figure 4.4(b), however, the data of the second Input Frame is afflicted by very low noise. Even though the camera position $\boldsymbol{c_2}$ and the direction vector $\boldsymbol{d_2}$ are only slightly different from their ground-truth data, the triangulation result $\boldsymbol{x'}$ is grossly inaccurate in respect to the ground-truth object position $\boldsymbol{x}$. This principle also holds true for the ellipsoid approximation. Therefore, a more complex Input Filtering algorithm is more desirable when dealing with noise afflicted input data.

4.5.2 Spherical Input Filter

The observation from the previous chapter that high similarity between direction vectors of Input Frames amplifies the influence of noise on the localization result can be generalized. In general, the lower the similarity between the direction vectors of Input Frames, the less influence does input data noise yield on the localization result. Therefore, an Input Filtering algorithm that selects Input Frames with a low similarity is more feasible. One simple algorithm that attempts to achieve this is the so-called Spherical Input Filter elaborated in this chapter.

As an initial step, the Spherical Input Filter generates a distribution of points $\boldsymbol{r_l} \in \mathbb{R}^3$ with $l \in \{1, \dots, n\}$ on a sphere so the minimum distance between any two points is as high as possible. The problem of generating a distribution of points on a sphere so the minimum distance between any two points is maximized has been heavily researched on [45], [46]. However, except for a subset of cases, this problem has not been solved yet. For this reason, an approximation of the point distribution may be used. The exact algorithm for this distribution can be chosen arbitrarily. That being said, the so-called Fibonacci Spiral Sphere [47], [48] is being used in this thesis.

The Fibonacci Spiral Sphere algorithm utilizes one essential property of the Fibonacci sequence, the golden ratio φ. This ratio is, using the Fibonacci sequence $(F_i)_{i\geq 0}$, defined as

$$\varphi \coloneqq \lim_{i\to\infty}\left(\frac{F_{i+1}}{F_i}\right) = \frac{1+\sqrt{5}}{2} \approx 1.618.$$

This means that the ratio between consecutive Fibonacci numbers approaches the golden ratio φ, More specifically, the so-called golden angle

$$\vartheta \coloneqq 2\pi(2-\varphi) \approx 2.4\text{rad}$$

is utilized to generate any number of points on the unit sphere. As the number of points can be chosen arbitrarily, a constant value is used for this. The Fibonacci Spiral Sphere algorithm circulates the unit sphere in a spiral from $latitude = -\pi/2$ to $latitude = \pi/2$. At any point, the radius of the spiral is proportional to the cosine of the latitude. Further, the continuous distribution function of the turns is proportional to $\sin(latitude) + 1$. Given this, the latitude of point $\boldsymbol{r_l}$ to be distributed on the sphere can be calculated as

$$latitude_l \coloneqq \sin^{-1}\left(l\frac{2}{N+1} - 1\right) \quad \forall l \in \{1, \dots, n\}.$$

Furthermore, the longitude of $\boldsymbol{r_i}$ is given as

$$longitude_l \coloneqq l\vartheta \quad \forall l \in \{1, \dots, n\}.$$

Therefore, the points $\boldsymbol{r_l} \in \mathbb{R}^3$ can be determined via

$$\boldsymbol{r_l} \coloneqq \begin{pmatrix} \cos(longitude_l)\cos(latitude_l) \\ \sin(longitude_l)\cos(latitude_l) \\ \sin(latitude_l) \end{pmatrix} \quad \forall l \in \{1, \dots, n\}$$

resulting in a sufficiently good approximation for the distribution problem.

Given the points $\boldsymbol{r_l}$, the surface of the unit sphere can be partitioned into so-called regions R_l. Here, a region R_l is defined as the set of all points on the sphere which have, out of all points $\boldsymbol{r_j}$, minimal distance to $\boldsymbol{r_l}$, i.e.

$$R_l \coloneqq \left\{\boldsymbol{x} \in \mathbb{R}^3 : \|\boldsymbol{x}\| = 1 \wedge l = \operatorname*{argmin}_j \left\|\boldsymbol{x} - \boldsymbol{r_j}\right\|\right\} \quad \forall l \in \{1, \dots, n\}.$$

Assuming that the points $\boldsymbol{r_l}$ are nearly evenly distributed on the unit sphere, these regions R_l have approximately the same size. The Spherical Input Filter now creates the Input Frame database with the capacity of the number of regions, i.e. the number of points distributed. Further, each of these entries is associated to exactly one region R_l. When being provided with an Input Frame, the Spherical Input Filter assigns the entry of the database which corresponds to the region R_l in which the direction vector $\boldsymbol{d}$ of the Input Frame is contained, i.e. $\boldsymbol{d} \in R_l$, with the provided Input Frame. Afterwards, it selects all Input Frames of the database that have been set before and forwards them to the object localization.

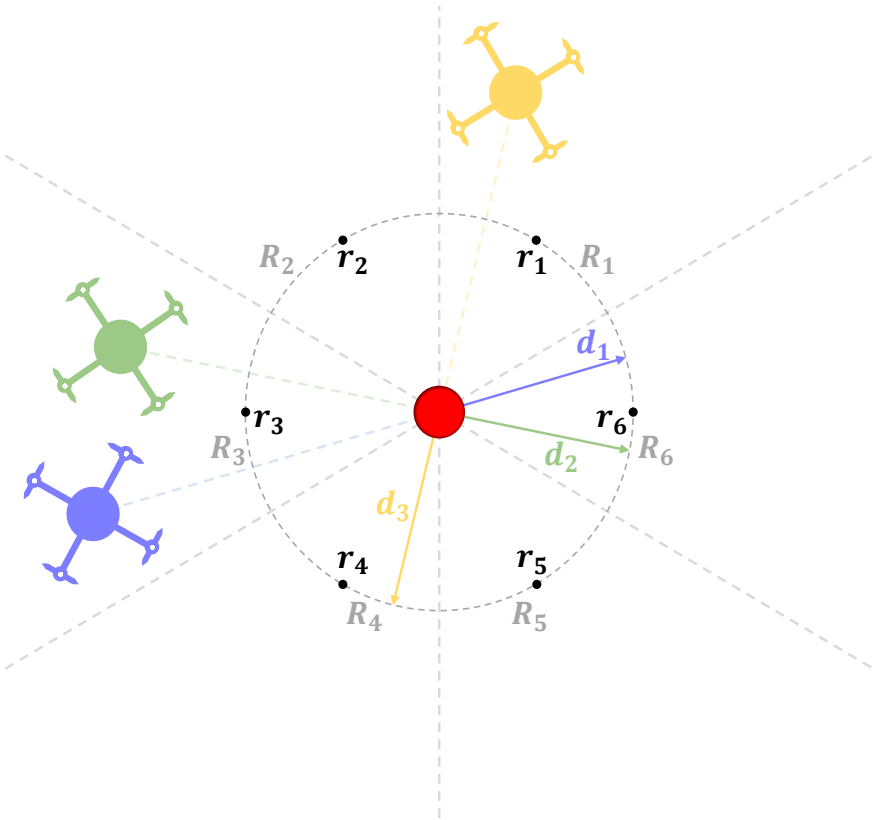


Figure 4.5: Top-Down View of Two-Dimensional Spherical Input Filter with Six Regions and Three Inputs.

A simplification of the Spherical Input Filter for the two-dimensions case is illustrated in Figure 4.5. There, the object to localize is visualized as a red sphere and the Spherical Input Filter generated six points $\boldsymbol{r_1}, \dots, \boldsymbol{r_6}$ on the unit circle from which the six regions $\boldsymbol{R_1}, \dots, \boldsymbol{R_6}$ were derived. The RTL then is provided with three inputs for which the corresponding UAV is colored blue, green and yellow in the figure respectively. This is also the order in which the inputs have been given. As the first direction vector $\boldsymbol{d_1}$ is element of the region R_6, the corresponding input data is stored in the Input Frame associated with R_6. The direction vector d_2 of the second input, however, also lies in R_6 and, therefore, the data of the same Input Frame is overwritten with the second input. As a result, there is only one Input Frame in the Spherical Input Filter's database which has been set yet because the view direction vectors $\boldsymbol{d_1}$ and $\boldsymbol{d_2}$ are similar and lie in the same region. The red object can, therefore, not be localized with either of the object localization approaches. For the third input, the direction vector $\boldsymbol{d_3}$ lies in a different region being R_4. Starting from this input, the triangulation approach can be used to localize the object. To use the ellipsoid approximation approach, at least one further input has to be provided for which its corresponding direction vector $\boldsymbol{d_l}$ is element of either of the regions R_1,R_2,R_3 or R_5.

The Spherical Input Filter tackles the issue of the Linear Input Filter that similar direction vectors $\boldsymbol{d_l}$ amplify the influence of noise on the localization result. It does so by building a database of Input Frames observing the object from multiple angles. Therefore, past Input Frames are being stored as long as there has not been an Input Frame observing the object from the same region. If the number of regions is chosen wisely, this technique improves the localization result immensely. If the number of regions, however, is below a certain threshold, then the direction vectors $\boldsymbol{d_l}$ may only be associated with a low number of regions even though they are changing drastically throughout the flight. Obviously, utilizing more Input Frames yields, in general, better object localization results. On the other hand, if too many regions are defined, we may risk allowing Input Frames with similar direction vectors $\boldsymbol{d_l}$ to be contained in different regions. This results in the same issue as the Linear Input Filter. That being said, this issue is no longer relevant if the direction vectors $\boldsymbol{d_l}$ vary enough resulting in a sufficient number of entries of the database being filled with Input Frame data.

Nevertheless, there are several disadvantages of the Spherical Input Filter. First and foremost, it introduces additional overhead, both in terms of memory and performance. In terms of memory, the Spherical Input Filter must be able to associate entries of the Input Frame database with their corresponding regions. It does so by storing the region's point $\boldsymbol{r_l}$. Furthermore, as the entry of the Input Frame database, which is to be set to a provided Input Frame, is selected using the view direction from the camera to the observed object, one cannot use a single Spherical Input Filter for all objects that are to be localized. Instead, there must be one Input Filter for each detected object. The overhead in terms of performance can be split into two aspects, the initialization of the regions R_l and the selection of the entry in the database corresponding to a provided Input Frame. As the initialization of the regions may only be done, this overhead is negligible. This is especially true if an efficient point generation algorithm is being used, e.g. the Fibonacci Spiral Sphere algorithm. On the other hand, we have the overhead of selecting the corresponding entry of the Input Frame database. This overhead is, however, also low as a simple loop which checks for the minimal distance between the region's point and the view direction is sufficient. Another disadvantage of the Spherical Input Filter is that, when using a wisely chosen number of regions, the number of Input Frames stored in the database might become too high for the object localization algorithms which leads to a high run-time of the object localization. This, however, can also be dealt with efficiently by simply limiting the number of Input Frames that are selected by the Spherical Input Filter. This limit is referred to as m and is realized as a constant. The simplest way of achieving such a limit is by only selecting the Input Frames of the first m entries of the database, in terms of their indices, that

have Input Frames stored. Depending on the algorithm used to generate the sphere points $\boldsymbol{r_l}$, however, a more complex approach might be more feasible. For example, when using the Fibonacci Spiral Sphere for the point generation, the minimal distance between consecutive points $\boldsymbol{r_l}$ approaches the theoretical minimum. Therefore, instead of selecting consecutive points, one can select the Input Frame entries of regions R_l that have a maximal minimum distance between their indices. While this is not a perfect selection algorithm, it is efficient and it guarantees not to perform worse than the naïve selection algorithm.

4.6 Object Localization

This chapter elaborates the integrated object localization algorithms of the RTL in detail. In particular, two approaches have been considered. While the triangulation approach, described in Chapter 4.6.1, operates on a single Fundamental Point for each object, the ellipsoid approximation, presented in Chapter 4.6.2, expects four Fundamental Points around each object.

Consequently, only two of the three FPD configurations have been considered. For the third case in which two Fundamental Points have been provided, as seen in Figure 4.2(b), no adequate localization approach has been discussed in literature. However, a simple localization algorithm for this Fundamental Point configuration can be developed easily. This algorithm constructs a subset of a plane in the three-dimensional Euclidean space for each pair of Fundamental Points provided in the input. These planes contain the camera's position as well as one reprojection per Fundamental Point. Therefore, they are uniquely determined. The utilized subset of a plane is defined as the set of all points which lie on any of the two rays, each containing the camera's position as well as a reprojection of their corresponding Fundamental Point, or in between them. Consequently, the subset can be described as a triangle of "infinite size", i.e. a triangle for which one of the edges has an infinitely high distance to the opposite corner. In the following, the subsets corresponding to an object are referred to as T_l with $l \in \{1, \dots, n\}$. The result of the object localization, i.e. the position of the object, can now be determined as the point $\boldsymbol{x'} \in \mathbb{R}^3$ which is closest on average to all subsets. This yields

$$\boldsymbol{x'} \coloneqq \underset{\boldsymbol{x'}}{\operatorname{argmin}} \sum_{l=1}^{n} \mathrm{d}(\boldsymbol{x'}, T_l)^2,$$

whereby the distance between the point $\boldsymbol{x'}$ and the subset T_l is denoted as $\mathrm{d}(*,*)$. Note especially the similarity between this approach and the Generalized Midpoint Triangulation introduced in Chapter 3.2.3. Because of this similarity and also because no other algorithm for this scenario has been discussed, the configuration in which only

two Fundamental Points are provided by the FPD has been discarded and no localization algorithm for this configuration has been integrated into the RTL.

4.6.1 Triangulation

In this chapter, the triangulation approach for which the FPD provides exactly one Fundamental Point per object is elaborated in detail. For this, three triangulation algorithms, whereby two of which have been presented in Chapter 3.2, are integrated. As a reminder, when using triangulation, a point $\boldsymbol{x} \in \mathbb{R}^3$ is being reconstructed which has been captured from different viewpoints as image points $\boldsymbol{u_1}, \dots, \boldsymbol{u_n} \in \mathbb{R}^2$. As the Fundamental Points of an object are placed at the object's center in their corresponding image, the triangulation result $\boldsymbol{x'} \in \mathbb{R}^3$ is the object's center in the three-dimensional Euclidean space. No further object information is being determined using the triangulation approach. In particular, the object's shape is not approximated when using triangulation. The integrated triangulation algorithms are the Generalized Midpoint Triangulation, elaborated in Chapter 3.2.3, the L_∞ Triangulation, which has been presented in Chapter 3.2.5, as well as a novel triangulation algorithm referred to as the Hybrid Triangulation. All other triangulation algorithms presented in Chapter 3.2 have crucial disadvantages when applied to the APOLI project. In particular, the Noiseless Triangulation is not feasible because it assumes that the input data is completely noiseless. This assumption cannot be made because the input data is gathered using sensors which are themselves afflicted by noise. Furthermore, the Midpoint Triangulation is not integrated because it only uses input data of two Input Frames. Therefore, noise in one of these Input Frames may have a great influence on the result. Also, the Generalized Midpoint Triangulation is able to deal with noise afflicted input data in a better way while following the same concept as the Midpoint Triangulation. Lastly, while the L_2 Triangulation is able to find the theoretical optimal solution for $\boldsymbol{x'}$ if only the image points are assumed to be noise afflicted, a non-trivial problem must be solved to determine the triangulation result which is not feasible for real-time capability.

4.6.1.1 Generalized Midpoint Triangulation

The first integrated triangulation algorithm is the Generalized Midpoint Triangulation which has been presented in Chapter 3.2.3. This algorithm determines the point $\boldsymbol{x'}$ which is closest on average to lines being the extensions of the rays reprojected via the captured image points, i.e. the Fundamental Points. For this algorithm, a closed-form solution exists. This solution is provided via (3.10), i.e.

$$\boldsymbol{x}' \coloneqq \frac{1}{n}(I + DD^T A)\sum_{i=1}^{n}\boldsymbol{c_i} - A\sum_{i=1}^{n}\langle \boldsymbol{c_i}, \boldsymbol{d_i}\rangle \boldsymbol{d_i}.$$

Here, the $\boldsymbol{c_l} \in \mathbb{R}^3$ denote the position of the camera of view l. These positions are provided as one of the components of the Input Frames as they have been determined in the Camera Construction step in Chapter 4.4. Furthermore, the direction vectors $\boldsymbol{d_l} \in \mathbb{R}^3$ of the rays are, using (3.6), given as

$$\boldsymbol{d_l} \coloneqq \frac{1}{\|\boldsymbol{v}_l - \boldsymbol{c}_l\|}(\boldsymbol{v}_l - \boldsymbol{c}_l),$$

with

$$\widetilde{\boldsymbol{v_l}} = \widetilde{P_l}^{-1}\widetilde{\boldsymbol{u_l}}$$

for any $d > 0$. Lastly, the matrices are determined via $D \coloneqq [\boldsymbol{d_1}|\ldots|\boldsymbol{d_n}] \in \mathbb{R}^{3\times n}$ and $A \coloneqq (nI - DD^T)^{-1} \in \mathbb{R}^{3\times 3}$. As only basic algebraic operations are used to determine the triangulation result $\boldsymbol{x}'$, the Generalized Midpoint Triangulation can be carried out efficiently.

4.6.1.2 L_∞ Triangulation

The second triangulation algorithm which has been implemented is the L_∞ Triangulation elaborated in Chapter 3.2.5. This triangulation approach determines the point $\boldsymbol{x}'$ which minimizes the L_∞ norm of the reprojection error, i.e.

$$\boldsymbol{x}' = \underset{\boldsymbol{x}'}{\operatorname{argmin}}\|\boldsymbol{f}(\boldsymbol{x}')\|_\infty = \underset{\boldsymbol{x}'}{\operatorname{argmin}}\left\|\begin{pmatrix}\|\boldsymbol{u_1} - \boldsymbol{u_1'}\| \\ \vdots \\ \|\boldsymbol{u_n} - \boldsymbol{u_n'}\|\end{pmatrix}\right\|_\infty,$$

whereby $\boldsymbol{f}(\boldsymbol{x}') \coloneqq (\|\boldsymbol{u_1} - \boldsymbol{u_1'}\|, \ldots, \|\boldsymbol{u_n} - \boldsymbol{u_n'}\|)^T$ and $\widetilde{\boldsymbol{u_l'}} = P_l\widetilde{\boldsymbol{x}'}$. It has been shown that for this problem, there exists a single minimum in the chirality domain. Therefore, the L_∞ Triangulation can be solved using a simple iterative algorithm, i.e. an initial $\boldsymbol{x'^{(0)}} \in \mathbb{R}^3$ in the chirality domain is determined and improved upon in every iteration step. For this, a way to compute the initial estimate $\boldsymbol{x'^{(0)}}$ is required. Here, the result of the Generalized Midpoint Triangulation may be used. While the Generalized Midpoint Triangulation suffers from projective properties of the pinhole camera, its localization result can be used as a good initial estimate $\boldsymbol{x'^{(0)}}$. The Generalized Midpoint Triangulation is also solved efficiently. In addition to this, because the Object Localization step of the RTL maintains an environment map in which all objects that have been localized yet are stored, the most recent position of the approximated object can also be used as an initial estimate for $\boldsymbol{x'^{(0)}}$. Obviously, this can only be done if the object that is to be localized has already been localized by the triangulation in the past. In this case, the initial estimate $\boldsymbol{x'^{(0)}}$ is either set to result of the Generalized Midpoint Triangulation or the approximated object position, depending on which of these points

has a lower L_∞ distance. In either case, it must be assured that the initial estimate $\boldsymbol{x'^{(0)}}$ lies in front of all camera views, i.e. in the chirality domain.

After an initial estimate $\boldsymbol{x'^{(0)}}$ has been determined, iterations are performed which improve upon it. Given a point $\boldsymbol{x'^{(m)}}$, the m-th iteration step with $m \in \{0, \dots, o-1\}$ determines a point $\boldsymbol{x'^{(m+1)}}$ in the chirality domain which has, when projected onto the image planes, a non-greater L_∞ distance to the image points $\boldsymbol{u_1}, \dots, \boldsymbol{u_n}$ than $\boldsymbol{x'^{(m)}}$. Using the notation of Chapter 3.2.5, this is equal to

$$\left\| \boldsymbol{f}\left(\boldsymbol{x'^{(m+1)}}\right) \right\|_\infty \leq \left\| \boldsymbol{f}\left(\boldsymbol{x'^{(m)}}\right) \right\|_\infty .$$

More precisely, a line search is performed in each iteration. Because there is no termination criterium for the iterations, a fixed number of iterations to perform must be defined beforehand. This is realized via a constant value o. The exact operations performed by the m-th iteration step are elaborated in detail now.

Firstly, a random line direction vector $\boldsymbol{v^{(m)}} \in \mathbb{R}^3$ of unit length has to be determined. This can be done by assigning random values in the interval $[-1; 1]$ to its three components and normalizing the vector afterwards. However, if the unnormalized vector generated this way has a length of greater than one, it must be rerandomized. If this rerandomization is not performed, the distribution of the vector $\boldsymbol{v^{(m)}}$ would not be even on the unit sphere. The resulting line $\boldsymbol{l^{(m)}}$ is, as before, denoted as

$$\boldsymbol{l^{(m)}}(t) := \boldsymbol{x'^{(m)}} + t\boldsymbol{v^{(m)}}$$

with $t \in \mathbb{R}$.

As a next step, boundaries for the line parameter t have to be determined. These will be referred to as t_1 for the lower and t_2 for the upper boundary. They should be generated in such a way the interval $[t_1; t_2]$ contains the minimum of the line $\boldsymbol{l^{(m)}}$. Here, the minimum of a line is defined as the parameter of the line point with the minimal L_∞ distance in respect to all other line points in the chirality domain. Boundaries for the line parameter are necessary because the L_∞ Triangulation only guarantees that there is exactly one line minimum in the chirality domain. Furthermore, they are required by the line search algorithm performed in the next step. t_1 and t_2 may be determined as the minimal or maximal line parameter, respectively, for which the corresponding line point still lies inside the chirality domain. This, however, results in two issues. Firstly, the chirality domain may not be limited in one or both of the line directions. Therefore, t_1 or t_2 could be $-\infty$ or ∞ respectively. Secondly, because these parameters are used in the line search of the next step, a smaller interval $[t_1; t_2]$ is of advantage in terms of

efficiency. The exact algorithm to generate the interval $[t_1; t_2]$ may be chosen arbitrarily as long as the minimum of $\boldsymbol{l}^{(\boldsymbol{m})}$ lies in $[t_1; t_2]$ and the boundaries t_1 and t_2 are finite. In the following, a simple iterative algorithm to determine these boundaries is described. This algorithm basically checks the L_∞ distance of line points of parameters with doubled distance between one another in each iteration step. Once the L_∞ distance allows for some conclusion about the minimum on the line, the boundary is set to the current parameter. In particular, the algorithm stats by setting

$$t_1^{(0)} \coloneqq 0,$$
$$t_1'^{(0)} \coloneqq 0,$$
$$p_1^{(0)} \coloneqq 1,$$
$$s_1^{(0)} \coloneqq -1.$$

In each iteration step l, the parameter t_1' is calculated as $t_1' \coloneqq t_1^{(l)} + s_1^{(l)} p_1^{(l)}$. If $\boldsymbol{l}^{(\boldsymbol{m})}(t_1')$ lies inside the chirality domain and if

$$\left\| \boldsymbol{f}\left(\boldsymbol{l}^{(\boldsymbol{m})}(t_1')\right) \right\|_\infty \geq \left\| \boldsymbol{f}\left(\boldsymbol{l}^{(\boldsymbol{m})}\left(t_1'^{(l)}\right)\right) \right\|_\infty, \tag{4.3}$$

t_1 is determined as t_1' as the line minimum cannot be smaller than t_1'. If instead $\boldsymbol{l}^{(\boldsymbol{m})}(t_1')$ lies inside the chirality domain but (4.3) does not yield true, no knowledge about the interval containing the line minimum can be concluded. In this case, we set $t_1^{(l+1)} \coloneqq t_1^{(l)}$, $t_1'^{(l+1)} \coloneqq t_1'$, $p_1^{(l+1)} \coloneqq 2p_1^{(l)}$ and $s_1^{(l+1)} \coloneqq s_1^{(l)}$. However, if $\boldsymbol{l}^{(\boldsymbol{m})}(t_1')$ does not lie inside the chirality domain anymore, the parameter increment must be decreased. This can be achieved by resetting the current power of two, i.e. p_1, and halving the scale parameter s_1. Furthermore, the initial parameter for the search is set to the most recent determined parameter that still lies inside the chirality domain. This yields $t_1^{(l+1)} \coloneqq t_1'^{(l)}$, $t_1'^{(l+1)} \coloneqq t_1'^{(l)}$, $p_1^{(l+1)} \coloneqq 1$ and $s_1^{(l+1)} \coloneqq 0.5 s_1^{(l)}$. After the lower boundary t_1 has been determined, an analogous iteration is repeated for the upper boundary. The initialization, however, is done by setting

$$t_2^{(0)} \coloneqq t_1,$$
$$t_2'^{(0)} \coloneqq t_1,$$
$$p_2^{(0)} \coloneqq 1,$$
$$s_2^{(0)} \coloneqq 1,$$

i.e. the search now begins at the lower boundary and, obviously, the search direction should now be along the positive line direction. This iterative algorithm is guaranteed to terminate at some iteration step as the L_∞ Triangulation guarantees the existence of exactly one line minimum in the chirality domain, i.e. $\left\| \boldsymbol{f}\left(\boldsymbol{l}^{(\boldsymbol{m})}(t)\right) \right\|_\infty$ with $t \in [t_1; t_2]$ is convex. Therefore, at some point (4.3) must be satisfied.

