Deutsches Zentrum
für Luft- und Raumfahrt

Hochschule Bremerhaven

Georeferencing and Spatial Representation of Two-Dimensional SONAR Images

Bachelor Thesis

Presented to gain Admission to
Bachelor of Science at the University of Applied Sciences Bremerhaven

Author: Lea Meyer, Bürgermeister-Smidt Str. 161, 27568, Bremerhaven
Matr.Nr.: 34468

Submission Date: 02/July/2020

Supervisor: Prof. Dr. Axel Bochert, University of Applied Sciences Bremerhaven
Second Supervisor: Dr. David Heuskin, German Aerospace Center

Affidavit

I, Lea Meyer, declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Date	Signature

Abstract

The presented thesis deals with the augmented visualization from a two- to three-dimensional scene of visualised backscattered signals, captured by a two-dimensional multi-beam BlueView P900-130 *Sound Navigation And Ranging* (SONAR), as well as its georefencing, against the background of harbour risk management and sheet pile wall examination.
Alongside the data acquisition and preparation a simple methodology for combining depth values, motion parameters, underwater positioning data and two-dimensional SONAR images is presented, in order to provide an economically priced alternative to high-quality, but also upscale three-dimensional multi-beam SONAR. By means of the provided sensors systems by the German Aerospace Center DLR, Institute for the Protection of Maritime Infrastructures, a measurement trial in the Fischereihafen 1 (Bremerhaven, Germany), was realised using concrete reference objects in the accuracy range of the SONAR