After a parameter interval $[t_1; t_2]$ for t has been determined, any feasible line search algorithm may be used to determine the minimum of $\boldsymbol{l}^{(m)}$ in that interval. Again, the convex property of $\left\| \boldsymbol{f}\left(\boldsymbol{l}^{(m)}(t)\right)\right\|_\infty$ with $t \in [t_1; t_2]$ may be used for this. In particular, one may select two parameters $\dot{t_1}, \dot{t_2} \in (t_1; t_2)$ with $\dot{t_1} < \dot{t_2}$ and calculate their corresponding L_∞ distances. Because of the convexity, the L_∞ distance is highest for one of the two boundaries, i.e. t_1 and t_2. This boundary can then be replaced by the $\dot{t_l}$ which is closest to it. Therefore, the search interval gets smaller in every iteration step. After a number of steps, the interval is sufficiently small. In this thesis, the Fibonacci Line Search [33] has been chosen as the line search algorithm of choice. This iterative algorithm follows the principle mentioned above while introducing a crucial advantage. Using the Fibonacci Fine Search, only one of the two parameters $\dot{t_1}, \dot{t_2}$ has to be recalculated in every iteration step. The other parameter can be directly adopted from the previous step, reducing the cost of the line search. The Fibonacci Line Search starts by determining the number of iteration steps necessary to guarantee an interval not greater than a given constant S. For this, the smallest value $k \in \mathbb{N}_0$ is determined which satisfies

$$F_k > \frac{t_2 - t_1}{S}. \tag{4.4}$$

Again, F_k is the k-th Fibonacci number. After the number of iteration steps has been determined, initial values are set up as

$$\begin{aligned}
\widehat{t_1}^{(0)} &:= t_1, \\
\widehat{t_2}^{(0)} &:= t_2, \\
\dot{t_1}^{(0)} &:= t_1 + \frac{F_{k-2}}{F_k}(t_2 - t_1), \\
\dot{t_2}^{(0)} &:= t_1 + \frac{F_{k-1}}{F_k}(t_2 - t_1).
\end{aligned}$$

In the l-th iteration step with $l \in \{0, \dots, k-3\}$, the L_∞ distance of the line points corresponding to the parameters $\dot{t_1}^{(l)}$ and $\dot{t_2}^{(l)}$ are determined. If

$$\left\| \boldsymbol{f}\left(\boldsymbol{l}^{(m)}\left(\dot{t_1}^{(l)}\right)\right)\right\|_\infty > \left\| \boldsymbol{f}\left(\boldsymbol{l}^{(m)}\left(\dot{t_2}^{(l)}\right)\right)\right\|_\infty, \tag{4.5}$$

then the lower boundary $\widehat{t_1}^{(l)}$ has the highest L_∞ distance out of all four parameters and can, therefore, safely be replaced by $\dot{t_1}^{(l)}$. In this case, we set $\widehat{t_1}^{(l+1)} := \dot{t_1}^{(l)}$ and $\widehat{t_2}^{(l+1)} := \widehat{t_2}^{(l)}$. In addition to this, the $\dot{t_1}$ to check the L_∞ distance for in the next iteration step is then set to $\dot{t_2}^{(l)}$, i.e. $\dot{t_1}^{(l+1)} := \dot{t_2}^{(l)}$. Because of this, a new upper parameter to check has to be calculated via

$$\dot{t_2}^{(l+1)} \coloneqq \widehat{t_1}^{(l+1)} + \frac{F_{k-l-2}}{F_{k-l-1}}\left(\widehat{t_2}^{(l+1)} - \widehat{t_1}^{(l+1)}\right).$$

If, however, (4.5) does not yield true, then the upper boundary $\widehat{t_2}^{(l)}$ is to be replaced. In this case, analogous operations are being performed, i.e.

$$\widehat{t_1}^{(l+1)} \coloneqq \widehat{t_1}^{(l)},$$
$$\widehat{t_2}^{(l+1)} \coloneqq \dot{t_2}^{(l)},$$
$$\dot{t_1}^{(l+1)} \coloneqq \widehat{t_1}^{(l+1)} + \frac{F_{k-l-3}}{F_{k-l-1}}\left(\left(\widehat{t_2}^{(l+1)} - \widehat{t_1}^{(l+1)}\right)\right),$$
$$\dot{t_2}^{(l+1)} \coloneqq \dot{t_1}^{(l)}.$$

At the end of these iterations, the size of $\left[\widehat{t_1}^{(k-2)}; \widehat{t_2}^{(k-2)}\right]$ is guaranteed to be not greater than S. Finally, we determine the minimum of the line $\boldsymbol{l}^{(m)}$ as the point which has the smallest L_∞ distance. Because the interval does not only contain a single point, the minimum must be approximated by sampling specific points. For this, the following points are being sampled:

$$\boldsymbol{x_1'} \coloneqq \boldsymbol{x'}^{(m)},$$
$$\boldsymbol{x_2'} \coloneqq \boldsymbol{l}^{(m)}\left(\widehat{t_1}^{(k-2)}\right),$$
$$\boldsymbol{x_3'} \coloneqq \boldsymbol{l}^{(m)}\left(\frac{\widehat{t_1}^{(k-2)} + \widehat{t_2}^{(k-2)}}{2}\right),$$
$$\boldsymbol{x_4'} \coloneqq \boldsymbol{l}^{(m)}\left(\widehat{t_2}^{(k-2)}\right).$$

The improved line point $\boldsymbol{x'}^{(m+1)}$ is then set to the $\boldsymbol{x_l'}$ with the smallest L_∞ distance which finalizes the m-th iteration step.

After all o iteration steps of the L_∞ Triangulation have been performed, we set the result of the triangulation to $\boldsymbol{x'} \coloneqq \boldsymbol{x'}^{(o)}$ which concludes the L_∞ Triangulation.

4.6.1.3 Hybrid Triangulation

Lastly, the so-called Hybrid Triangulation has also been integrated. This triangulation algorithm combines the Generalized Midpoint Triangulation and the L_∞ Triangulation presented in Chapter 4.6.1.1 and 4.6.1.2 respectively. More specifically, it performs both algorithms yielding $\boldsymbol{x_1'}$ as the result of the Generalized Midpoint Triangulation and $\boldsymbol{x_2'}$ as the L_∞ Triangulation's result. After $\boldsymbol{x_1'}$ and $\boldsymbol{x_2'}$ have been determined, the Hybrid Triangulation now compares the L_2 distances of both of these points. The final triangulation result $\boldsymbol{x'}$ of the Hybrid Triangulation is set to $\boldsymbol{x_1'}$ if

$$\|\boldsymbol{f}(\boldsymbol{x_1'})\| \leq \|\boldsymbol{f}(\boldsymbol{x_2'})\| \tag{4.6}$$

yields true. Otherwise, $\boldsymbol{x_2'}$ is chosen as the result of the Hybrid Triangulation. Therefore, it selects the triangulation result with the lower L_2 distance as its final result.

The Hybrid Triangulation performs two triangulation algorithms, the Generalized Midpoint Triangulation as well as the L_∞ Triangulation. The goal is to always select the triangulation result with the lower L_2 distance. Therefore, the minimization problem of the Hybrid Triangulation is equivalent to the one of the L_2 Triangulation, i.e. (3.11). However, the L_2 distance is only determined for two discrete sample points, $\boldsymbol{x_1'}$ and $\boldsymbol{x_2'}$. As a result, the Hybrid Triangulation may not select the more optimal point $\boldsymbol{x_l'}$, i.e. the point which is closer to the optimal solution $\boldsymbol{x'}$. This is because the problem of finding the point $\boldsymbol{x'} \in \mathbb{R}^3$ which minimizes the L_2 distance usually does not imply that the lower the L_2 distance of a point $\hat{\boldsymbol{x}}$, the closer $\hat{\boldsymbol{x}}$ to $\boldsymbol{x'}$. Therefore, if provided with two object localization results $\boldsymbol{x_1'}$ and $\boldsymbol{x_2'}$, the point with the lower L_2 distance does not have to be the point which is closer to the optimal solution $\boldsymbol{x'}$. This is further amplified by the fact that for this minimization problem multiple minima may exist as shown in Chapter 3.2.4. Lastly, the L_2 Triangulation only generates triangulation results $\boldsymbol{x'}$ which are more likely to be optimal if only the Fundamental Points $\boldsymbol{u_1}, \dots, \boldsymbol{u_n}$ are afflicted by noise. Again, this is not the case for most UASs and certainly not for the APOLI project. Nevertheless, this algorithm is still worth being analyzed on several datasets to evaluate its performance. On another note, it may seem as the Hybrid Triangulation introduces overhead in comparison to the L_∞ Triangulation. This is, however, not the case as the result of the Generalized Midpoint Triangulation, i.e. $\boldsymbol{x_1'}$, has been directly used as a candidate for the initial estimate of the L_∞ Triangulation. Therefore, the only overhead added by the Hybrid Triangulation is the trivial comparison (4.6).

4.6.2 Ellipsoid Approximation

The second object localization approach, which is elaborated in this chapter, is the ellipsoid approximation approach. Fundamentals of this approach have been introduced in Chapter 3.3. There, the object that is to be localized is approximated as an ellipsoid $Q \in \mathbb{R}^{4\times4}$. More specifically, an estimation of this ellipsoid, i.e. $Q' \in \mathbb{R}^{4\times4}$, is constructed as the reprojection of ellipses $C_1, \dots, C_n \in \mathbb{R}^{3\times3}$ in the image planes placed precisely inside bounding rectangles which describe the object's detection. Therefore, unlike the triangulation approach elaborated in the previous chapter, the ellipsoid Q can be used to not only determine the object's location but its shape as well. In regards to the APOLI project, the bounding rectangles can be constructed easily using four Fundamental Points around the object. The following subchapters describe the three ellipsoid approximation algorithms which have been integrated into the RTL. These algorithms are the Noiseless Ellipsoid Approximation, introduced in Chapter

3.3.1, the LfD Ellipsoid Approximation, presented in Chapter 3.3.2, as well as the LfDC Ellipsoid Approximation, described in Chapter 3.3.3.

4.6.2.1 Noiseless Ellipsoid Approximation

The Noiseless Ellipsoid Approximation is the first integrated ellipsoid approximation algorithm. It has been presented in Chapter 3.3.1. In this chapter, however, it will be applied to the more precisely formulated input data of the Object Localization step of the RTL. In particular, the bounding rectangle around an object which has been detected is axis-aligned. Let $\boldsymbol{b_l} \in \mathbb{R}^2$ be the center, $w_l \in \mathbb{R}$ the width and $h_l \in \mathbb{R}$ the height of the bounding rectangle of the l-th view. Consequently, the corresponding ellipse C_l is determined as

$$C_l \coloneqq \begin{bmatrix} \frac{1}{w_l^2} & 0 & -\frac{b_{l_1}}{w_l^2} \\ 0 & \frac{1}{h_l^2} & -\frac{b_{l_2}}{h_l^2} \\ -\frac{b_{l_1}}{w_l^2} & -\frac{b_{l_2}}{h_l^2} & \frac{b_{l_1}^2}{w_l^2} + \frac{b_{l_2}^2}{h_l^2} - \frac{1}{4} \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{4.7}$$

Using these ellipses, the vectorizations of the ellipses of the dual space, i.e. $\widehat{\boldsymbol{c_l}}$, are determined as $\widehat{\boldsymbol{c_l}} \coloneqq \mathrm{vecs}(\mathrm{adj}(C_l))$. Furthermore, for each view l a matrix G_l exists for which a closed-form solution has been formulated which only uses the view's camera matrix P_l provided as input to the Object Localization step. The resulting system (3.19) of the Noiseless Ellipsoid Approximation, i.e.

$$M\boldsymbol{w} = \boldsymbol{0} \tag{4.8}$$

with

$$M \coloneqq \begin{bmatrix} G_1 & -\widehat{\boldsymbol{c_1}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ G_2 & \boldsymbol{0} & -\widehat{\boldsymbol{c_2}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_n & \boldsymbol{0} & \boldsymbol{0} & \cdots & -\widehat{\boldsymbol{c_n}} \end{bmatrix} \in \mathbb{R}^{6n\times(10+n)},$$

$$\boldsymbol{w} \coloneqq \begin{pmatrix} \widehat{\boldsymbol{q}} \\ s_1 \\ \vdots \\ s_n \end{pmatrix} \in \mathbb{R}^{10+n},$$

can be solved via an SVD leading to the dual space ellipsoid $\widehat{Q'} \coloneqq \mathrm{vecs}^{-1}(\widehat{\boldsymbol{q}'})$. More specifically, the right singular value corresponding to the minimum singular value represents the solution of (4.8), i.e. $\boldsymbol{w}'$, which further contains $\widehat{\boldsymbol{q}'}$. As the final step, the ellipsoid Q' of the Euclidean space is determined via $Q' \coloneqq \mathrm{adj}^{-1}(\widehat{Q'})$. Subsequently, the parameters of Q' can be extracted as described in Chapter 3.3.1.

As stated in Chapter 3.3.1, the Noiseless Ellipsoid Approximation is easily influenced by input noise in a way invalid ellipsoids, i.e. quadrics which do not represent ellipsoids, are approximated. Again, to test whether Q' represents a valid ellipsoid, (3.25) may be used. Because of this, the Noiseless Ellipsoid Approximation should not be used by systems operating on noise afflicted data, e.g. the APOLI project. That being said, the Noiseless Ellipsoid Approximation is a necessary component of other ellipsoid approximation algorithms, e.g. the LfD Ellipsoid Approximation.

4.6.2.2 LfD Ellipsoid Approximation

The second ellipsoid approximation which has been integrated is the LfD Ellipsoid Approximation introduced in Chapter 3.3.2. As a reminder, this ellipsoid approximation algorithm modifies the Noiseless Ellipsoid Approximation to preprocess the input data and, as a result, decrease the influence of noise. In particular, it normalizes the ellipses C_l of each view to $\dot{C}_l$ so they are centered at the image planes' centers and their axis lengths are normalized. In addition to this, the final ellipsoid Q' is also transformed to be located around the origin $(0,0,0)^T$.

The LfD Ellipsoid Approximation begins by determining the transformation matrix T which transforms the origin of the three-dimensional Euclidean space to the center of the ellipsoid Q'. For this, the Noiseless Ellipsoid Approximation of Chapter 4.6.2.1 is performed resulting in the dual space quadric $\widehat{Q'_{NL}}$. Even though $\widehat{Q'_{NL}}$ may not represent a valid ellipsoid, the center of the quadric can still be determined via

$$\boldsymbol{c}_{\boldsymbol{Q'_{NL}}} := \frac{1}{\widehat{Q'}_{NL_{4,4}}} \begin{pmatrix} \widehat{Q'}_{NL_{1,4}} \\ \widehat{Q'}_{NL_{2,4}} \\ \widehat{Q'}_{NL_{3,4}} \end{pmatrix}.$$

Using this, the transformation matrix T is given via (3.26), i.e.

$$T := \begin{bmatrix} I & \boldsymbol{c}_{\boldsymbol{Q'_{NL}}} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}.$$

Furthermore, the affine transformation matrices H_l, which normalize the ellipses, are a necessary component of the preprocessing as well. Using $\boldsymbol{b_l} \in \mathbb{R}^2$, $w_l \in \mathbb{R}$ and $h_l \in \mathbb{R}$ as the parameter of the bounding rectangle in view l as defined before, H_l is given as

$$H_l \coloneqq \begin{bmatrix} \sqrt{\frac{w_l^2}{4} + \frac{h_l^2}{4}} & 0 & b_{l_1} \\ 0 & \sqrt{\frac{w_l^2}{4} + \frac{h_l^2}{4}} & b_{l_2} \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}.$$

The normalized ellipse $\dot{C}_l$ is, therefore, calculated via

$$\dot{C}_l \coloneqq \begin{bmatrix} \frac{\frac{w_l^2}{4} + \frac{h_l^2}{4}}{w_l^2} & 0 & 0 \\ 0 & \frac{\frac{w_l^2}{4} + \frac{h_l^2}{4}}{h_l^2} & 0 \\ 0 & 0 & -\frac{1}{4} \end{bmatrix} \in \mathbb{R}^{3\times 3}.$$

The corresponding dual space ellipse $\widehat{\dot{C}_l}$ can then be determined as usual via $\widehat{\dot{C}_l} \coloneqq \mathrm{adj}(\dot{C}_l)$. Alternatively, one can also use $\widehat{\dot{C}_l} \coloneqq H_l^{-1}\widehat{C}_l H_l^{-T}$. Again, the ellipses' vectorizations are subsequently calculated via $\widehat{\dot{\boldsymbol{c}}_{\boldsymbol{l}}} \coloneqq \mathrm{vecs}\left(\widehat{\dot{C}_l}\right)$.

Using this preprocessing, a similar system as seen in the Noiseless Ellipsoid Approximation, i.e.

$$\dot{M}\dot{\boldsymbol{w}} = \boldsymbol{0},$$

whereby

$$\dot{M} \coloneqq \begin{bmatrix} \dot{G}_1 & -\widehat{\dot{\boldsymbol{c}}_{\boldsymbol{1}}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \dot{G}_2 & \boldsymbol{0} & -\widehat{\dot{\boldsymbol{c}}_{\boldsymbol{2}}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{G}_n & \boldsymbol{0} & \boldsymbol{0} & \cdots & -\widehat{\dot{\boldsymbol{c}}_{\boldsymbol{n}}} \end{bmatrix},$$

is used to approximate the ellipsoid Q'. In particular, an SVD is applied on $\dot{M}$ to derive $\widehat{\dot{\boldsymbol{q}}'}$. Note that each matrix $\dot{G}_l$ is now constructed using the substituted camera matrix $\dot{P}_l \coloneqq H_l^{-1}P_l T$ in contrast to the matrices G_l which were calculated using P_l directly. The final result Q' of the LfD Ellipsoid Approximation is then determined via

$$Q' \coloneqq \mathrm{adj}^{-1}\left(T\,\mathrm{vecs}^{-1}\left(\widehat{\dot{\boldsymbol{q}}'}\right)T^T\right).$$

4.6.2.3 LfDC Ellipsoid Approximation

The last integrated ellipsoid approximation algorithm is the LfDC Ellipsoid Approximation. This algorithm, described in Chapter 3.3.3, further modifies the LfD Ellipsoid Approximation by introducing additional linear systems. In particular, it uses the observation that the center of the ellipsoid Q shall project onto the center of each

ellipse C_l. As the introduced linear systems only utilize the substituted camera matrices $\dot{P}_l$, the LfDC Ellipsoid Approximation cannot be defined more precisely. Therefore, this chapter only gives a brief summary of the algorithm.

The additional linear systems, which represent the observation about the centers of the ellipsoid and the ellipses, are given in (3.31), i.e.

$$\mathbf{0} = \dot{G}_l{}'\widehat{\dot{\boldsymbol{q}}}, \tag{4.9}$$

whereby the matrix $\dot{G}_l{}'$ is determined via

$$\dot{G}_l{}' := \begin{bmatrix} 0 & 0 & 0 & \dot{P}_{l_{1,1}} & 0 & 0 & \dot{P}_{l_{1,2}} & 0 & \dot{P}_{l_{1,3}} & \dot{P}_{l_{1,4}} \\ 0 & 0 & 0 & \dot{P}_{l_{2,1}} & 0 & 0 & \dot{P}_{l_{2,2}} & 0 & \dot{P}_{l_{2,3}} & \dot{P}_{l_{2,4}} \end{bmatrix} \in \mathbb{R}^{2\times 10}.$$

Using these linear systems, the final system is derived as

$$\dot{M}'\dot{\boldsymbol{w}} = \mathbf{0}$$

with

$$\dot{M}' := \begin{bmatrix} \dot{G}_1 & -\widehat{\dot{\boldsymbol{c}}_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \dot{G}_1{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \dot{G}_2 & \mathbf{0} & -\widehat{\dot{\boldsymbol{c}}_2} & \cdots & \mathbf{0} \\ \dot{G}_2{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \dot{G}_n & \mathbf{0} & \mathbf{0} & \cdots & -\widehat{\dot{\boldsymbol{c}}_n} \\ \dot{G}_n{}' & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{8n\times(10+n)},$$

Again, it is solved by applying an SVD on $\dot{M}'$ resulting in $\dot{\boldsymbol{w}}'$. $\dot{\boldsymbol{w}}'$ is then used to derive the solution Q' via

$$Q' := \mathrm{adj}^{-1}\left(T\,\mathrm{vecs}^{-1}\left(\widehat{\dot{\boldsymbol{q}}'}\right)T^T\right).$$

Both, the LfD and the LfDC Ellipsoid Approximation are more stable when used on noise afflicted data. However, while the LfD Ellipsoid Approximation is more efficiently solved, the LfDC Ellipsoid Approximation supposably generates more accurate ellipsoids. To assess which of these algorithms is more suited for application in UASs, their performance in regards to their accuracy and computational cost is evaluated later on in Chapter 6.

4.7 Output Processing

As a last step of the RTL, the Output Processing utilizes the created environment map to generate output as discussed in Chapter 4.2.3. Most of the necessary output data can be derived in a trivial manner. For example, the L_2 distance between the position

$\boldsymbol{p_l}$ of an object and the UAV's position $\boldsymbol{c_{UAV}}$ is simply calculated as $\|\boldsymbol{p_l} - \boldsymbol{c_{UAV}}\|$ because the UAV position $\boldsymbol{c_{UAV}}$ is directly provided in the input data of the RTL.

If specified by the corresponding constant and if ground-truth information for the localized object is provided, the output data also consists of ground-truth evaluation data. Again, the L_2 distance between the localized object's position and the position of the ground-truth object is trivial to determine. However, if the Ellipsoid Approximation localization algorithm is being used, the ground-truth evaluation data also consists of the two overlaps, each between two ellipsoids. As given by (4.1), the overlap $o(Q_1, Q_2)$ is defined as the volume of the intersection of Q_1 and Q_2 divided by the volume of the union of Q_1 and Q_2. Because $Q_1 \cap Q_2 \subseteq Q_1 \cup Q_2$ yields true and also because the volume cannot be negative, the overlap $o(Q_1, Q_2)$ is a value in the interval $[0; 1]$. In particular, the overlap between the approximated ellipsoid Q' and the ground-truth ellipsoid Q as well as the overlap between the translated ellipsoid $\overline{Q'}$ and Q shall be determined. Because the ground-truth ellipsoid data of an object is given as constants only consisting of the ellipsoids center $\boldsymbol{c_Q}$ as well as its principle axes $\boldsymbol{p_{Q_l}}$ with $l \in \{1,2,3\}$, a method must be defined which derives the quadric matrix form of $Q \in \mathbb{R}^{4\times4}$. This method is also needed to determine $\overline{Q'} \in \mathbb{R}^{4\times4}$.

Firstly, let $\dot{Q} \in \mathbb{R}^{4\times4}$ be the unit sphere in quadric matrix form. Obviously, $\dot{Q}$ is a special ellipsoid which is centered at the origin $(0,0,0)^T$ and all of its principle axes have lengths of one. As before, any given point $\boldsymbol{x} \in \mathbb{R}^3$ lies on $\dot{Q}$ if, and only if,

$$\tilde{\boldsymbol{x}}^T \dot{Q} \tilde{\boldsymbol{x}} = 0 \tag{4.10}$$

yields true [36]. Now, (4.10) can be modified so the derivation of the quadric matrix form Q of the ground-truth ellipsoid, given its center $\boldsymbol{c_Q}$ and the principle axes $\boldsymbol{p_{Q_l}}$ with $l \in \{1,2,3\}$, becomes clear. Instead of transforming the ellipsoid $\dot{Q}$ in (4.10), we can instead transform $\boldsymbol{x}$. More specifically, this transformation, in the following referred to as $T \in \mathbb{R}^{4\times4}$, shall transform the principle axis $\boldsymbol{p_{Q_l}}$ to the base vector $\boldsymbol{b_l} \in \mathbb{R}^3$ with $b_{l_k} \coloneqq 0$ if $k \neq l$ and $b_{l_k} \coloneqq 1$ if $k = l$. Furthermore, T also transforms the center of the ellipsoid $\boldsymbol{c_Q}$ to the origin $(0,0,0)^T$. Using this information, T is defined through

$$T \coloneqq \begin{bmatrix} \boldsymbol{p_{Q_1}} & \boldsymbol{p_{Q_2}} & \boldsymbol{p_{Q_3}} & \boldsymbol{c_Q} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \in \mathbb{R}^{4\times4}. \tag{4.11}$$

Note that T^{-1} does not represent a Euclidean transformation because the principle axes $\boldsymbol{p_{Q_l}}$ may have arbitrary lengths and, therefore, the determinant of the upper-left 3×3 submatrix can be any value other than zero. Using T, the condition to test whether $\boldsymbol{x}$ lies on Q is now given as

$$\tilde{\boldsymbol{x}}^T T^T \dot{Q} T \tilde{\boldsymbol{x}} = 0.$$

Here we can see that the transformation of $\boldsymbol{x}$, i.e. T, can also be directly applied to the ellipsoid itself. Therefore, $Q \in \mathbb{R}^{4\times4}$ is determined as

$$Q \coloneqq T^T \dot{Q} T, \tag{4.12}$$

whereby a closed-form equation for T is given in (4.11). Furthermore, knowing that $\dot{Q}$ is the unit sphere, it is obvious that it is defined as

$$\dot{Q} \coloneqq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{4\times4}.$$

This can also be derived by extending (4.7) to an ellipsoid. (4.12), therefore, provides a closed-form solution for Q. An analogous method can be used to determine $\overline{Q'}$. There, the ellipsoid's principle axes are $\boldsymbol{p}_{\boldsymbol{Q'}_l}$ while its center remains $\boldsymbol{c}_{\boldsymbol{Q}}$.

After the quadric matrices Q, Q' and $\overline{Q'}$ have been provided, an approach is needed to calculate the volume of the union of two ellipsoids as well as the volume of their intersection used in the overlap. Because these are complex problems themselves and because the performance of such an approach is not the focus of it, as it is used in the evaluation only, the determination of these volumes can be done by simply sampling discrete points with fixed distances. If the sampling rate is sufficiently high, the volumes calculated using sampling approach their real volume. Therefore, the sampled overlap also approaches the real overlap.

5 Implementation

In this chapter, the implementation of the RTL is elaborated. The implementation of the Real-Time Object Localizer itself will be referred to as “the software” or simply “the implementation”. As stated in Chapter 1.2, the software shall be integrated into the APOLI project. However, currently there is no implementation of the only component of APOLI the RTL shall communicate with, namely the CTH. For this reason, the implementation communicates with lower-level components instead. As there was also no implementation of the FPD available at the time the RTL has been implemented, the communication is limited to the MAL only.

In Chapter 5.1, some general information about the software are presented. After that, Chapter 5.2 lists and describes the C/C++ libraries that have been integrated into the software for different fields. A brief overview of the implemented software architecture is provided in Chapter 5.3 and, finally, Chapter 5.4 elaborates how to start and the use the software.

5.1 Software Information

The implemented software can be found on the enclosed CD. The folder structure of the project is elaborated in Attachment **Error! Reference source not found.**. The software is written in C++ using the C++17 standard. Furthermore, it has been developed using the Microsoft Visual Studio 2019 IDE. The Visual Studio solution is set up for building the software as a 64-bit application on Windows. In particular, Windows 10 has been used for this. Furthermore, the solution also supports building the software as a 32-bit application on a remote Linux machine. An ODROID-XU4 has been used as this remote machine.

5.2 Utilized Libraries

The software uses a number of C/C++ libraries for different areas. These are:

- GSL [49] (Version 2.6),
- OpenCV [50] (Version 3.4.5),
- AirLib [51] (Git commit 659a7db),
- OpenGL [52] (Version 4.3),
- glad [53] (Version 0.1.33),
- GLFW [54] (Version 3.3.2) and
- GLM [55] (Version 0.9.9.8).

GSL (GNU Scientific Library) [49] is a C library providing a variety of functions for numerical calculations. The majority of vector and matrix operations of the RTL are implemented using this library. GSL is linked against statically on both, Windows and Linux.

The C++ library OpenCV (Open Computer Vision Library) [50] provides functionality for image I/O as well as image manipulation and image operations in general. It is used in two separate areas. Firstly, for the evaluation a simple implementation of the FPD was required. Here, OpenCV is being used to generate the Fundamental Points given a camera image. Furthermore, OpenCV is also used to retrieve images themselves from a camera. This library is linked dynamically at runtime.

Next, AirLib [51] is a C++ library which provides an interface between C++ and the AirSim [51] simulator used in the SITL simulation of APOLI. It is, among others, used to retrieve images shot from the camera mounted on the simulated UAV. On Windows, this library is linked statically. This library is also the reason the Windows project of the RTL does not support compiling a 32-bit version of the software as AirLib currently only supports 64-bit. On Linux, this library is supported in theory. However, because it requires an installation of the Unreal Engine and also because it is used for the integration of the simulation only, it was deemed too heavy for the utilized ODROID-XU4 and, therefore, disregarded on Linux as a whole.

OpenGL (Open Graphics Library) [52], in combination with glad [53] and GLFW (Graphics Library Framework) [54], are used for the hardware accelerated rendering of the software's scene. This scene includes, but is not limited to, the UAV, the coordinate systems used to calculate the camera's parameters, as elaborated in Chapter 4.4.1, and the objects of the environment map. Because the ODROID-XU4 used does not support conventional OpenGL, all these libraries are only linked against on Windows. There, they are linked statically.

Lastly, GLM (OpenGL Mathematics) [55] is a header-only C++ library which provides basic functionality for numerical operations on vectors and matrices with a focus on application in OpenGL. In particular, while GLS stores matrices row-major, GLM uses a column-major layout. Because this library is a header-only library, it does not need to be linked against at all.

5.3 Architecture

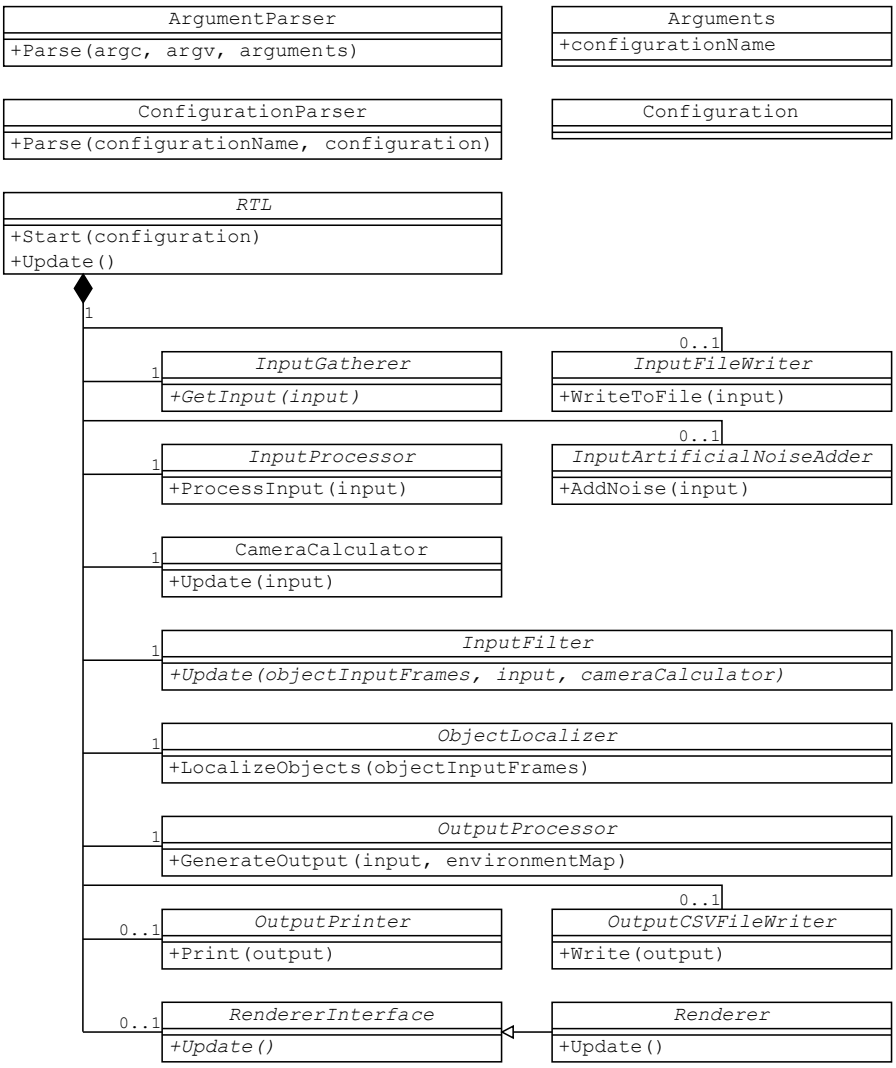


Figure 5.1: Simplified UML Class Diagram of the Software.

In this chapter, a brief overview of the architecture of the implemented software is given. For this, Figure 5.1 illustrates a simplification of the software's architecture as an UML class diagram. The figure only shows a handful of classes for which only some functions are visualized. Furthermore, to improve the readability of the figure, the composition relationships between the `RTL` class and the classes of its member objects below it are combined into one relationship even though all member objects have a separate composition relationship with the `RTL`. The main procedure begins by processing the command-line arguments provided using the `ArgumentParser` class. As there is only one argument, the name of the configuration file to load, the implementation of this class is simply. In the context of the implemented software, a configuration is defined as the set of all constants of the RTL. These configurations are saved and loaded as files. In particular, the main procedure calls the

`ArgumentParser::Parse()` function with the number of arguments, `argc`, as well as an array containing all provided command-line arguments, `argv`. The `ArgumentParser` then parses the given arguments into an object of the `Argument` structure.

After the command-line arguments have been parsed into the `Argument` object, the main procedure of the software then parses the corresponding configuration file via the `ConfigurationParser` class. When being provided with the name of the configuration file to read, the `ConfigurationParser::Parse()` function parses all read configuration values into the provided `Configuration` object. An elaboration on the structure of any configuration file can be found in Attachment B. It should be noted here, however, that the configuration files consist of sections which may be structured hierarchically. For this reason, each of these sections possesses a corresponding section parser class which parses only the section's part of the configuration file. Consequently, `ConfigurationParser` and the other section parser classes parse the keyword-value pairs of their corresponding section themselves but outsource the parsing of subsections to their corresponding parsers.

Finally, the main procedure is ready to start the RTL. As seen in Figure 5.1, the name of the `RTL` class as well as the names for a majority of its members' classes, are given in italics. This indicates that these are either abstract C++ classes, i.e. classes which cannot be initialized because they are missing logic, or template classes. In the following, both of such classes will simply be referred to as abstract classes. Abstract functions are also written in italics. As the control flow of the RTL, or more specifically its individual steps, varies depending on the configuration of the FPD, the templates of these abstract classes differentiate between data used for the triangulation or the ellipsoid approximation approach. In addition to this, some of the subclasses of these abstract classes implement more specific logic for the corresponding localization approach. In general, the names of the subclasses end with their corresponding localization approach, i.e. `…Triangulation` and `…EllipsoidApproximation`. The main procedure starts by allocating an `RTL` object of the approach to use as specified by the `Configuration` object, i.e. `RTLTriangulation` or `RTLEllipsoidApproximation`. The `RTL` class itself does not implement much logic. Instead, the logic of each individual step of the `RTL` is outsourced to dedicated classes. In the figure, each member object of `RTL` represents such a step, i.e. there are a total of eleven steps. While some of these steps correspond exactly to one step of the RTL's pipeline as elaborated in Chapter 4, other steps represent an extension of the RTL's pipeline. The specific step classes to allocate are defined by the two

subclasses of RTL, i.e. `RTLTriangulation` and `RTLEllipsoidApproximation`. After the main procedure allocated a subclass of `RTL`, it starts the RTL via the `RTL::Start()` function. This function allocates and starts all classes of the steps to use as specified by the `Configuration` object. Once the RTL and its steps are started, the main procedure enters a loop in which it updates the RTL via the `RTL::Update()` function for as long as it is started. Note that the RTL does only stop if its input could not be gathered and the `Configuration` specifies that the `RTL` shall stop in such cases. Otherwise, the main procedure remains in this loop indefinitely.

Now, the update procedure of the RTL will be elaborated briefly. As this procedure represents one pass of the RTL's pipeline, it starts by gathering input data of the RTL. For this, the `InputGatherer::GetInput()` function is called. The source of the input data is specified in the `Configuration` object and implemented via subclasses of `InputGatherer`. After valid input data has been gathered and if desired, the `InputFileWriter::WriteToFile()` function is then used to write this input data into a file. This is useful to create input datasets which later can be directly used as input to the RTL. The next step represents the first step of the RTL's concept, i.e. the Input Processing, as elaborated in Chapter 4.3. In particular, this Input Processing step is performed via `InputProcessor::ProcessInput()`. The `RTL` continues, if specified by the `Configuration`, by adding artificial noise to some input data using the `InputArtificialNoiseAdder::AddNoise()` function. This can be used to evaluate the influence of input data noise on the localization's result. Subsequently, `CameraCalculator::Update()` is called to calculate the camera's parameters, i.e. perform the second step of the RTL's concept. Note that this is the only step which is not implemented as an abstract class. As discussed in the conceptualization of the RTL, the next step is the Input Filtering. This is integrated into the RTL via the `InputFilter::Update()` function. Afterwards, the Object Localization step is executed using `ObjectLocalizer::LocalizeObjects()`. The final step of the concept, i.e. the Output Processing, is then performed by calling `OutputProcessor::GenerateOutput()`. The next steps of the RTL, i.e. `OutputPrinter::Print()` and `OutputCSVFileWriter::Write()`, are only executed if specified by the `Configuration` object. These procedures provide functionality to scatter the output of the RTL. In particular, while the `OutputPrinter` class provides functionality to print the output to the console, `OutputCSVFileWriter` can be used to write it into so-called comma-separated values (CSV) files. Additional ways to scatter the output can be implemented easily by adding additional steps. Obviously, the exact order in which these output scattering steps are executed does

not matter. Finally, as the last step the RTL updates, if specified by the configuration, the rendered scene by calling the `RendererInterface::Update()` function. Note that an interface is used for this because the class which implements some of the rendering logic, i.e. the `Renderer` class, may not be available. In particular, the `Renderer`, all its subclasses as well as their dependencies are only included in the project when building on Windows as the ODROID-XU4 board does not support OpenGL.

A more in-depth elaboration of some of the individual steps' classes of the `RTL` and their implementation can be found in Attachment A.1. Additionally, Attachment A.2 also provides a description of some of the modifications made to the MAL to allow for the communication with the RTL.

5.4 Usage Guide

As elaborated previously, the software must be started via the command line as additional command-line arguments are required. Otherwise or if the command-line arguments provided are of invalid format, a dialog is printed to the console describing the arguments needed. In particular, the software only possesses a single argument. This is the name of the configuration file to load. More specifically, the software appends the file extension `.cfg` and looks for the configuration file inside the `Configurations/` subdirectory in respect to the working directory. To specify the file name, the command-line argument `--configuration` or `-c` must be used. This indicates that the next argument stands for the name of the configuration file. For example, starting the software with the command-line arguments `--configuration development` loads the development configuration file being located at `Configurations/development.cfg`. A list of preexisting configuration files can be found in Attachment B.

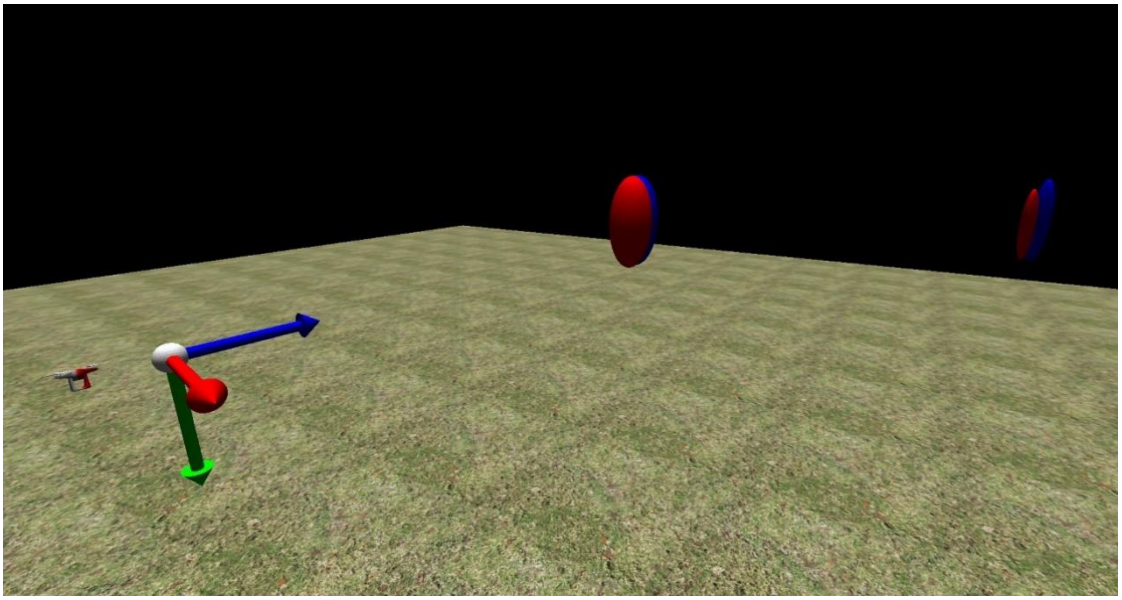


Figure 5.2: Rendered Scene containing the UAV, the Camera Coordinate System, the localized Objects (Red) and the Ground-Truth Objects (Blue).

Once the software has been started, its output can be handled in any combination of three different ways as specified by the read configuration file. It may be printed to the console, written into a CSV file and/or rendered as a three-dimensional scene. In Figure 5.2 a rendered scene comprised of multiple objects is shown. On the left-hand side of the figure, the UAV as well as the Camera coordinate system can be seen. Here, it is obvious that at least one offset vector which was specified in the corresponding configuration was unequal to the zero vector. The Camera coordinate system, alongside all other coordinate systems that may be rendered, is visualized via three colored arrows each of one meter length. In particular, the x-axis is rendered red, the y-axis is colored green and the z-axis is blue. In addition to this, the scene also contains the localization of two objects. More specifically, while the objects' approximated ellipsoids are rendered red, the ground-truth objects are visualized as blue ellipsoids. Lastly, a ground plane is also rendered to allow for easy orientation in the scene. Using the configuration, the user can specify which objects are to be rendered.

The user may only directly interact with the software if the scene is rendered. There, they are able to, among other things, manipulate the camera used for rendering or print output information corresponding to specific object localizations. A full list of user interactions with the rendered scene can be found in Attachment C.

6 Evaluation

This chapter evaluates the general performance of the Real-Time Object Localizer (RTL) using the two supported FPD configurations, i.e. the configurations in which either one or four Fundamental Points per detected object are provided. In particular, in Chapter 6.1 the triangulation approach, which has been elaborated in Chapter 4.6.1, is being used for evaluation while Chapter 6.2 evaluates the RTL using the ellipsoid approximation presented in Chapter 4.6.2. For both of these object localization approaches, the integrated localization algorithms are analyzed and evaluated in regards to accuracy and computation cost.

For this, a number of input datasets have been generated in different environments. The outputs of the RTL generated using these datasets, i.e. the objects' localizations, are then compared to the ground-truth information of these objects. In particular, three different environments have been used to generate the input datasets. Firstly, input datasets were produced in a synthetically generated environment. Furthermore, the SITL simulation of APOLI was utilized to simulate the second environment. Finally, the RTL is also evaluated on input datasets generated in a real-world outdoor environment using the UAV of the Indoor Flight Center, or IFC, of the APOLI project.

For the evaluation of the computational performance, the number of inputs the RTL is able to process per second using the corresponding localization algorithm is measured on two different systems. For this, a desktop workstation as well as an ODROID-XU4, which is also used in the APOLI project, have been used. The hardware components integrated into the ODROID-XU4 have already been described in Chapter 2.3.1. The desktop workstation, on the other hand, features an AMD Ryzen™ 9 3900X processor. This is a twelve core CPU with a base clock speed of 3.8GHz. In addition to this, the desktop workstation possesses 16GB of main memory. It should be mentioned that the software implementation of the RTL only supports multi-threading for limited areas. In particular, these areas are the generation of the ground-truth evaluation information of the RTL's output data as well as the scene rendering. Because both of these procedures are not executed for the evaluation of the computational performance, the RTL only uses a single core of the CPU for this evaluation.

6.1 Triangulation

In this chapter, the RTL which uses the triangulation approach is evaluated. Therefore, the FPD configuration in which one Fundamental Point per detected object is being

provided in the input data is used. The following subchapters evaluate the RTL using the triangulation approach in the different environments used to gather the datasets.

In the following, the RTL used the Spherical Input Filter as presented in Chapter 4.5.2. The number of regions generated by it was defined differently for the different environments and is, therefore, specified in their corresponding chapter. However, the maximum number of Input Frames that could be selected by the Spherical Input Filter was always set to 20. Furthermore, for the L_∞ Triangulation and the Hybrid Triangulation, a total of $o = 100$ line searches in random directions were performed for each localization and the constant which defines the maximum distance between the parameter of the found minimum of each line and the real minimum's parameter was set to $S = 0.01$.

6.1.1 Synthetic Environment

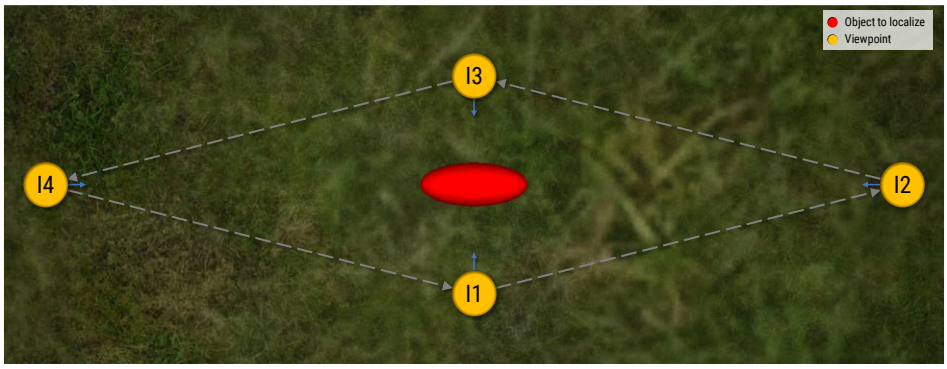


Figure 6.1: Top-Down View of the Synthetical Environment.

The first environment in which the RTL has been evaluated in is a synthetically generated environment. A top-down view on it can be seen in Figure 6.1. There, the object that is to be localized was placed at a fixed position and is colored red. For the generation of the input data, four different viewpoints located around the object have been used. These viewpoints are labelled I1, I2, I3 and I4 in the figure. This is also the order in which the input data was generated. After I4 was reached, the input sequence started anew until a total of 1000 inputs were generated, i.e. each viewpoint has been used to generate input 250 times. The location of the viewpoints were specified as $\boldsymbol{c_{I1}} \coloneqq (0,0,0)^T$, $\boldsymbol{c_{I2}} \coloneqq (10,40,0)^T$, $\boldsymbol{c_{I3}} \coloneqq (20,0,0)^T$ and $\boldsymbol{c_{I4}} \coloneqq (10,-40,0)^T$ in the UAV Local NED coordinate system. Further, the position of the object was $\boldsymbol{c} \coloneqq (10,0,0)^T$ while its principle axes were defined as $\boldsymbol{p_1} \coloneqq (2,0,0)^T$, $\boldsymbol{p_2} \coloneqq (0,5,0)^T$ and $\boldsymbol{p_3} \coloneqq (0,0,3)^T$. Therefore, two of the viewpoints, i.e. I1 and I3, have a distance of ten meters to the object's center while I2 and I4 are located 40 meters away from the object. Each input's virtual camera was placed at the same position as the viewpoint itself while

pointing directly at the object's center. In the figure, the view direction of each camera is visualized as a blue arrow. Lastly, the resolution of the virtual camera image has been specified as 1000×1000 with a field of view of 80°.

This synthetical environment has further been used to generate a number of datasets on which different kinds of artificial noise is applied. More specifically, for each kind of noise seven datasets have been generated for evaluation as the noise as well as the L_∞ Triangulation and, therefore, the Hybrid Triangulation themselves are non-deterministic. An evaluation for the noiseless scenario, i.e. using the input dataset directly, is not included in this thesis as all of the triangulation algorithms integrated into the RTL generated optimal solutions. Furthermore, as only four distinct viewpoints are used, the number of regions generated by the Spherical Input Filter was set to 2000 to allow input data noise to switch into neighboring regions consistently.

6.1.1.1 Fundamental Point Noise

Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Generalized Midpoint	~0.37	~0.61	~46,052.63	12,500
L_∞	~0.48	~0.85	~616.36	~180.66
Hybrid	~0.39	~0.69	~615.76	~180.24

Table 6.1: Evaluation Data for Input Datasets generated using the Synthetic Environment with Fundamental Point Noise.

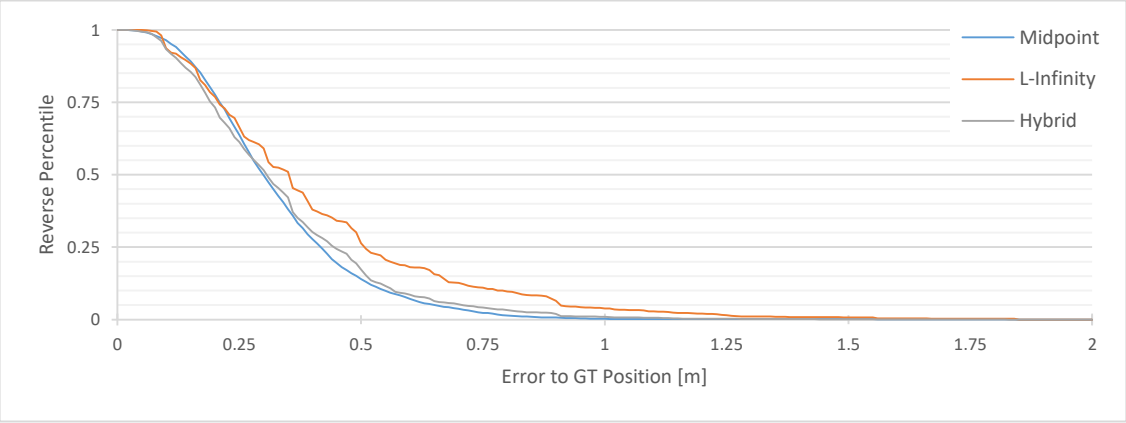


Figure 6.2: Reverse Percentile of Ground-Truth Position Errors for Input Datasets generated using the Synthetic Environment with Fundamental Point Noise.

The first evaluation is performed on datasets on which noise has been applied to the positions of the Fundamental Points. These Fundamental Points, which are for the noiseless scenario always located at $(500,500)^T$, are displaced in a random direction by a random distance. More specifically, the displacement distance is generated using a Gaussian distribution with a mean of $\mu = 0$ and a standard deviation of $\sigma = 16$. Here,

a negative displacement factor simply means that the displacement is performed in negative direction. Some of the evaluation results can be seen in Table 6.1. In particular, the table lists the average root mean square error (RMSE) as well as the average 95th percentile of each of the triangulation algorithms in respect to the ground-truth position of the object. Given a vector of errors $\boldsymbol{e} \coloneqq (e_1, \dots, e_n)^T \in \mathbb{R}^n$, the RMSE is defined as

$$\mathrm{RMSE}(\boldsymbol{e}) \coloneqq \sqrt{\frac{1}{n}\sum_{l=1}^{n} e_l}.$$

Therefore, larger errors e_l have a disproportionally large effect on the RMSE. Furthermore, Figure 6.2 plots the so-called reverse percentile against certain ground-truth position errors. Here, this reverse percentile is simply defined as one minus the percentile.

The results indicate that the Generalized Midpoint Triangulation performs the best even if noise is applied on the Fundamental Points. Not only was its average RMSE and average 95th percentile the lowest, but the distribution of errors shown in Figure 6.2 demonstrate that it generated the least number of localizations with large errors to the ground-truth position. The L_∞ Triangulation generated worse results than the Generalized Midpoint Triangulation because it is, as stated in Chapter 3.2.5, heavily afflicted by outliers in the input data. However, the RTL does not perform any removal of outliers and, therefore, these outliers in the input data are used for the triangulation. Furthermore, as long as the Input Filtering is not being provided with new input corresponding to the same region as the outliers, such outliers are stored in the Input Frame database indefinitely and potentially selected for future triangulations. In fact, another disadvantage of the Spherical Input Filter is that the ratio between the number of outlying and non-outlying Input Frames in the Input Frame database is disproportional to the actual ratio between outlying and non-outlying inputs given to the Input Filtering step. Regarding the Hybrid Triangulation, while it generated more accurate object positions than the L_∞ Triangulation, it was still outclassed by the Generalized Midpoint Triangulation for a majority of datasets. In particular, even though Figure 6.2 indicates that the Hybrid Triangulation is able to generate a greater number of localization results with a low error to the ground-truth position, it also generated a greater number of results with high errors. In fact, the Hybrid Triangulation only generated the overall most accurate localizations for a handful of datasets throughout the whole evaluation. The reasons for this lie in the non-trivial minimization problem of the Hybrid Triangulation as elaborated in detail in Chapter 4.6.1.3.

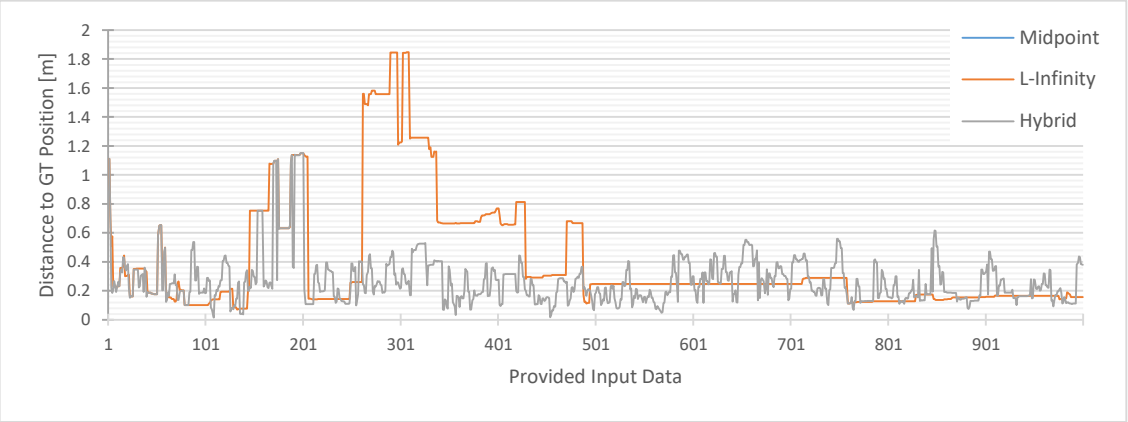


Figure 6.3: Distance to Ground-Truth Position for one Input Dataset of the Synthetic Environment with Fundamental Point Noise.

The impact of outliers in the input data is also illustrated in Figure 6.3 in which the distance between the ground-truth object position and the position of the localized object is plotted for all considered triangulation algorithms on the same noise afflicted input data. There, it can be seen that, for example, around the 140th input data provided to the RTL was an outlier. This outlier then greatly influenced the triangulation result for the rest of the localization. In cases for which the ground-truth error of the L_∞ or the Hybrid Triangulation decreased, the RTL was either provided with outliers that negated the effect of previous outliers or the outliers' Input Frames were not selected by the Spherical Input Filter. In general, it can also be seen that the Generalized Midpoint Triangulation generates localization results which were much more stable than the results of the L_∞ Triangulation or the Hybrid Triangulation. In addition to this, Figure 6.3 also illustrates that for all three triangulation algorithms, the accuracy is low at the beginning. This is because the implementation of the RTL begins to triangulate as soon as possible, i.e. as soon as there are two Input Frames. By doing so, however, noise influences the localization results greatly as it cannot be smoothed out effectively. These inaccuracies in the beginning of datasets can easily be avoided by determine a higher threshold than two for the required number of Input Frames. All of these observations apply to all of the seven input datasets.

Lastly, Table 6.1 also lists computational performance criteria for all three triangulation algorithms, i.e. the number of inputs the RTL is able to process per second, for the two reference systems. It can be seen that the Generalized Midpoint Triangulation is, as a closed-form solution was being provided, by far the most efficient algorithm. The L_∞ and Hybrid Triangulation performed 100 line searches each which themselves are, as described in Chapter 4.6.1.2, implemented iteratively. Therefore, to improve the performance of these algorithms, the number of line searches o or the maximum

distance S have to be adjusted. This, however, will also negatively influence the accuracy.

6.1.1.2 Limited Fundamental Point Noise

Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Generalized Midpoint	~0.33	~0.53	~46,979.87	~12,681.16
L_∞	~0.29	~0.55	~697.21	~205.16
Hybrid	~0.3	~0.52	~697.14	~204.59

Table 6.2: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point Noise.

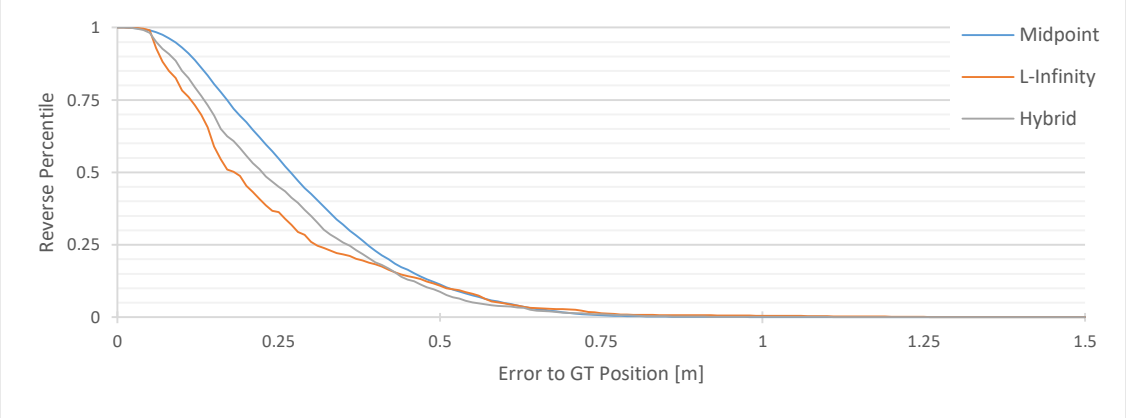


Figure 6.4: Reverse Percentile of Ground-Truth Position Errors for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point Noise.

For the next evaluation, another set of datasets on which artificial noise was applied to the Fundamental Points' positions, has been generated. Again, a Gaussian distribution with $\mu = 0$ and $\sigma = 16$ was used for this. However, the magnitude of the artificial noise was limited to $32 = 2\sigma$ meaning that if the displacement distance generated using the Gaussian distribution has a greater distance to $\mu = 0$ than 32, the value was rerandomized. By limiting the noise applied to the Fundamental Points, outliers were effectively removed from the datasets. Some of the evaluation's results can be seen in Table 6.2 and Figure 6.4. As expected, the L_∞ and Hybrid Triangulation now outperform the Generalized Midpoint Triangulation in regards to accuracy. The reverse percentile clearly indicates this as the Generalized Midpoint Triangulation produced a larger percentile of object locations with ground-truth errors between $0.05m$ and $0.5m$. In regards to ground-truth errors larger than $0.5m$, the triangulation algorithms performed nearly identical.

6.1.1.3 Limited UAV Position Noise

Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Generalized Midpoint	~0.58	~0.98	~47,297.3	~13,133.21
L_∞	~0.72	~1.25	~2,324.04	~664.01
Hybrid	~0.67	~1.15	~2,322.5	~663.82

Table 6.3: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited UAV Position Noise.

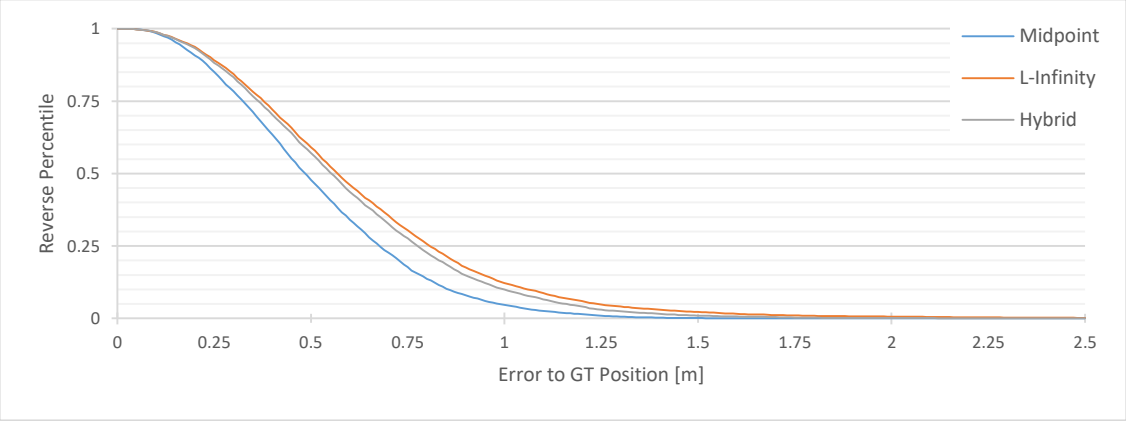


Figure 6.5: Reverse Percentile of Ground-Truth Position Errors for Input Datasets generated using the Synthetic Environment with Limited UAV Position Noise.

The results of the third evaluation performed using the synthetic environment are illustrated in Table 6.3 as well as Figure 6.5. For this evaluation, artificial noise was not applied on the Fundamental Points' positions but instead on the positions of the UAV themselves displacing them in a random direction. Once more, the magnitude of the translation was generated using a Gaussian distribution with $\mu = 0m$ and $\sigma = 1m$. To avoid generating outliers, the displacement distance was limited to $2m = 2\sigma$. Therefore, the noise added to the UAV's positions were of high magnitude as a $2m$ displacement is unrealistically high for real-world applications using reasonable GNSS sensors. It can be seen that the Generalized Midpoint Triangulation provides the most accurate triangulation results out of all three algorithms integrated into the RTL. This is also to be expected as the noise is directly applied to the cameras' positions, i.e. the start points of the rays or rather the lines, in the three-dimensional Euclidean space. Because the Generalized Midpoint Triangulation fully operates on this exact space, it is, theoretically, able to smoothen out the applied noise perfectly. The L_∞ Triangulation and Hybrid Triangulation, on the other hand, operate on the two-dimensional image planes and use reprojections to generate their localization results. When doing so, the applied noise does not influence the localization linearly like it does if the Generalized Midpoint Triangulation is used. For these reasons and also because the Generalized Midpoint Triangulation provides a closed-form solution, it is the triangulation algorithm of choice when dealing with input on which the cameras' positions have been afflicted

by noise. It should be noted that, because noise was added to the UAV positions, the view directions of one viewpoint were always the same. Therefore, the same viewpoint was associated with the same region of the Spherical Input Filter at all times and the Object Localization step of the RTL was provided with the four most recent Input Frames. For this reason, the L_∞ and Hybrid Triangulation were not influenced by outliers as much as such outliers were overwritten with new input data as soon as the same viewpoint was used.

When comparing the computation performance of each triangulation algorithm with their corresponding costs in Table 6.1, it can be seen that while the RTL using the L_∞ and Hybrid Triangulation generated their triangulation results significantly faster. For the Generalized Midpoint Triangulation this performance gain was neglectable. The overall cause of the increased performance is the fact that the number of Input Frames selected is limited to four. Before, the selected Input Frames easily reached twenty, i.e. the maximum input size of the Object Localization step. As the Generalized Midpoint Triangulation can be solved extremely efficiently, the decreased number of Input Frames only had a small influence on the overall performance. On the other hand, solving the L_∞ and Hybrid Triangulation has high computational costs and, therefore, the reduced input size drastically improved their performance.

6.1.1.4 Limited Fundamental Point and UAV Position Noise

Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Generalized Midpoint	~0.44	~0.73	~46,357.62	~12,567.32
L_∞	~1.38	~2.63	~698.18	~203.87
Hybrid	~0.49	~0.83	~696.79	~203.48

Table 6.4: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point and UAV Position Noise.

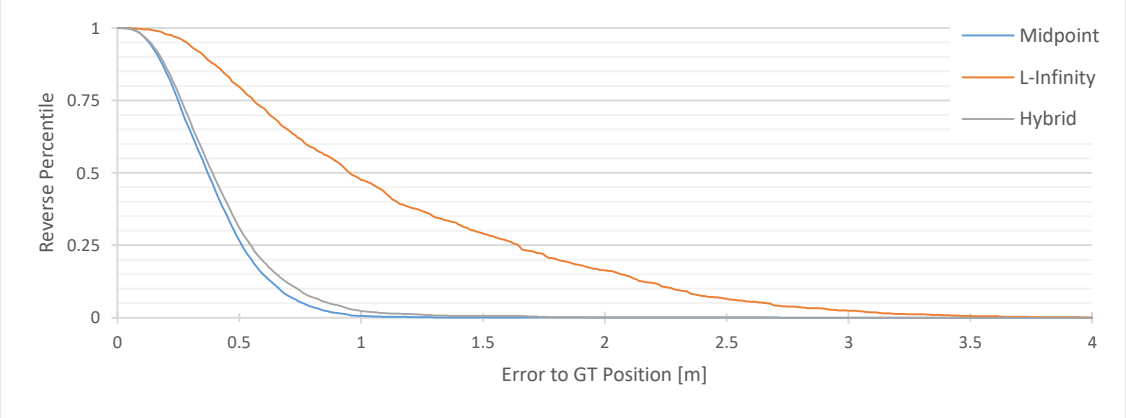
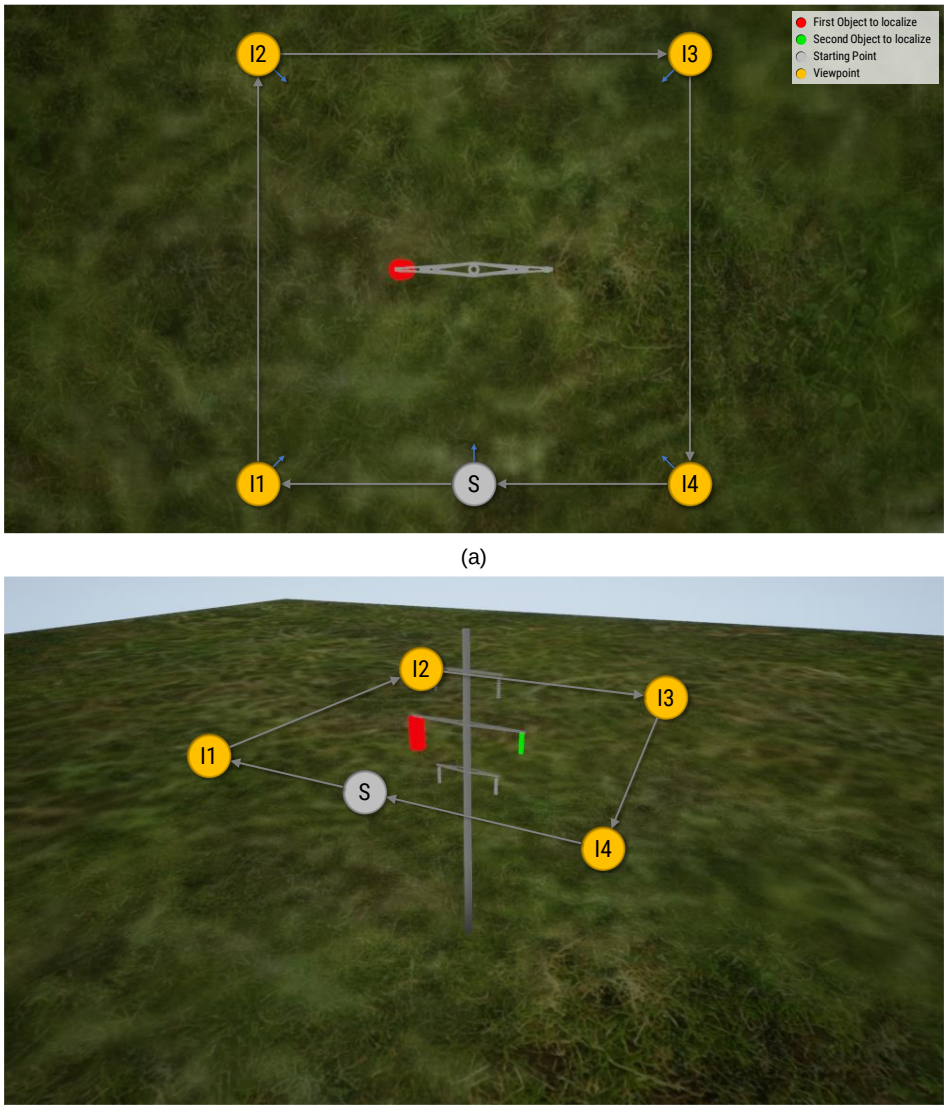


Figure 6.6: Reverse Percentile of Ground-Truth Position Errors for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point and UAV Position Noise.

Table 6.4 and Figure 6.6 illustrate the result of the last evaluation performed on the synthetic environment. For this evaluation, both kinds of noise, i.e. limited noise on the Fundamental Points and limited noise on the UAV's positions, were used to generate the input datasets. The results clearly indicate that the Generalized Midpoint Triangulation performs the best when faced with such datasets. The combination of both kinds of noises produced a higher number of outliers in the input data which influenced the L_∞ Triangulation drastically. Because the Hybrid Triangulation occasionally selected localization results of the L_∞ Triangulation, i.e. those results with a lower L_2 distance, it did not generate more accurate results than the Generalized Midpoint Triangulation.

6.1.2 Simulation Environment



(a)

(b)

Figure 6.7: Top-Down View (a) and Diagonal View (b) of the Simulation Environment.

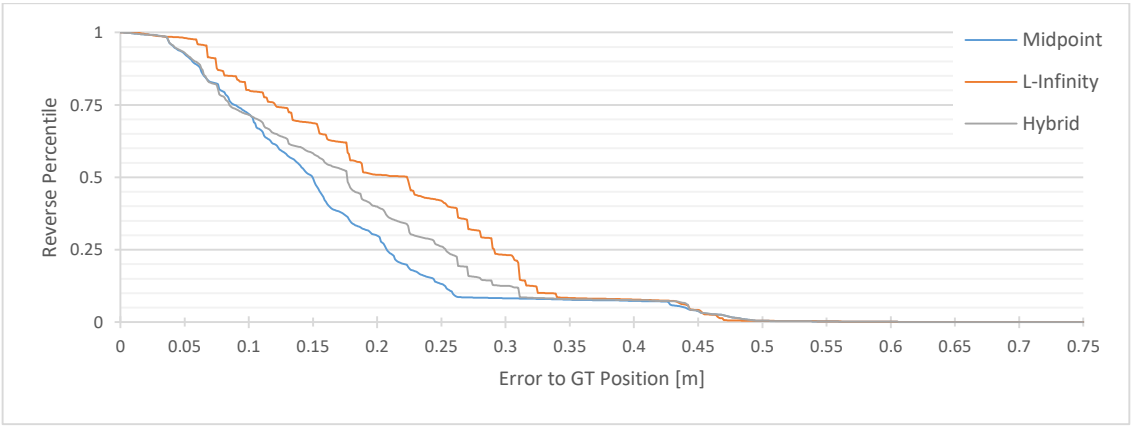
The next environment in which the RTL has been evaluated in is an environment simulated in the SITL simulation of the APOLI project. A top-down as well as a diagonal view of it can be seen in Figure 6.7. The environment consisted of a power pole as well as two cylindrical objects that were to be localized. These objects, in the figure colored red and green respectively, were placed at two of the power pole's insulators while their bases were parallel to the ground. Therefore, this environment represents a concrete test environment for the RTL in the APOLI project. While the red cylinder's center $\boldsymbol{c_R}$, its radius r_R and its height h_R were defined as $\boldsymbol{c_R} \coloneqq (10.04, -3.49, -15.92)$, $r_R \coloneqq 0.5$ and $h_R \coloneqq 1.03$, the corresponding parameters of the green cylinder were given as $\boldsymbol{c_G} \coloneqq (10.04, 3.42, -16.37)$, $r_G \coloneqq 0.13$ as well as $h_G \coloneqq 0.63$. Again, these values are in the UAV Local NED coordinate system. Figure 6.7 also illustrates the flight path the simulated UAV had taken in order to gather the input data. Note that, while the input data of the synthetic environment was generated using discrete viewpoints, the simulation environment's flight path was continuous. This flight path started at the point $\boldsymbol{c_S} \coloneqq (0, 0, -16)$. The UAV then traversed the viewpoints $\boldsymbol{c_{I1}} \coloneqq (0, -9.75, -16)$, $\boldsymbol{c_{I2}} \coloneqq (20, -9.75, -16)$, $\boldsymbol{c_{I3}} \coloneqq (20, 9.75, -16)$ and $\boldsymbol{c_{I4}} \coloneqq (20, 9.75, -16)$ in this order. After reaching the viewpoint I4, the four-viewpoint sequence was repeated once. Once the UAV reached I4 a second time, it returned to the starting position S which concluded the flight path.

As this environment was used to evaluate the RTL in the context of the APOLI project, its integration into the project was used to gather input. Therefore, the MAL was used to provide all of the UAV's flight data. However, as there is currently no implementation of the FPD, the interface of the SITL simulation was utilized to generate the object detection input. More specifically, a virtual camera mounted on the simulated UAV with a resolution of 1280×720 and a field of view of 90° was utilized to retrieve images from the simulation. This camera was pointed at $(10, 0, -16)$, i.e. the power pole, at all times. Simple color-based object detection algorithms were then applied to generate bounding rectangles around the detected objects which consequently resulted in the Fundamental Points. Lastly, the gimbal was set up to keep a roll angle of $0°$ at all times, i.e. keep the camera aligned to the horizon.

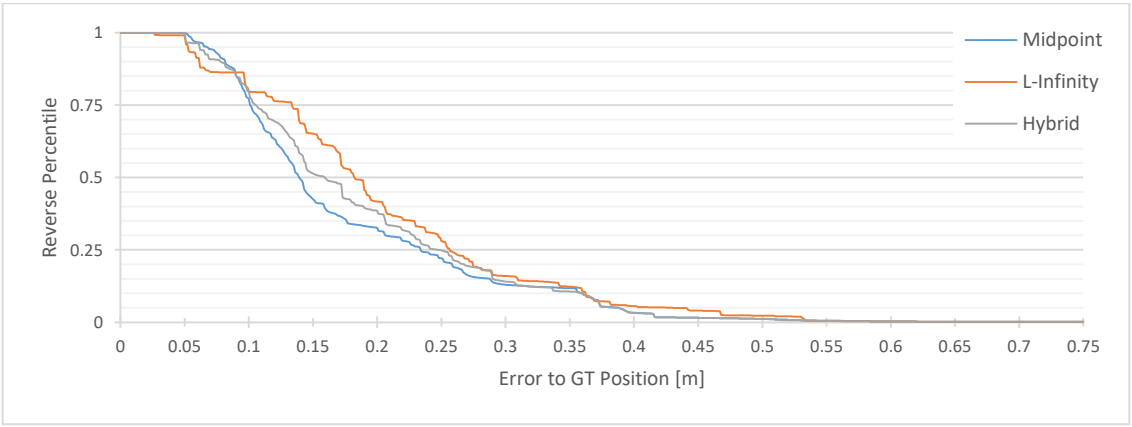
The simulation environment was used to generate seven input datasets on which the following evaluation has been performed. To weight all of the datasets equally in the evaluation, each of them contained exactly 4500 inputs. Furthermore, the number of regions generated by the Spherical Input Filter was set to 100.

	Red Object		*Green Object*	
Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*
Generalized Midpoint	~0.19	~0.29	~0.22	~0.35
L_∞	~0.24	~0.34	~0.25	~0.4
Hybrid	~0.21	~0.32	~0.36	~0.36

Table 6.5: Evaluation Data for Input Datasets generated using the Simulation Environment.



(a)



(b)

Figure 6.8: Reverse Percentile of Ground-Truth Position Errors of Red (a) and Green (b) Object for Input Datasets generated using the Simulation Environment.

Some of the results of the evaluation are illustrated in Table 6.5 as well as Figure 6.8. As can be seen, the RTL is able to localize both objects reasonably accurate with all triangulation algorithms. This is also to be expected as the MAL provided UAV flight data only afflicted by small noise. It should be noted that the timestamps of the input received from the MAL were not equal to one another and, therefore, input processing, as described in Chapter 4.3, had to be applied. However, further noise comes from the generation of the object detection input. In particular, an image received from the simulation was used for this. While the detection of object pixels worked perfectly, the digitalization of the image lead to noise in the precise determination of the Fundamental Points' positions. As the green object was smaller than the red one, its detection was

more affected by this issue. In addition to this, the object detection was not able to gather valid input if the corresponding object that is to be detected was occluded by another object, i.e. the other object or the power pole. In those scenarios, the objects' centers where determined incorrectly. Overall, the results indicate that the Generalized Midpoint Triangulation provides the most accurate object localization even if the occlusion issue is omitted.

6.1.3 Outdoor Environment

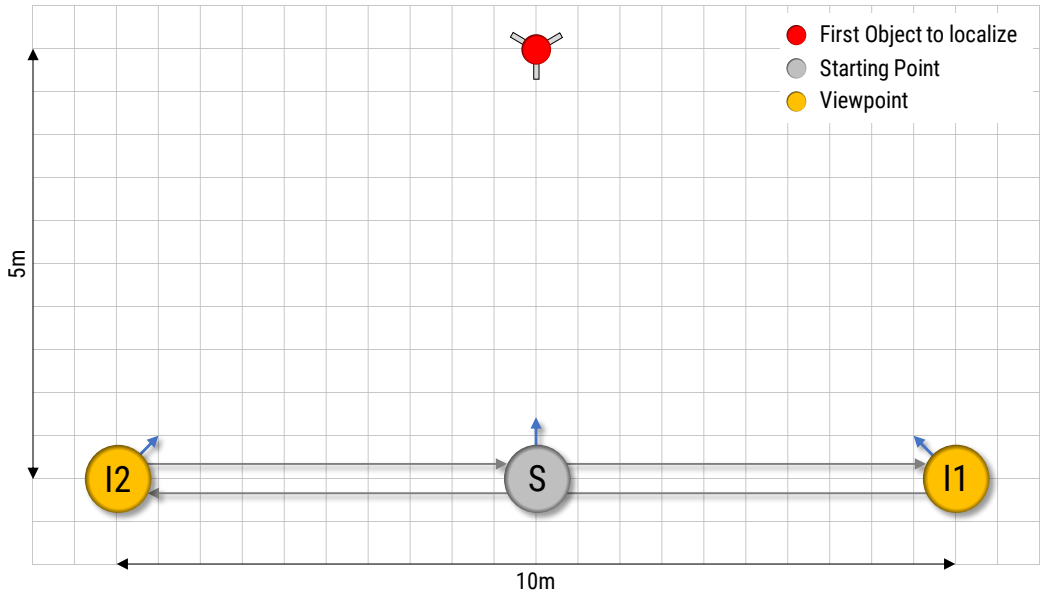


Figure 6.9: Simplified Top-Down View of the Outdoor Environment. The visualized Red Object is larger than its Real-World Correspondence.

The final environment utilized for the evaluation of the RTL is a real-world outdoor environment for which the UAV of the APOLI's IFC was used. A simplified top-down view of this environment is illustrated in Figure 6.9. In particular, the figure shows the red object that was to be localized as well as the flight path the UAV took. The object, which was placed on a tripod, was a cylinder with a radius $r = 0.06m$ and a height of $h = 0.5m$. Furthermore, it was oriented so its base was parallel to the ground. Because the whole setup itself was not aligned to the real north direction, i.e. there was a $156.2°$ offset, for simplicity reasons the following coordinates are given in respect to a coordinate system based on the UAV Local NED coordinate system being rotated to align with a different north direction. In particular, north is then located at the top of Figure 6.9 while east is on its right side. The red object's center was defined as $\boldsymbol{c}' \coloneqq (5m, 0m, -1.75m)$. Again, the flight path of the UAV, which is also visualized in Figure 6.9, is continuous. It starts at the point $\boldsymbol{c}'_{\boldsymbol{S}} \coloneqq (0m, 0m, -1.7m)$. The UAV began by relocating to the first viewpoint being located at $\boldsymbol{c}'_{\boldsymbol{I1}} \coloneqq (0m, 5m, -1.7m)$. Afterwards, it moved to $\boldsymbol{c}'_{\boldsymbol{I2}} \coloneqq (0m, -5m, -1.7m)$. After reaching the viewpoint I2, the UAV completed

the flight path by returning to the starting position S. As a consequence, the distance between the UAV and the red object to localize ranged from $5m$ to approximately $7m$.

As the UAV of the IFC was used to gather the input datasets in this environment, the RTL's provisionally integration into the APOLI project was, once again, utilized. In particular, while the MAL directly provided the RTL with the flight data of the UAV, the object detection input was generated by the RTL itself. For this, camera images shot from a camera mounted on the UAV passed through the same color-based object detection algorithm as used in the simulation environment. The used camera was a Logitech® HD Pro Webcam C920 [56]. This camera is capable of capturing 15 megapixels RGB images at a resolution of 1920×1080 and at a framerate of up to 30Hz. However, for the input datasets generated in the outdoor environment, the camera images were downscaled to a resolution of 640×480 to achieve better runtimes. In addition to this, images can be retrieved from the C920 camera using a USB 2 interface. The camera was directly mounted on the UAV without the use of a gimbal. In particular, the camera's orientation matched the UAV's orientation at all times. Therefore, the gimbal orientation input simply matched the UAV's orientation. Finally, the UAV itself pointed at the red object throughout its flight.

As with the previous environments, seven input datasets were generated for evaluation. All of these datasets were composed of exactly 250 inputs. As only a limited variety of direction vectors between the UAV's camera and the red object to localize could be achieved using the described flight path, the number of regions generated by the Spherical Input Filter was set to 800.

Triangulation Algorithm	*Avg. RMSE [m]*	*Avg. 95th Percentile [m]*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Generalized Midpoint	~0.53	~0.64	~29,166.67	~12,733.72
L_∞	~0.69	~1.28	~811.69	~243.43
Hybrid	~0.51	~0.68	~799.45	~241.85

Table 6.6: Evaluation Data for Input Datasets generated using the Outdoor Environment.

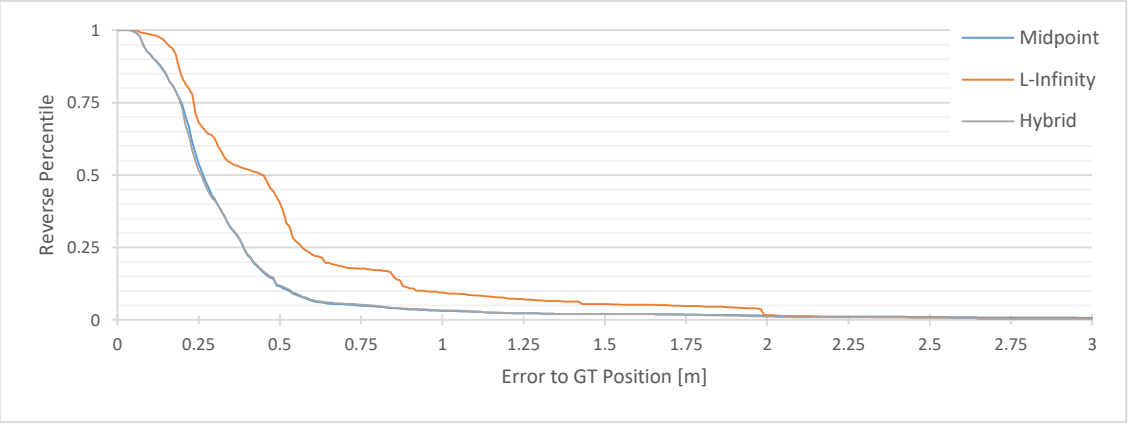


Figure 6.10: Reverse Percentile of Ground-Truth Position Errors for Input Datasets generated using the Outdoor Environment.

Some of the evaluation's results can be seen in Table 6.6 and Figure 6.10. The results demonstrate that the RTL is able to localize the red object reasonably well using any of the triangulation algorithms even when using real-world sensors to determine the input data. Obviously, such sensors are always subject to some kind of noise. As before, while object detection algorithm performed nearly perfectly, the camera image's digitalization and, subsequently, its downscaling lead to additional noise in the position of the Fundamental Points. Moreover, the influence of noise is further amplified if similar view directions are used for the triangulation. The flight path, however, only covers a $90°$ angle in front of the object. If the flight path is extended to cover more view directions, the accuracy of all triangulation algorithms is improved. In particular, the position errors resulted for the most part from inaccurate localizations in the north-south direction in Figure 6.9. Regarding the specific triangulation algorithms, it can be concluded that the L_∞ Triangulation performed the worst out of the three algorithms. Again, this is caused by its susceptibility for outliers in the input dataset. The Generalized Midpoint Triangulation and the Hybrid Triangulation, on the other hand, performed nearly identical for most of the input datasets. In particular, while the Generalized Midpoint Triangulation performed slightly better than the Hybrid Triangulation for six of the seven input datasets, there was one dataset for which the Hybrid Triangulation generated the best object localizations. However, the Hybrid Triangulation only outperformed the Generalized Midpoint Triangulation at the beginning of this input dataset where just a few Input Frames were utilized. Overall, the Generalized Midpoint Triangulation should be the preferred triangulation algorithm because of its robustness to outliers in the input as well as its low computational cost.

6.2 Ellipsoid Approximation

This chapter evaluates the RTL using all integrated ellipsoid approximation algorithms. For this, the FPD configuration providing four Fundamental Points per detected object

in the input data was being used. Because the ellipsoid approximation approach does not only localize objects, i.e. estimates their positions, but also approximates their shapes, the overlap between the localized objects' ellipsoids and their ground-truth ellipsoid is evaluated in addition to their ground-truth position errors. More specifically, to separate the error in the position from the shape error, the translated overlap as introduced in Chapter 4.7 is used. All of the ellipsoid approximation algorithms may also fail to approximate a valid ellipsoid with the given Input Frames. Therefore, the failure rate at which the localization failed to generate valid ellipsoids instead of generic quadrics is also evaluated.

To allow a direct comparison between the triangulation's and the ellipsoid approximation's evaluations, the same fundamental constants have been used. Therefore, the Spherical Input Filter with the corresponding number of regions in respect to the environment as well as a maximum selection size of 20 Input Frames was utilized. As the overlap is implemented via sampling, a value had to be chosen for the number of samplings performed. In the following, 250 samples were performed in each dimension leading to a total of 15,625,000 sample points.

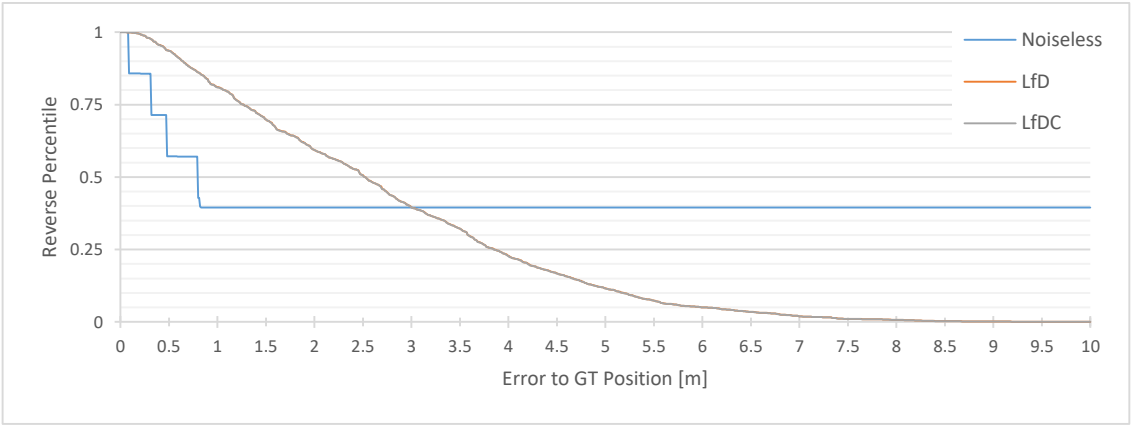
6.2.1 Synthetic Environment

The first environment used for the evaluation of the ellipsoid approximation algorithms is the synthetic environment which has been described in detail in Chapter 6.1.1. The following chapters present evaluations for the different kinds of noise applied to the dataset of the synthetic environment. In the scenario in which artificial noise was applied to the positions of the Fundamental Points, the upper-left and lower-right Fundamental Points were modified. Consequently, the positions of the upper-right and lower-left Fundamental Points were adjusted to form an axis-aligned rectangle. Again, the scenario in which no noise has been applied to the input dataset is neglected as all three integrated ellipsoid approximation algorithms provide optimal object localization results for such noiseless input.

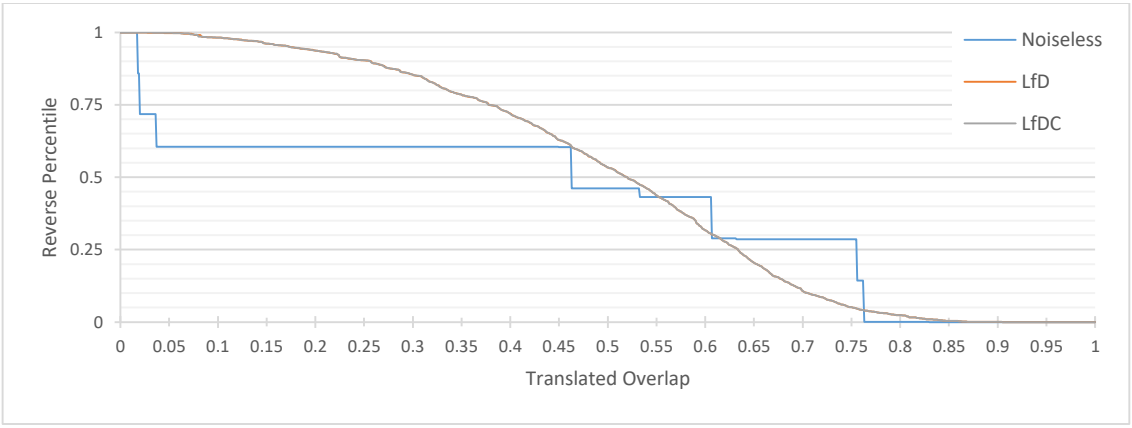
6.2.1.1 Fundamental Point Noise

Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Noiseless	~99.61%	~9.53	~60.7%	~7,927.52	~2,119.29
LfD	~30.12%	~3.26	~53.09%	~4,164.19	~1,097.52
LfDC	~30.13%	~3.25	~53.08%	~3,695.88	~968.05

Table 6.7: Evaluation Data for Input Datasets generated using the Synthetic Environment with Fundamental Point Noise.



(a)



(b)

Figure 6.11: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Synthetic Environment with Fundamental Point Noise.

For the first evaluation, artificial noise was applied to the position of the Fundamental Points by displacing them in a random direction and with a distance generated using a Gaussian distribution with $\mu = 0$ and $\sigma = 16$. Some of the evaluation results can be seen in Table 6.7 and Figure 6.11. Firstly, the Noiseless Ellipsoid Approximation was not feasible for this kind of input at all as it failed to approximate valid ellipsoids nearly at all times. More specifically, it was only able to generate valid ellipsoids at the beginning of each dataset for a few inputs, i.e. when the Input Frame dataset was barely filled with noise afflicted Input Frames. After an Input Frame with small noise was added to the Input Frame dataset, the Noiseless Ellipsoid Approximation never succeeded to localize the object again. Therefore, it cannot be used when faced with such input data as inaccurate approximations of ellipsoids then remain in the environment map for the rest of the RTL's lifespan.

The LfD and LfDC Ellipsoid Approximation, on the other hand, failed to generate valid ellipsoids only at a rate of $\sim 30\%$. In particular, their success range spanned the entirety of each input dataset making them much more feasible. Accuracy-wise, the LfDC

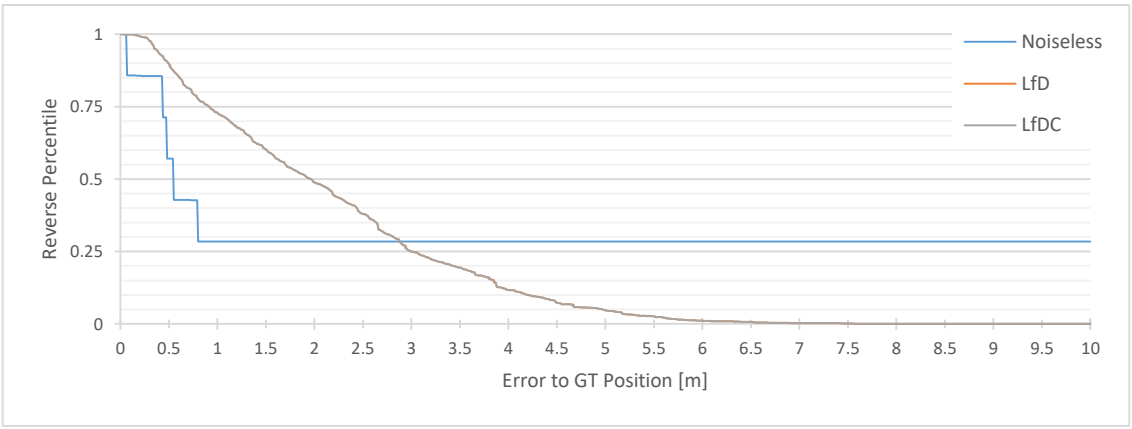
Ellipsoid Approximation only outperformed the LfD Ellipsoid Approximation marginally, both in terms of error in the position of the approximated ellipsoids as well as their translated overlap error. This is, in fact, true for all further evaluations as well. When comparing the results with the corresponding triangulation results in Chapter 6.1.1.1, it can be seen that the ellipsoid approximation algorithms generated way larger ground-truth position errors than the triangulation approach. Therefore, if only the position is of interest, the triangulation approach should be preferred. This is also true for the majority of evaluations. The accuracy of the shape of the approximated ellipsoid has been measured using the translated overlap. The results indicate a medium accuracy meaning that while it may be sufficient for some real-world applications, other systems might require more accurate ellipsoids. The exact shape of each approximated ellipsoids also changed frequently over the RTL's lifespan.

Lastly, while the LfDC Ellipsoid Approximation only leads to a small performance overhead in respect to the LfD Ellipsoid Approximation, the nearly identical accuracies of these algorithms lead to the conclusion that the LfD Ellipsoid Approximation should be the algorithm of choice on this particular input data.

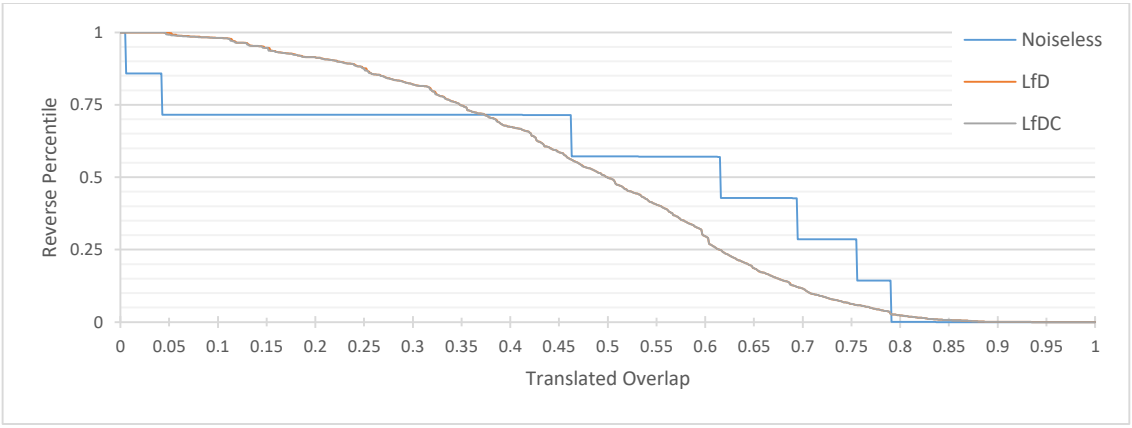
6.2.1.2 Limited Fundamental Point Noise

Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Noiseless	~99.54%	~7.62	~51.9%	~9,283.82	~2,539.91
LfD	~43.09%	~2.6	~54,87%	~4,954	~1,351.61
LfDC	~43.11%	~2.6	~54.91%	~4,450.1	~1,205.44

Table 6.8: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point Noise.



(a)



(b)

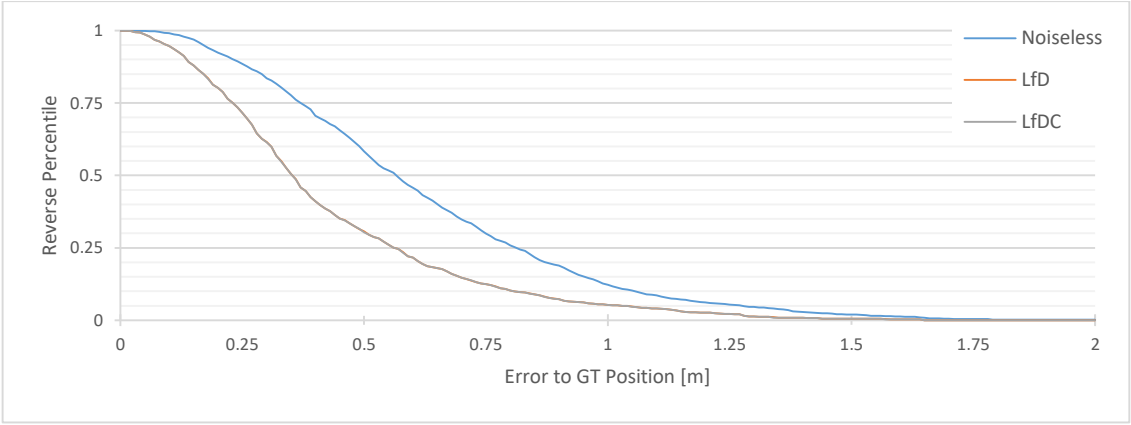
Figure 6.12: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point Noise.

For the second evaluation, the Fundamental Point Noise had been limited to a maximum displacement distance of $32 = 2\sigma$ which removed potential outliers. The results are illustrated in Table 6.8 and Figure 6.12. It can be seen that the Noiseless Ellipsoid Approximation still failed to generate valid ellipsoids at a very high rate making it unusable even if the Fundamental Point noise is limited. That being said, by removing the outliers the overall accuracy of the ellipsoid approximation algorithms improved in regards to the positional error. Once more, the LfD and the LfDC Ellipsoid Approximations generated ellipsoids of nearly identical accuracy and, also, the positional error was greater than the positional error using the triangulation approach as seen in Chapter 6.1.1.2.

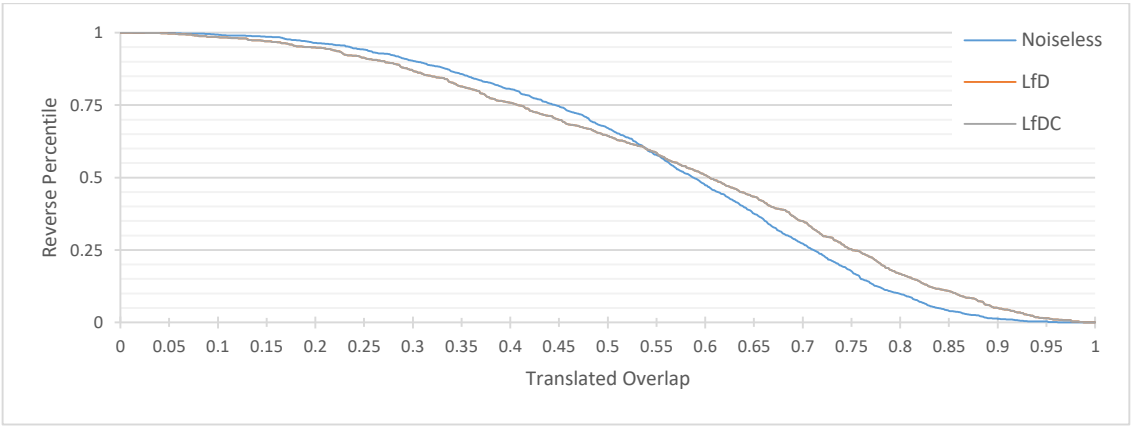
6.2.1.3 Limited UAV Position Noise

Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Noiseless	~46.74%	~0.7	~46.85%	~19,178.08	~5,902.19
LfD	~61.05%	~0.51	~47.51%	~12,006.86	~3,769.52
LfDC	~61.05%	~0.51	~47.52%	~11,475.41	~3,569.61

Table 6.9: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited UAV Position Noise.



(a)



(b)

Figure 6.13: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Synthetic Environment with Limited UAV Position Noise.

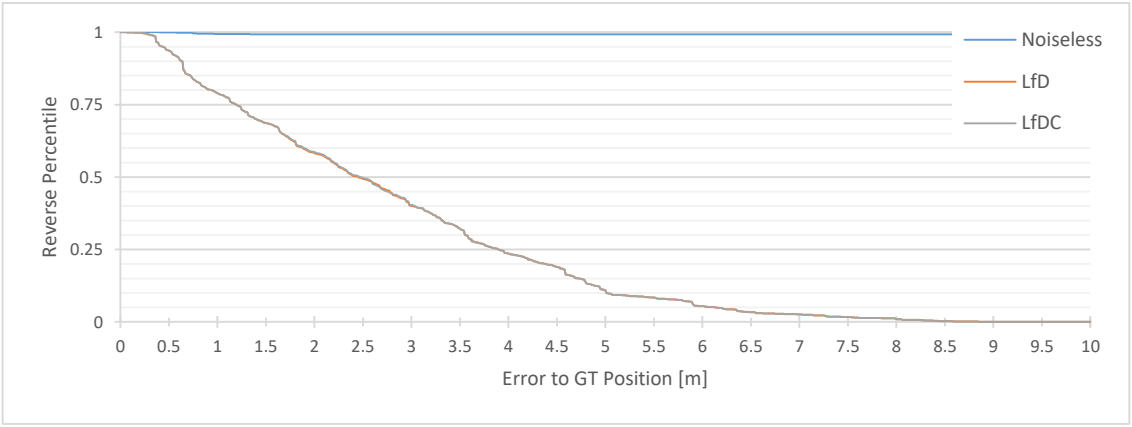
The results of the next evaluation for which limited noise was added to the UAV's positions instead of the positions of the Fundamental Point are illustrated in Table 6.9 and Figure 6.13. Again, a Gaussian distribution with $\mu = 0m$, $\sigma = 1m$ and a maximum displacement distance of $2m = 2\sigma$ was used to generate the artificial noise. The evaluation results show that the Noiseless Ellipsoid Approximation now only failed at a rate of $\sim 47\%$. This is because the Spherical Input Filter's database only consisted of a maximum of four Input Frames as the view directions of one viewpoint always stayed the same regardless of the noise. Therefore, Input Frames of the same viewpoint

constantly overwrote one another and noise afflicted input which caused the Noiseless Ellipsoid Approximation to fail before now influenced at most four localizations. That being said, the Noiseless Ellipsoid Approximation was yet again outperformed by the LfD and the LfDC Ellipsoid Approximation regarding the positional error. Shape-wise, however, it is not clear which of these algorithms performed the best. While the Noiseless Ellipsoid Approximation generated ellipsoids with the lowest translated overlap RMSE on average, the distribution of the translated overlap must also be considered. This distribution is illustrated in Figure 6.13(b). There, it can be seen that the Noiseless Ellipsoid Approximation estimated a lower percentile of ellipsoids with large shape errors but it also produced a lower number of approximated ellipsoids with low translated overlap errors. It is worth mentioning that with this kind of artificial noise, the ellipsoid approximation approach generated more accurate object positions than the triangulation. That being said, whether this holds true in other environments heavily depends on the exact environment.

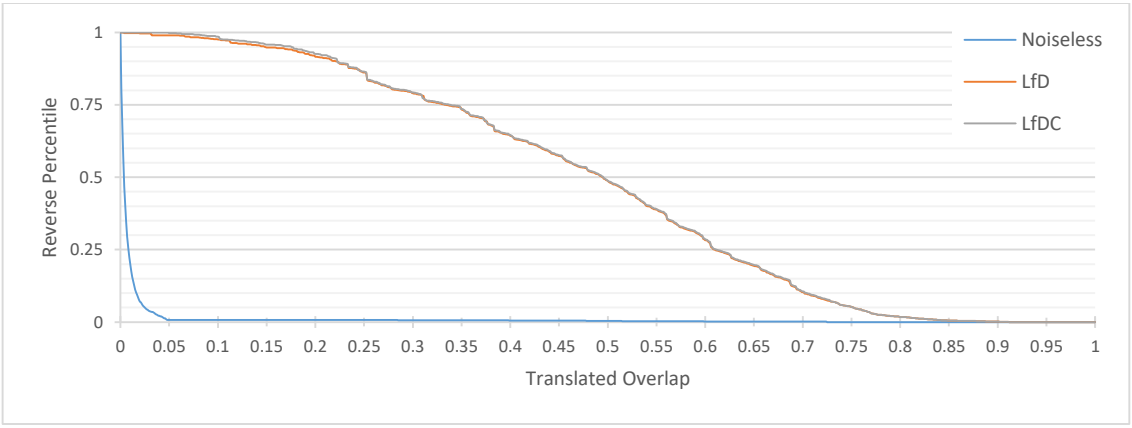
6.2.1.4 Limited Fundamental Point and UAV Position Noise

Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Noiseless	~15.5%	~313.59	~99.09%	~9,271.52	~2,530.73
LfD	~58.19%	~3.26	~55.8%	~4,950.5	~1,345.9
LfDC	~58.25%	~3.26	~55.45%	~4,396.99	~1,200.48

Table 6.10: Evaluation Data for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point and UAV Position Noise.



(a)



(b)

Figure 6.14: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Synthetic Environment with Limited Fundamental Point and UAV Position Noise.

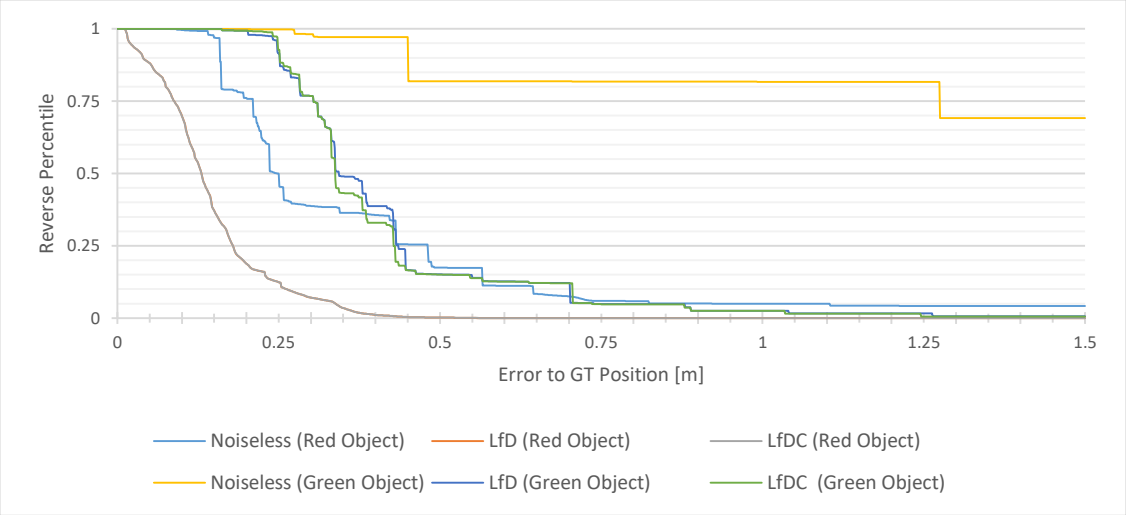
The results of the last evaluation using the synthetic environment can be seen in Table 6.10 and Figure 6.14. Here, limited artificial noise was applied to the Fundamental Point positions as well as the positions of the UAV. The results demonstrate that, while the Noiseless Ellipsoid Approximation was able to approximate valid ellipsoids at a rate of $\sim 84.5\%$, its localizations were grossly inaccurate. Not only did the Noiseless Ellipsoid Approximation generate ellipsoids with an average positional error of $\sim 313.59m$, these ellipsoids were, for the most part, way too large. On the other hand, the LfD and the LfDC Ellipsoid Approximation approximated reasonable accurate ellipsoids.

6.2.2 Simulation Environment

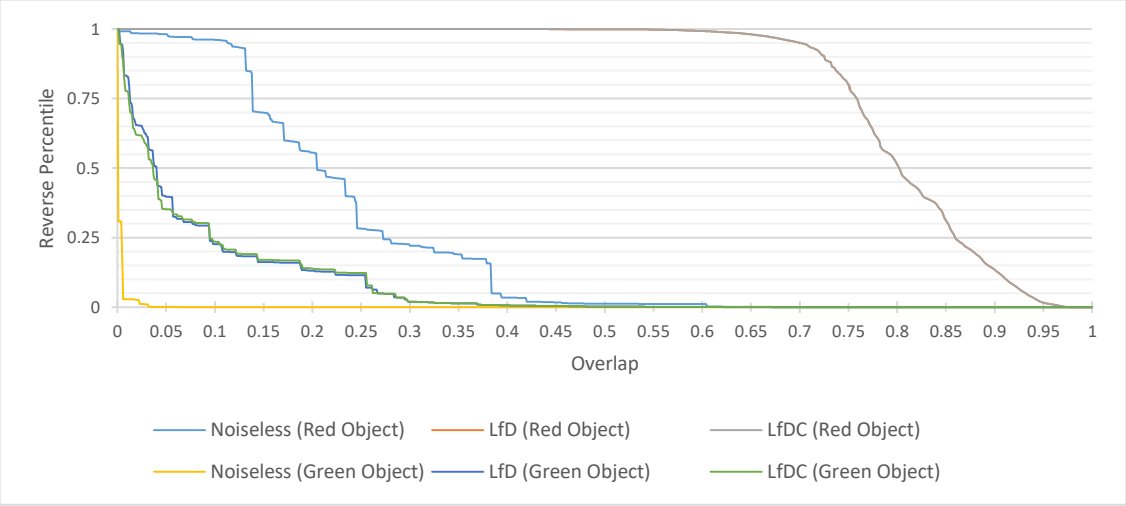
The second environment utilized to evaluate the RTL using the ellipsoid approximation approach is the simulation environment. It has been elaborated in detail in Chapter 6.1.2. This environment contained two objects that were to be localized, a larger red one as well as a smaller green one.

	Red object			Green Object		
Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*
Noiseless	~85.57%	~2.74	~77.86%	~66.98%	~4,267.89	~99.78%
LfD	~3.55%	~0.16	~20.16%	~58.05%	~0.82	~92.33%
LfDC	~3.54%	~0.16	~20.12%	~58.05%	~0.8	~92.38%

Table 6.11: Evaluation Data for Input Datasets generated using the Simulation Environment.



(a)



(b)

Figure 6.15: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Simulation Environment.

The evaluation's results are illustrated in Table 6.11 and Figure 6.15. They indicate that the Noiseless Ellipsoid Approximation was not feasible for the input datasets of this environment. Not only did the generated quadrics more often than not represent invalid ellipsoids, the accuracy of valid ellipsoids was not satisfying as well. Regarding

the red object in particular, the results show that the approximation of the object's ellipsoid was very accurate using the LfD and the LfDC Ellipsoid Approximation. These algorithms even generated ellipsoids with an, overall, lower positional error than the triangulation approach. In addition to this, the approximated ellipsoids also had high translated overlaps indicating good accuracy in terms of shape.

The same, however, cannot be said about the localizations of the smaller green object. Here, the Noiseless Ellipsoid Approximation generated ellipsoids with extremely large positional errors at times making this algorithm completely infeasible. While the positional accuracy did improve using the LfD and LfDC Ellipsoid Approximation, the shape of the object could not be estimated well at all. This was mostly caused by the object detection algorithm used for the generation of the Fundamental Points. As this algorithm used an image shot from the UAV to detect objects, the precision of the object detection was limited by the image's resolution. This was further amplified by the fact that the green object was only a few pixels wide for some of the images. This resulted in an Input Frame database containing inputs afflicted by stronger object detection noise. Therefore, whenever the LfD or the LfDC Ellipsoid Approximation were able to generate valid ellipsoids, their shape was heavily afflicted by this noise. This leads to the conclusion that the RTL using the ellipsoid approximation approach is not feasible if the object detection input is afflicted by relatively high noise in comparison to its ground-truth information. To deal with this digitalization issue, the implementation of the RTL features a configuration value effectively allowing the user to filter out object detections in images which result in axis-aligned bounding rectangles of small area which are defined by the Fundamental Points. This configuration, however, was not used for this evaluation.

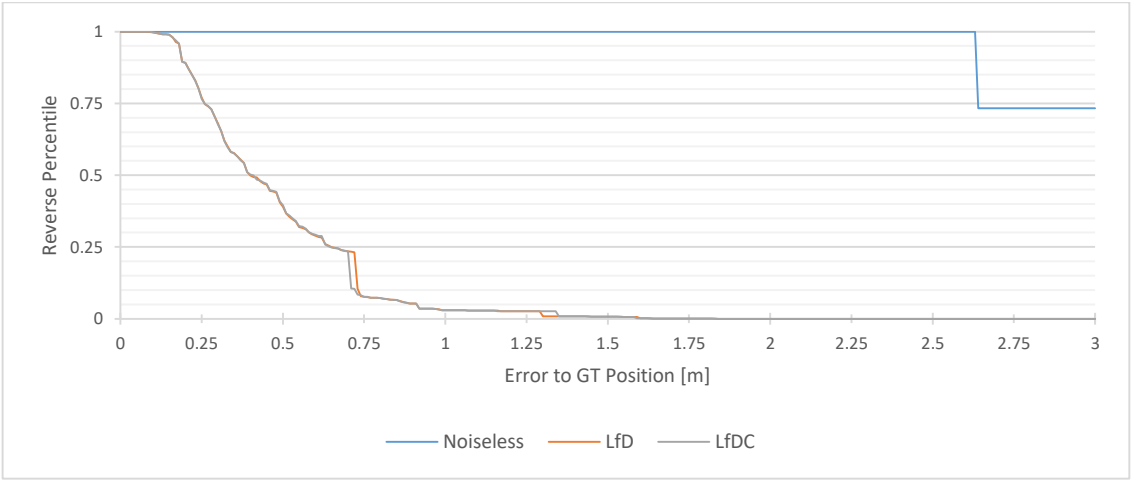
6.2.3 Outdoor Environment

Finally, the outdoor environment, which has been presented in Chapter 6.1.3, has also been used to evaluate the RTL using the ellipsoid approximation approach. For this environment, one red object was to be localized. It is worth noting that the absence of a gimbal results in some issues when using an ellipsoid approximation algorithm for object localization. In particular, in contrast to the previous environments, the camera's roll angle may be unequal to $0°$ as the camera's orientation matched the UAV's orientation precisely. As a consequence, the assumption that the axis-aligned bounding rectangle around the detected object in the image is equal to the object's bounding rectangle of minimal size does not hold true anymore. Therefore, the resulting ellipse may not represent the object in the image well. That being said, the

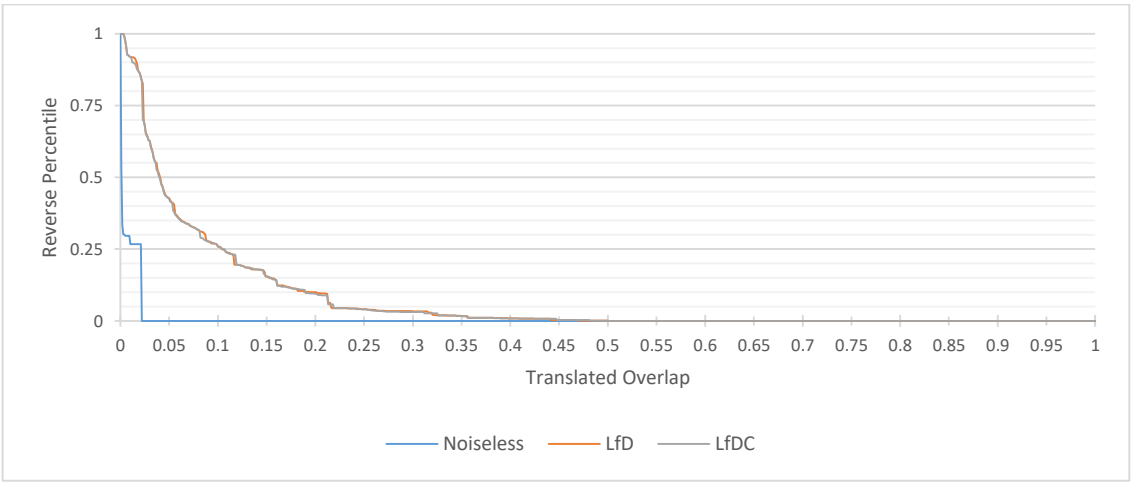
UAV was, for the most part, up straight, i.e. its absolute roll angle was no higher than 10°.

Ellipsoid Approximation Algorithm	*Avg. Failure Rate*	*Avg. Position RMSE [m]*	*Avg. Translated Overlap RMSE*	*Avg. Inputs/s (Desktop)*	*Avg. Inputs/s (ODROID)*
Noiseless	~95.74%	~234.85	~99.34%	~5,319.15	~1,493.17
LfD	~46.2%	~0.52	~92.72%	~2,892.56	~781.95
LfDC	~46.2%	~0.52	~92.79%	~2,655.54	~684.66

Table 6.12: Evaluation Data for Input Datasets generated using the Outdoor Environment.



(a)



(b)

Figure 6.16: Reverse Percentile of Ground-Truth Position Error (a) and Translated Overlap (b) for Input Datasets generated using the Outdoor Environment.

Table 6.12 and Figure 6.16 illustrate some results of the evaluation. The results show that, once more, the Noiseless Ellipsoid Approximation cannot be used for noise afflicted input data at all. It does not only have a rate of ~95.74% at which it does not generate a valid ellipsoid, the valid ellipsoids produced by the Noiseless Ellipsoid Approximation are neither accurate regarding their positions nor shapes. On the other

hand, the LfD and the LfDC Ellipsoid Approximation did not generate valid ellipsoids at a rate of $\sim 46.2\%$. As before, these algorithms performed nearly identical. Regarding their positional errors, it can be seen that the position of the object could be determined reasonably well. That being said, the triangulation approach performs better if only the object's position is of interest.

However, the shape of the object could not be approximated well for any of the ellipsoid approximation algorithms at all. For this, there are several reasons. Firstly, just like with the green object of the simulation environment, the object to localize itself was rather thin. Therefore, even small noise in any input data may have a high influence on the object's localization. Secondly, the limited direction vectors between the camera and the object amplifies the influence of noise. In fact, similarly to the observation of Chapter 6.1.3, the approximated ellipsoids are for the most part stretched in the north-south direction. By observing the red object from a greater variety of view angles, the overlap error will decrease drastically. In addition to this, the camera was not aligned to the horizon, and therefore to the object itself, anymore, i.e. the roll angle may be unequal to $0°$. This, however, can be dealt with easily by adding a gimbal which keeps the camera aligned. It can be concluded that, when attempting to localize such small objects, a greater variety of view angles is a necessity to localize the object accurately. This is especially true if the object's shape is of interest.

7 Conclusion

7.1 Summary

The objective of this master thesis was to develop an object localization algorithm to localize objects in a three-dimensional Euclidean space. In particular, this algorithm had to be applicable on UASs and, therefore, fulfill certain requirements, e.g. real-time capability and its inputs are limited to information which can be gathered via a UAS only. In particular, this localization algorithm was to be integrated into the Automated Power Line Inspection (APOLI) project. In the context of this thesis, the Real-Time Object Localizer (RTL) was developed, implemented and evaluated as such an object localization algorithm. When provided with a sufficient number of input data each consisting of some of the UAV's flight data, the gimbal orientation as well as object detection information, the RTL is able to reliably and accurately localize the detected objects. It has also been shown that this is the case even if the input data is afflicted by some noise. For this reason, the RTL is feasible to generate environment maps for the purpose of collision avoidance. However, the evaluation also demonstrated that the RTL is susceptible to outliers in the input data, i.e. input information which is either heavily afflicted by noise or fundamentally incorrect. In particular, the RTL does not integrate any technique to remove such outliers.

For the object localization itself, two approaches have been integrated into the RTL's pipeline. These approaches are the triangulation and the ellipsoid approximation. While the RTL using the triangulation approach only provides the object's position as object information output, the ellipsoid approximation generates an ellipsoid corresponding to the detected object. Therefore, not only does the RTL using this approach estimate the object's location but it also provides an approximation of the object's shape as its output. For the triangulation, three algorithms have been integrated and evaluated. These are the Generalized Midpoint Triangulation, the L_∞ Triangulation as well as the Hybrid Triangulation. The evaluations showed that not only is the Generalized Midpoint Triangulation by far the most efficient algorithm, it is also, overall, the triangulation algorithm generating the most accurate estimations of the objects' locations. Therefore, the Generalized Midpoint Triangulation should be the triangulation algorithm of choice. The ellipsoid approximation approach has also been integrated into the RTL via three algorithms, i.e. the Noiseless Ellipsoid Approximation, the LfD Ellipsoid Approximation and the LfDC Ellipsoid Approximation. Evaluations proofed that the Noiseless Ellipsoid Approximation performs poorly when dealing with input data being slightly noise afflicted. This means that not only were the object

localizations grossly inaccurate in some scenarios, the algorithm also more often than not failed to approximate valid ellipsoids at all. The LfD and the LfDC Ellipsoid Approximation, on the other hand, demonstrated to be more robust and also more accurate. While the LfDC Ellipsoid Approximation generated, in most cases, marginable more accurate ellipsoids, it also added a performance overhead when compared to the LfD Ellipsoid Approximation. For these reasons, the LfD Ellipsoid Approximation is the most feasible ellipsoid approximation algorithm for usage in UASs.

7.2 Outlook

The evaluations showed that the RTL using any of the two object localization approaches, i.e. the triangulation or the ellipsoid approximation, may heavily be affected by outliers in the input data. In particular, these outliers are stored in the Input Frame database of the Spherical Input Filter until either a similar outlier or valid input corresponding to the outlier's region is provided. Because there might never be valid input which corresponds to this region, the outlier could potentially influence all future object localizations. For the Generalized Midpoint Triangulation, this outlier may only have a limited effect if enough valid Input Frames are provided along with it as the noise will be smoothed out. However, if only a low number of valid Input Frames are given, the object localization most likely fails to generate an accurate object location. When using the LfD Ellipsoid Approximation, the outlier may potentially have an even higher influence. Not only does it decrease the accuracy of the approximated ellipsoid, the LfD Ellipsoid Approximation may fail to estimate an ellipsoid at all. For these reasons, an adequate technique to remove outliers should be integrated into the RTL. In particular, the RANSAC approach may be used.

Another potential extension of the RTL modifies the object detection input information. As a reminder, this information does not only consist of the detected object's Fundamental Points but also its label. Throughout this thesis, the object labels were always assumed to be unique, i.e. there must not be two objects possessing the same label. However, the corresponding object detector may not provide such unique object labels. Therefore, one could consider the scenario in which the object labels are removed from the input. In particular, the RTL itself now has to generate such labels. For this, it may use all or a selection of Input Frames and determine object labels using a number of localizations, i.e. RANSAC. As a high number of localizations may be carried out, the performance of the RTL, or more specifically its Object Localization step, is of crucial importance for this. Therefore, even when using the LfD Ellipsoid Approximation, the object labels could be determined via RANSAC in combination with

the Generalized Midpoint Triangulation. The performance of the object label determination can further be improved if the object detection input includes class labels for each object.

References

References of Professorship of Computer Engineering

[1] U. Tudevdagva, B. Battseren, W. Hardt, S. Blokzyl, and M. Lippmann, “UAV-based Fully Automated Inspection System for High Voltage Transmission Lines.”

[2] B. Battseren, U. Tudevdagva, and W. Hardt, “A Finite State Machine Based Adaptive Mission Control of Mini Aerial Vehicle,” *Embedded Selforganizing Systems (ESS)*, vol. 5, no. 1, pp. 14–18, 2018, doi: 10.14464/ess51225.

[3] H. Ariane, H. Reda, B. Battseren, and W. Hardt, “AREIOM: Adaptive Research Multicopter Platform,” *IFOST*, pp. 219–223, 2019.

[4] M. Stephan, B. Battseren, and U. Tudevdagva, “Autonomous Unmanned Aerial Vehicle Development: MAVLink Abstraction Layer,” *International Symposium on Computer Science, Computer Engineering and Educational Technology*, vol. 10, pp. 45–49, 2020, [Online]. Available: https://www.ibs-laubusch.de/en/publications/.

External References

[5] R. Szeliski, *Computer Vision*. London: Springer London, 2011.

[6] D. L. Milgram, “Computer Methods for Creating Photomosaics,” *IEEE Transactions on Computers*, vol. C–24, no. 11, pp. 1113–1119, Nov. 1975, doi: 10.1109/T-C.1975.224142.

[7] A. K. Jain, R. Bolle, and S. Pankanti, Eds., *Biometrics: Personal Identification in Networked Society*. Boston, MA: Springer US, 1996.

[8] M. H. Yang, D. J. Kriegman, and N. Ahuja, “Detecting faces in images: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002, doi: 10.1109/34.982883.

[9] G. Cheng and J. Han, “A survey on object detection in optical remote sensing images,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 117, pp. 11–28, Jul. 2016, doi: 10.1016/j.isprsjprs.2016.03.014.

[10] C. Plagemann, T. Müller, and W. Burgard, “Vision-based 3D object localization using probabilistic models of appearance,” *Lecture Notes in Computer Science*, vol. 3663, pp. 184–191, 2005, doi: 10.1007/11550518_23.

[11] R. I. Hartley, “Chirality,” *International Journal of Computer Vision*, vol. 26, no. 1, pp. 41–61, 1998, doi: 10.1023/A:1007984508483.

[12] Hardkernel co. Ltd., “ODROID-XU4: User Manual.” 2015, [Online]. Available: https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf.

[13] PX4 Dev Team, “3DR Pixhawk 1 Flight Controller (Discontinued).” https://docs.px4.io/v1.11/en/flight_controller/pixhawk.html (accessed Jan. 25, 2021).

[14] PX4 Dev Team, “Pixhawk 4.” https://docs.px4.io/master/en/flight_controller/pixhawk4.html (accessed Jan. 25, 2021).

[15] ArduPilot Dev Team, “ArduCopter.” https://ardupilot.org/copter/ (accessed Sep. 19, 2020).

[16] FLIR Systems, “FLIR Blackfly S USB3 BFS-U3-31S4: Datasheet.” [Online]. Available: https://flir.app.boxcn.net/s/x0r0bdsjstdkudjd2dwlxoa5fdvj2gpg/file/535140254571.

[17] FLIR Systems, “FLIR Blackfly S GigE BFS-PGE-120S: Datasheet.” [Online]. Available: https://flir.app.boxcn.net/s/mj27am7zik371ivyzv352gmt390yqt6z/file/535140060152.

[18] HD Air Studio, “HD Air Studio - aerial gimbals & drones for industrial and film use.” https://hdairstudio.com/ (accessed Jun. 25, 2020).

[19] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, *GNSS — Global Navigation Satellite Systems*. Vienna: Springer Vienna, 2008.

[20] Djexplo, “Latitude and Longitude of the Earth,” *Wikimedia Commons*, 2011. https://commons.wikimedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg (accessed Jul. 13, 2020).

[21] M. Kotsonis, “Force and moments measurements,” in *Experimental Aerodynamics*, 1st ed., S. Discetti and A. Ianiro, Eds. Boca Raton: CRC Press, 2017, pp. 429–448.

[22] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[23] A. Joglekar, D. Joshi, R. Khemani, S. Nair, and S. Sahare, “Depth Estimation Using Monocular Camera,” *International Journal of Computer Science and Information Technolgies*, vol. 2, no. 4, pp. 1758–1763, 2011.
[24] M. Knorr, *Self-Calibration of Multi-Camera Systems for Vehicle Surround Sensing*. 2018.
[25] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, “An evaluation of the RGB-D SLAM system,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, vol. 3, no. c, pp. 1691–1696, doi: 10.1109/ICRA.2012.6225199.
[26] R. I. Hartley and P. Sturm, “Triangulation,” *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, Nov. 1997, doi: 10.1006/cviu.1997.0547.
[27] P. A. Beardsley, A. Zisserman, and D. W. Murray, “Navigation using affine structure from motion,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 801 LNCS, pp. 85–96, 1994, doi: 10.1007/bfb0028337.
[28] S. Ramalingam, S. K. Lodha, and P. Sturm, “A generic structure-from-motion framework,” *Computer Vision and Image Understanding*, vol. 103, no. 3, pp. 218–228, 2006, doi: 10.1016/j.cviu.2006.06.006.
[29] R. I. Hartley and P. Sturm, “Triangulation,” *Proceedings of the ARPA Image Understanding Workshop*, pp. 957–966, 1994.
[30] J. Chen, D. Wu, P. Song, F. Deng, Y. He, and S. Pang, “Multi-View Triangulation: Systematic Comparison and an Improved Method,” *IEEE Access*, vol. 8, pp. 21017–21027, 2020, doi: 10.1109/ACCESS.2020.2969082.
[31] K. Madsen, H. B. Nielsen, and O. Tingleff, “Methods for Non-Linear Least Squares Problems (2nd ed.),” 2004.
[32] R. Hartley and F. Schaffalitzky, “L∞ minimization in geometric reconstruction problems,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, no. 1, pp. 2–7, 2004.
[33] W. P. Fox, “Fibonacci Search in Optimization of Unimodal Functions,” *Department of Mathematics Francis Marion University, Florence, SC 29501*, 2002, [Online]. Available: https://www.maplesoft.com/applications/view.aspx?SID=4193&view=html.
[34] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.
[35] C. Rubino, M. Crocco, and A. Del Bue, “3D Object Localisation from Multi-View Image Detections,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1281–1294, 2018, doi: 10.1109/TPAMI.2017.2701373.
[36] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. 2004.
[37] G. H. Golub and C. Reinsch, “Singular value decomposition and least squares solutions,” *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, 1970, doi: 10.1007/BF02163027.
[38] J. N. Franklin, *Matrix Theory*. Dover Publications, 2012.
[39] P. Rüegg-Reymond, “Classification of Quadratic Surfaces,” vol. 1, no. 5, 2012.
[40] P. Gay and A. Del Bue, “Objects Localisation from Motion with Constraints.,” *arXiv Computer Vision and Pattern Recognition*, 2018, doi: arXiv:1803.10474v2.
[41] J. Engel, J. Sturm, and D. Cremers, “LSD-SLAM: Large-Scale Direct Monocular

SLAM," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1449–1456, 2013.
[42] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015, doi: 10.1109/TRO.2015.2463671.
[43] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, 2017, doi: 10.1186/s41074-017-0027-2.
[44] B. Triggs *et al.*, *Bundle Adjustment – A Modern Synthesis To cite this version : Bundle Adjustment — A Modern Synthesis*. 2010.
[45] E. B. Saff and A. B. J. Kuijlaars, "Distributing many points on a sphere," *Mathematical Intelligencer*, 1997, doi: 10.1007/BF03024331.
[46] D. P. Hardin, T. Michaels, and E. B. Saff, "A comparison of popular point configurations on S2," *Dolomites Research Notes on Approximation*, vol. 9, pp. 16–49, 2016.
[47] B. Duvenhage, "Generating Equidistant Points on a Sphere," 2019. https://bduvenhage.me/geometry/2019/07/31/generating-equidistant-vectors.html (accessed Sep. 21, 2020).
[48] B. Duvenhage, K. Bouatouch, and D. G. Kourie, "Numerical verification of bidirectional reflectance distribution functions for physical plausibility," *ACM International Conference Proceeding Series*, no. January 2015, pp. 200–208, 2013, doi: 10.1145/2513456.2513499.
[49] M. Galassi *et al.*, *GNU Scientific Library Reference Manual (3rd Ed.)*. 2009.
[50] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
[51] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, 2017, [Online]. Available: https://arxiv.org/abs/1705.05065.
[52] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.
[53] D. Herberth, "glad," *GitHub repository*. GitHub, 2020, [Online]. Available: https://github.com/Dav1dde/glad.
[54] M. Geelnard and C. Berglund, "GLFW Reference Manual," 2010.
[55] G-Truc Creation, "OpenGL Mathematics (GLM)," *GitHub repository*. GitHub, 2020, [Online]. Available: https://github.com/g-truc/glm.
[56] Logitech, "Logitech HD Pro Webcam C920 für Windows, Mac und Chrome OS." [Online]. Available: https://www.logitech.com/de-de/product/hd-pro-webcam-c920.
[57] E. C. Carter *et al.*, "CIE 015:2018 Colorimetry, 4th Edition," Vienna, Oct. 2018. doi: 10.25039/TR.015.2018.

A. Implementation Details

In this appendix, some implementations are elaborated in greater detail. In particular, while Chapter A.1 presents more in-depth descriptions of some of the `RTL`'s steps' implementations, some of the modifications implemented in the MAL are introduced in Chapter A.2.

A.1 Steps of the implemented Real-Time Object Localizer

A.1.1 Input Gatherer

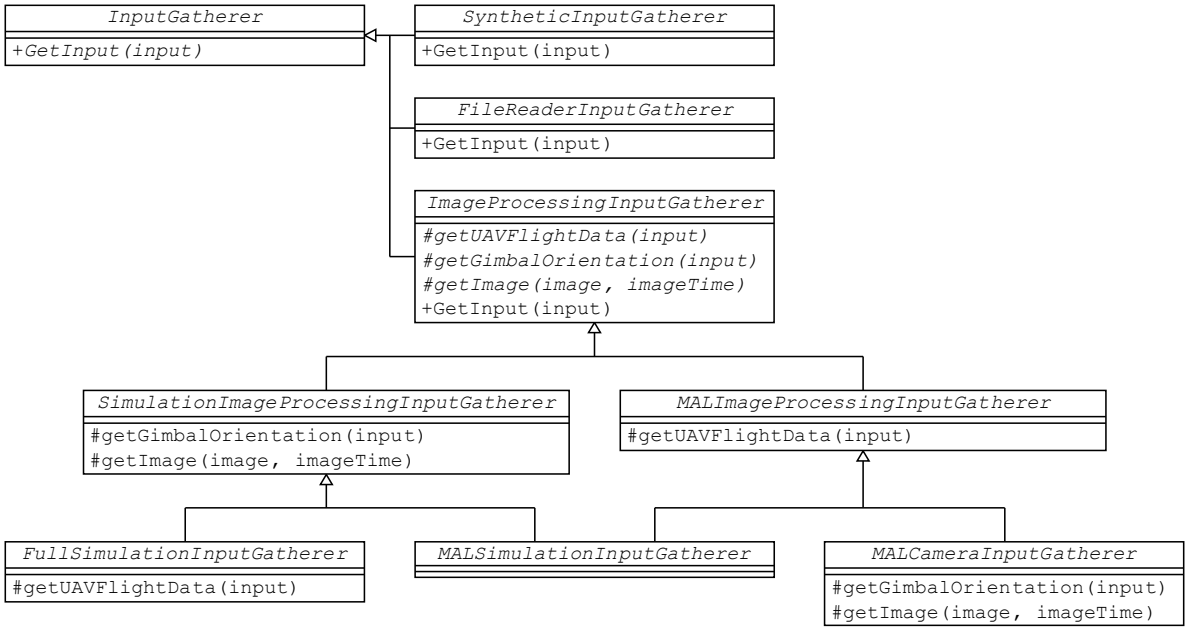


Figure A.1: Simplified UML Class Diagram of the Input Gathering Step.

In the first step of the RTL, input is gathered from one or more sources which is specified via the `Configuration` object. Depending on the corresponding configuration value, the RTL allocates a specific subclass of the `InputGatherer` class. Figure A.1 illustrates the fundamental abstract subclasses of `InputGatherer`. The only logic the final subclasses, i.e. the classes which are not further inherited in the figure, are missing depends on the FPD configuration. Therefore, for each of these classes two subclasses exist, one for the triangulation and one for the ellipsoid approximation approach. It should be noted that when using the ellipsoid approximation approach the number of Fundamental Points per object in the object detection input of the `RTL` is two, rather than four, in contrast to the convention used in this thesis. More specifically, while the software's first Fundamental Point denotes the upper-left corner of the bounding rectangle, the second Fundamental Point specifies the lower-right corner. However, there occurs no information loss as the four

Fundamental Points generated by the FPD result in an axis-aligned bounding rectangle.

As can be seen in the figure, there are three top-level input sources. The input can either be synthetically generated via `SyntheticInputGatherer`, read from an input dataset file using `FileReaderInputGatherer` or generated with `ImageProcessingInputGatherer`. In particular, the Image Processing Input Gatherer `ImageProcessingInputGatherer` class itself only implements logic to generate the object detection input from an image, i.e. detecting objects in the image by generating their Fundamental Points and the objects' labels. For this, a pixel mask is determined by performing a simple color range check. For each object to detect, the `Configuration` provides the object's color, a maximum color distance, the object label as well as a minimum contour area. Note that the color range check is not performed in the RGB color space but instead in the so-called CIELAB color space [57] in which the light intensity is more appropriately separated from the object's color itself. After the pixel mask has been computed, the Image Processing Input Gatherer generates contours around accumulations of positive mask entries and selects the contour of the largest area. By using these contours, noise and outliers are removed. Furthermore, the contour of the largest area is only utilized for further calculations if its area is sufficiently large, i.e. not smaller than the minimum contour area provided. If, instead, the area is too small, the corresponding object is not detected in the image. This process prevents incorrect object detections of small area. Finally, the minimal-sized axis-aligned bounding rectangle is determined around the contour of largest area which is then used to determine the exact positions of the Fundamental Points.

Subclasses of `ImageProcessingInputGatherer` provide the rest of the input data, i.e. the UAV flight data using `getUAVFlightData()` and the gimbal orientation input via `getGimbalOrientation()`, as well as the image itself through `getImage()`. In particular, `SimulationImageProcessingInputGatherer` uses the AirLib library to connect to the SITL simulation of APOLI. Using this connection, this class captures images directly shot from the simulated UAV and also provides the gimbal orientation input as the orientation of the virtual camera used to capture the image. `MALImageProcessingInputGatherer`, on the other hand, connects with the MAL by opening a shared memory block. As the MAL writes all UAV flight data input into this memory block, the `MALImageProcessingInputGatherer` uses this information and provides it as the UAV flight data input directly.

Finally, the three classes on the lowest level of Figure A.1 make use of these connections to fill out the rest of the input data. More specifically, `FullSimulationInputGatherer` utilizes the position and orientation of the virtual camera to generate the UAV flight data. Next, `MALSimulationInputGatherer` combines both connections. This means that the UAV flight data input is retrieved from the MAL while the gimbal orientation and the object detection input are generated from simulation data. Lastly, `MALCameraInputGatherer` captures the image from a physical camera connected to the system directly and the gimbal orientation is further set to match the UAV's orientation exactly.

A.1.2 Input Filter

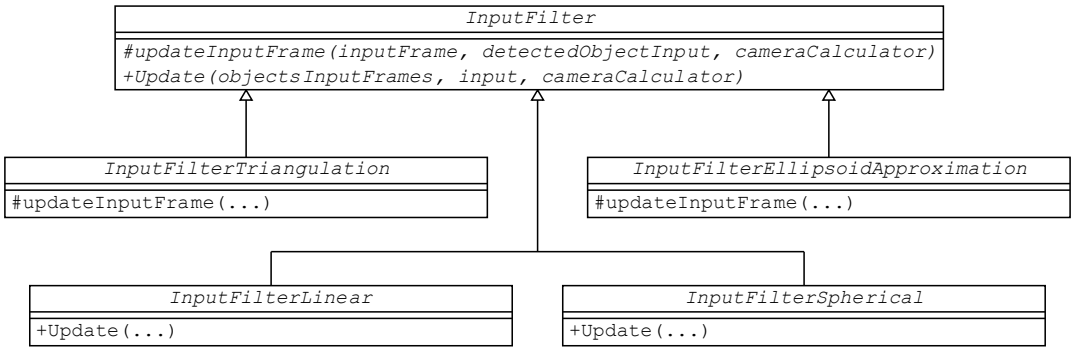


Figure A.2: Simplified UML Class Diagram of the Input Filtering Step.

The Input Filtering step of the RTL is implemented via the `InputFilter` class and its subclasses. A simplified model of their architecture is illustrated in Figure A.2. The `InputFilter` base class defines two abstract functions. Firstly, the `Update()` function is used to iterate over all detected objects in the input data. For each of these objects, it creates a new Input Frame database corresponding to this object, if it does not already exist. Subsequently, it generates or retrieves the exact Input Frame which shall be updated using the input data and calls `updateInputFrame()`. This function updates a single Input Frame of an object by setting its values.

As elaborated in Chapter 4.5, the Input Frames could theoretically consist only of the input data as well as the camera parameters. However, the subclasses corresponding to the two localization approaches, i.e. `InputFilterTriangulation` and `InputFilterEllipsoidApproximation`, preprocess this information as much as possible to improve the performance of the Object Localization step. In particular, the exact data stored in the Input Frames depends not only on the used localization approach but also the exact localization algorithm. For the triangulation approach, the Input Frames always contains the camera matrix $P \in \mathbb{R}^{3\times4}$ as well as preprocessed

information for the Generalized Midpoint Triangulation, i.e. the starting point $\boldsymbol{c} \in \mathbb{R}^3$ of the casted ray as well as the ray's direction $\boldsymbol{d} \in \mathbb{R}^3$. This data can easily be determined using the camera matrix P. If the L_∞ Triangulation or the Hybrid Triangulation is selected, the position of the object's Fundamental Point is also contained in the Input Frame. If, however, the ellipsoid approximation approach is the localization approach of choice, the preprocessing is more complex. It begins by always determining preprocessed information used for the Noiseless Ellipsoid Approximation. In particular, two variables are determined being the matrix $G \in \mathbb{R}^{6\times10}$, as defined in [35], as well as the negative of the vectorized dual space ellipse, i.e. $-\hat{\boldsymbol{c}} \in \mathbb{R}^6$. If either the LfD or the LfDC Ellipsoid Approximation algorithm is selected via the configuration, further information is computed. This data is the negative of the vectorization of the normalized dual space ellipse $-\hat{\boldsymbol{c}} \in \mathbb{R}^6$ as well as $H^{-1}P \in \mathbb{R}^{3\times4}$ which is a component of the calculation of the substituted camera matrix $\dot{P} \in \mathbb{R}^{3\times4}$. In the software, $H^{-1}P$ is referred to as the half-transformed camera matrix.

As the process of selecting the Input Frame to update depends on the Input Filtering algorithm used, the `Update()` function is implemented in `InputFilterLinear` and `InputFilterSpherical`. These classes provide an implementation of the Linear Input Filter and the Spherical Input Filter respectively. As a reminder, while the Linear Input Filter is simple and leads to nearly no overhead, it amplifies the influence of noise drastically. For this reason, this Input Filter should not be used and the `InputFilterLinear` class was mainly created for debugging purposes. Both of these classes maintain an Input Frame database which itself consists of multiple Input Frame databases, one for each object that has been detected in the past. While the number of the object's Input Frames is set to the number of Input Frames to select for the Linear Input Filter, the Spherical Input Filter instead allocates memory proportional to the number of regions as defined by the configuration. The selection of the Input Frame to update is implemented via a ring buffer for the Linear Input Filter. The Spherical Input Filter, on the other hand, has to iterate over all regions and determine the one region for which its vector has minimal distance to the normalized view direction vector pointing from the camera to the center of the detected object. This region corresponds to exactly one Input Frame which is the Input Frame to update. Lastly, a selection of Input Frames which shall be further used for the Object Localization step is made. For the Linear Input Filter, this is trivial as it simply selects all Input Frames. For the Spherical Input Filter, however, the number of regions may be higher than the maximum number of Input Frames to select. Therefore, it selects the appropriate number of Input Frames by maximizing the minimal distance between their indices. To further improve the runtime of the RTL, if the selection of Input Frames

using the Spherical Input Filter does not include the Input Frame which has been updated, the Input Frame selection is not passed to subsequent steps preventing a localization of this object because the newly gathered input is not included.

Finally, the Linear and the Spherical Input Filter each possess two subclasses, one for each localization approach. These subclasses inherit from both, the Linear or the Spherical Input Filter class respectively as well as from `InputFilterTriangulation` or `InputFilterEllipsoidApproximation`. Therefore, a total of four non-abstract Input Filter classes exist, e.g. `InputFilterSphericalTriangulation`.

A.1.3 Output Processor

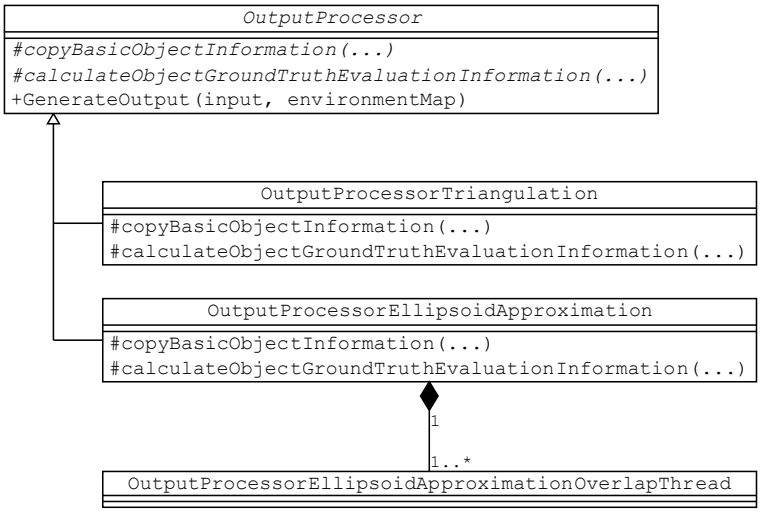


Figure A.3: Simplified UML Class Diagram of the Output Processing Step.

After the `ObjectLocalizer`, or more specifically its subclasses, determined the localization of objects and updated the environment map accordingly, the `OutputProcessor` generates the output of the RTL. Again, Figure A.3 illustrates a simplification of the architecture of `OutputProcessor` and its subclasses. In particular, the output is generated using the `GenerateOutput()` function. The format of the RTL's output depends on the localization approach used as the localization of an object may not only consist of its position but also its shape when using the ellipsoid approximation approach. To generate the output information corresponding to this basic object information, the abstract `copyBasicObjectInformation()` function is used. The implementations of this function in the two subclasses of Output Processor, each corresponding to one of the localization approaches, simply copy the localization

information of objects from the object in the environment map into the output object provided.

Furthermore, the `calculateObjectGroundTruthEvaluationInformation()` function determines ground-truth evaluation information if desired, i.e. if specified by the configuration and if there is ground-truth information about the localized objects. In particular, for the triangulation approach this ground-truth evaluation information only consists of the distance between the ground-truth position and the localized object's position which is derived in a trivial manner. This ground-truth evaluation information is also computed for the ellipsoid approximation approach where the position of an object is considered its center's position. As elaborated in Chapter 4.2.3, however, the ground-truth evaluation output also consists of two ellipsoid overlaps if the ellipsoid approximation is used for localization, i.e. $o(Q, Q')$ and $o(Q, \overline{Q'})$. In particular, these overlaps are determined by sampling discrete points. As a high sampling rate is desired, which however impacts the performance of the Output Processing step drastically, the computation of the overlaps between two ellipsoids is outsourced to threads of the `OutputProcessorEllipsoidApproximationOverlapThread` class. In particular, the Output Processor of the ellipsoid approximation approach allocates a number of objects of this thread class matching the number of concurrent threads the platform supports. To decrease thread creation overhead, these threads stay allocated until the `RTL` is stopped. That being said, they change into the blocked state whenever they finished calculating their corresponding share of the overlap. When every thread finished determining their share, the Output Processor of the ellipsoid approximation simply accumulates the sample results.

A.1.4 Renderer

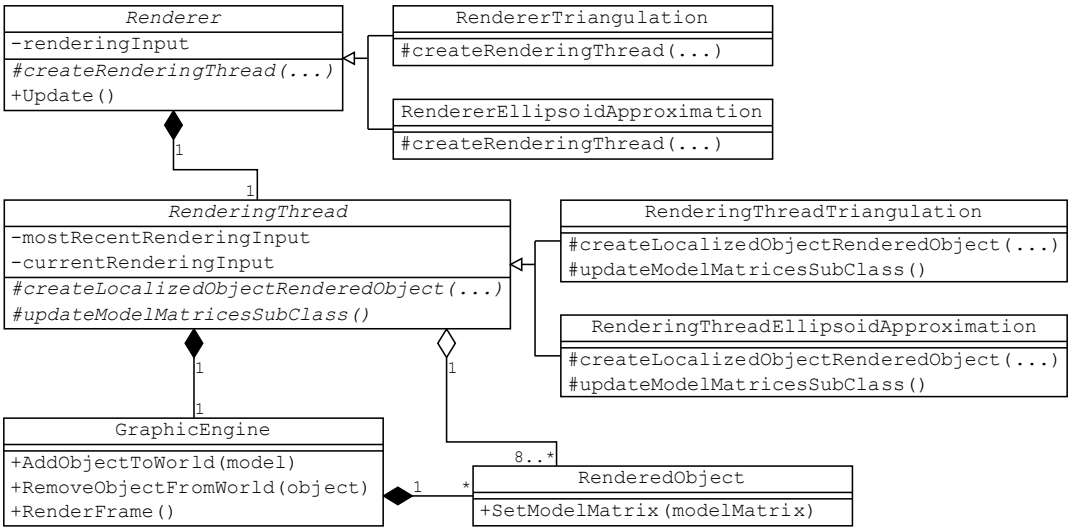


Figure A.4: Simplified UML Class Diagram of the Rendering Step.

As the last step of the RTL and if specified by the configuration, the rendered scene and its environment is updated via `Renderer::Update()`. Figure A.4 visualizes a simplified architecture of the `Renderer`, its subclasses alongside some additional classes used for the rendering process. The `Renderer` class itself does not implement much logic. Instead, the rendering and the corresponding logic is implemented in a separate thread, the `RenderingThread`. For this reason, the subclasses of Renderer simply specify which subclass of `RenderingThread` to allocate. By outsourcing the rendering process to a separate thread, not only is the rendered scene more responsive to user interaction but the performance impact on the `RTL` itself is also minimized. In particular, the `Renderer::Update()` function simply notifies the `RenderingThread` about updated rendering input by copying its own rendering input into the `mostRecentRenderingInput` member variable of the thread. The rendering input of the Renderer itself is updated by the `RTL` and some of its steps earlier. The Rendering Thread renders the scene in a loop whereby it copies the value of its `mostRecentRenderingInput` member into the `currentRenderingInput` variable which will then be used for the rendering itself. This allows the Renderer to copy its rendering input into `mostRecentRenderingInput` nearly at all times further reducing the performance impact on the `RTL`.

For the low-level rendering, a simple graphic engine has been implemented via the `GraphicEngine` class. It allows to add, manipulate and remove objects from the scene easily. In particular, when being provided with an object model, it creates and

returns an object of the `RenderedObject` class which represents the added object in the scene. Using this class, the scene object can be manipulated.

After updating its `currentRenderingInput` variable, the Rendering Thread begins by updating the RTL's output object information of all localized objects. While doing so, it also checks for additional as well as deleted objects in the rendering input, i.e. objects which are not present in the scene. Note that currently neither the RTL nor any of its steps delete objects from the environment map. As a result, objects which were present in the RTL's output once stay localized. However, future extensions of the `RTL` may lead to such scenarios in which objects are removed from the output of the RTL. Therefore, the implementation of all steps of the `RTL` are able to deal with such cases. Whenever the `RenderingThread` detects newly localized objects in the RTL's output, it calls the `createLocalizedObjectRenderedObject()` function which creates a `RenderedObject` scene object corresponding to the localized object. Analogously, if there are objects in the scene which are not present in the output of the RTL anymore, they are deleted from the scene. Next, the Rendering Thread updates the model matrices of all rendered objects. For the rendered objects corresponding to localized objects, the `updateModelMatricesSubClass()` function is used. Subsequently, the `RenderingThread` also handles user input and updates the virtual camera of the rendered scene. Finally, the scene is rendered using the `GraphicEngine` class by calling `GraphicEngine::RenderFrame()`.

A.2 Modification of the MAVLink Abstraction Layer

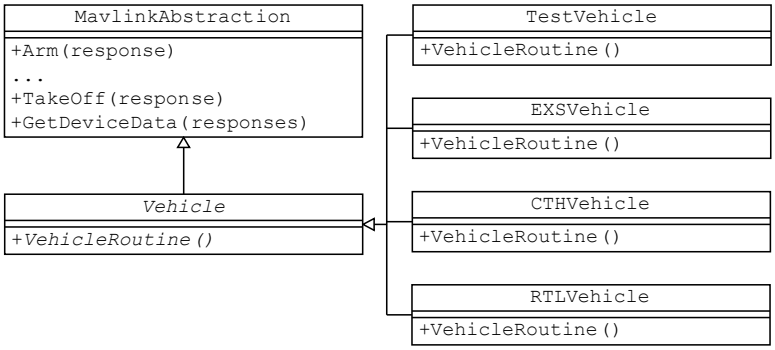


Figure A.5: Simplified UML Class Diagram of the MAL.

The MAVLink Abstraction Layer (MAL) of the APOLI project acts a software-in-the-middle between the higher-level decision-making components and the lower-level hardware-controlling components of APOLI. Such an abstraction layer simplifies the usage of the complex MAVLink protocol, which is used for the communication with the lower-level components, significantly. An overview of the MAL's implementation prior

to any modifications is given in [4]. In particular, the class `MavlinkAbstraction` and its members' classes implement the complete abstraction. As there are different use cases for the MAL, the `MavlinkAbstraction` class is further inherited to the `Vehicle` class. In the context of the MAL, a vehicle simply denotes a specialization of `MavlinkAbstraction` for a specific use case. Originally, there were three such vehicles. These were the Test Vehicle, used for testing and debugging purposes, the EXS Vehicle, which communicates with the Expert System of APOLI, as well as the CTH Vehicle demonstrating how the communication with APOLI's Control Handler shall be implemented. To select the vehicle which is to be used, the `--vehicle` or `-v` argument is utilized. This argument indicates that the next argument specifies the name of the vehicle to start. In particular, `test`, `exs` and `cth` were the corresponding vehicle names. In the following, the two most important modifications implemented in the MAL are elaborated. A simplified overview of the MAL's architecture with these modifications applied is illustrated in Figure A.5.

Firstly, as none of the preexisting vehicles were feasible for usage with the RTL, a fourth vehicle has been added. In particular, this is the RTL Vehicle implemented in the `RTLVehicle` class. To start the MAL using this vehicle, the vehicle name `rtl` must be used. Upon being started, the RTL Vehicle requests the timestamp indicating when the FLC was started, given as the number of microseconds since the UNIX epoch, as well as the position of the origin of the UAV Local NED coordinate system in GNSS coordinates. Note again that, currently, this is the home position. In the near future, however, a release of ArduCopter will most likely change the origin of the UAV Local NED coordinate system to be set only once when the FLC boots. As soon as the FLC is flashed with a release which correctly fixes this issue, the implementation of the MAL's request which retrieves the origin of the UAV Local NED system should be adjusted accordingly. In particular, a `TODO` comment has been added in the source code of the corresponding class which also specifies the necessary modifications. After the initial information has been retrieved from the FLC, the MAL executes the vehicle's routine, i.e. `RTLVehicle::VehicleRoutine()`, in a loop. In its routine, the `RTLVehicle` requests information about the UAV's position in the UAV Local NED system as well as its orientation. Once this information has been retrieved, the RTL Vehicle gathered all data of the RTL's UAV flight data input. This information is then written into a shared memory block so it is available to the RTL. Afterwards, the vehicle routine is executed anew.

Next, to retrieve data from the FLC, the MAL only provided a single abstraction function. In particular, `MavlinkAbstraction::GetDeviceData()` requested all

flight data there was from the FLC. While this was done so all flight data was available for further evaluation as one object, it led to overhead in most cases as the user will rarely use all of the retrieved flight data. This issue was already acknowledged in [4]. Because the flight data provided to the RTL should be as synchronous as possible, this problem has been addressed resulting in the second modification. For this, the parameter of the `GetDeviceData()` function has been changed to a vector of responses. In particular, these responses inherit the abstract `ResponseGetDeviceDataBase` class. Now, each of the objects in the response vector specify which flight data shall be requested from the FLC via their own request implemented as a subclass of `RequestGetDeviceDataBase`. As a result, the user can now easily request only a subset of flight data from the FLC by defining vector of those responses corresponding to the flight data they require. Therefore, the overhead is minimized.

B. Configuration Files

B.1 Structure of Configuration Files

```
[INPUT]
{
    source = synthetic

    [SYNTHETIC]
    {
        input_amount = 2000
    }
}
```

Figure B.1: Snippet of a Configuration File.

The format of configuration files used by the implemented software of the RTL is inspired by the INI format. Therefore, it contains keyword-value pairs separated by the = sign, i.e. `keyword=value`. As a convention, the names of keywords only consist of lower-case characters. Each of these keyword-value pairs corresponds to exactly one section which is defined by its name. In contrast to INI files, however, a section does not only start via `[SECTION]`, whereby `SECTION` stands for the section's name, but it also explicitly defines a scope. More specifically, a line only consisting of `{` denotes the start and a single `}` on a line stands for the end of the scope. This allows for the most crucial distinction to INI files. The sections in configuration files can be structed hierarchically which means that a section may contain an arbitrary number of subsections improving the readability of configuration files. A snippet of a configuration file is illustrated in Figure B.1. There, the `INPUT` section is comprised of the keyword-value pair `source=synthetic` as well as the `SYNTHETIC` section which itself contains another keyword-value pair, i.e. `input_amount=2000`. The naming convention used for section names defines them as fully capitalized. That being said, the parsing of configuration files is case-insensitive meaning that section names, the names of keywords and even some of the values' names may be capitalized arbitrarily. Furthermore, the parsing process also ignores empty lines, leading and trailing whitespace characters as well as whitespace characters around the = sign of keyword-value pairs. Lastly, a comment may start at any position in a line via the `#` sign and lasts until the end of that line.

B.2 Preexisting Configuration Files

The software's resources also include a number of preexisting configuration files. These are

- `full_simulation_ellipsoid_approximation`,

- `full_simulation_triangulation`,
- `mal_simulation_ellipsoid_approximation`,
- `mal_simulation_triangulation`,
- `mal_camera_ellipsoid_approximation`,
- `mal_camera_triangulation`,
- `development` and
- `template`.

The `full_simulation…` configuration files are set up to use the RTL in the SITL simulation of APOLI. Furthermore, the input is retrieved from the simulation only. This stands in contrast to the `mal_simulation…` files where the input is gathered from the simulation and the MAL. Next, the configuration files starting with `mal_camera…` are intended to be used together with the UAV of the Indoor Flight Center (IFC) of APOLI. Here, the input is retrieved from the MAL and the object detection input is generated using camera images. Finally, `development` and `template` are auxiliary configurations. While `development` was used in the development process, `template` contains a complete list of all configuration sections and their keyword-value pairs each paired with a detailed description. For the keyword-value pairs, this description also contains the range of allowed values as well as the default value.

C. User Interactions with the Scene

Key Bind	*Action*
`Escape`	Closes the window of the rendered scene
`P`	Pauses/resumes the update process of the rendered scene
`W`	Moves the camera forwards
`S`	Moves the camera backwards
`A`	Moves the camera leftwards
`D`	Moves the camera rightwards
`Page Up`	Moves the camera upwards
`Page Down`	Moves the camera downwards
`Up Arrow`	Rotates the camera upwards
`Down Arrow`	Rotates the camera downwards
`Left Arrow`	Rotates the camera leftwards
`Right Arrow`	Rotates the camera rightwards
`Space`	Increases the movement/rotation speed while being hold
`1`	Toggles the visibility of the UAV
`2`	Toggles the visibility of the UAV Local NED coordinate system
`3`	Toggles the visibility of the UAV NED coordinate system
`4`	Toggles the visibility of the UAV coordinate system
`5`	Toggles the visibility of the Gimbal NED coordinate system
`6`	Toggles the visibility of the Gimbal coordinate system
`7`	Toggles the visibility of the Camera coordinate system
`8`	Toggles the visibility of the localized objects
`9`	Toggles the visibility of the ground-truth objects
`0`	Toggles the visibility of the ground plane

Table C.1: Key Binds and their corresponding Actions available for User Interaction with the rendered Scene.

The user can interact with the rendered scene of the software implementation in various ways. For a majority of these user interactions, keyboard input is used. A complete list of key binds as well as their corresponding action is illustrated in Table C.1. The camera movement is performed at a speed of $1m/s$ while its increased speed, i.e. while holding `Space`, is $5m/s$. Analogously, the rotation speed is $90°/s$ and $180°/s$ respectively. It should be emphasized that pausing or resuming the update process by pressing `P` does not pause or resume the RTL nor does it pause/resume the rendering process itself. Instead, by pausing using `P` the content of the rendered

scene’s environment is simply not updated which allows to inspect specific localizations more easily. In addition to key binds, the user may also directly click on object localizations in the scene using the mouse cursor. By doing that, the output information corresponding to the clicked object is printed to the console.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

CSR-14-01 International Summerschool Computer Science 2014, Proceedings of Summerschool 7.7.-13.7.2014, Juni 2014, Chemnitz

CSR-15-01 Arne Berger, Maximilian Eibl, Stephan Heinich, Robert Herms, Stefan Kahl, Jens Kürsten, Albrecht Kurze, Robert Manthey, Markus Rickert, Marc Ritter, ValidAX - Validierung der Frameworks AMOPA und XTRIEVAL, Januar 2015, Chemnitz

CSR-15-02 Maximilian Speicher, What is Usability? A Characterization based on ISO 9241-11 and ISO/IEC 25010, Januar 2015, Chemnitz

CSR-16-01 Maxim Bakaev, Martin Gaedke, Sebastian Heil, Kansei Engineering Experimental Research with University Websites, April 2016, Chemnitz

CSR-18-01 Jan-Philipp Heinrich, Carsten Neise, Andreas Müller, Ähnlichkeitsmessung von ausgewählten Datentypen in Datenbanksystemen zur Berechnung des Grades der Anonymisierung, Februar 2018, Chemnitz

CSR-18-02 Liang Zhang, Guido Brunnett, Efficient Dynamic Alignment of Motions, Februar 2018, Chemnitz

CSR-18-03 Guido Brunnett, Maximilian Eibl, Fred Hamker, Peter Ohler, Peter Protzel, StayCentered - Methodenbasis eines Assistenzsystems für Centerlotsen (MACeLot) Schlussbericht, November 2018, Chemnitz

CSR-19-01 Johannes Dörfelt, Wolfram Hardt, Christian Rosjat, Intelligente Gebäudeklimatisierung auf Basis eines Sensornetzwerks und künstlicher Intelligenz, Februar 2019, Chemnitz

CSR-19-02 Martin Springwald, Wolfram Hardt, Entwicklung einer RAD-Plattform im Kontext verteilter Systeme, März 2019, Chemnitz

Chemnitzer Informatik-Berichte

CSR-19-04 Johannes Götze, René Schmidt, Wolfram Hardt, Hardwarebeschleunigung von Matrixberechnungen auf Basis von GPU Verarbeitung, März 2019, Chemnitz

CSR-19-05 Vincent Kühn, Reda Harradi, Wolfram Hardt, Expert System for Adaptive Flight Missions, Juni 2019, Chemnitz

CSR-19-06 Samer Salamah, Guido Brunnett, Christian Mitschke, Tobias Heß, Synthesszing gait motions from spline-based progression functions of controlled shape, Juni 2019, Chemnitz

CSR-19-07 Martin Eisoldt, Carsten Neise, Andreas Müller, Analyse verschiedener Distanzmetriken zur Messung des Anonymisierungsgrades θ, Juni 2019, Chemnitz

CSR-19-08 André Langer, Valentin Siegert, Martin Gaedke, Informationsverwertung basierend auf qualitätsoptimierten semistrukturierten Datenbeständen im Wachstumskern "LEDS", Juli 2019, Chemnitz

CSR-20-01 Danny Kowerko, Chemnitzer Linux-Tage 2019 - LocalizeIT Workshop, Januar 2020, Chemnitz

CSR-20-02 Robert Manthey, Tom Kretzschmar, Falk Schmidsberger, Hussein Hussein, René Erler, Tobias Schlosser, Frederik Beuth, Marcel Heinz, Thomas Kronfeld, Maximilian Eibl, Marc Ritter, Danny Kowerko, Schlussbericht zum InnoProfile-Transfer Begleitprojekt localizeIT, Januar 2020, Chemnitz

CSR-20-03 Jörn Roth, Reda Harradi, Wolfram Hardt, Indoor Lokalisierung auf Basis von Ultra Wideband Modulen zur Emulation von GPS Positionen, Februar 2020, Chemnitz

CSR-20-04 Christian Graf, Reda Harradi, René Schmidt, Wolfram Hardt, Automatisierte Kameraausrichtung für Micro Air Vehicle basierte Inspektion, März 2020, Chemnitz

CSR-20-05 Julius Lochbaum, René Bergelt, Time Pech, Wolfram Hardt, Erzeugung von Testdaten für automatisiertes Fahren auf Basis eines Open Source Fahrsimulators, März 2020, Chemnitz

CSR-20-06 Narankhuu Natsagdorj, Uranchimeg Tudevdagva, Jiantao Zhou, Logical Structure of Structure Oriented Evaluation for E-Learning, April 2020, Chemnitz

CSR-20-07 Batbayar Battseren, Reda Harradi, Fatih Kilic, Wolfram Hardt, Automated Power Line Inspection, September 2020, Chemnitz

CSR-21-01 Marco Stephan, Batbayar Battseren, Wolfram Hardt, Object Localization in 3D Space for UAV Flight using a Monocular Camera, März 2021, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz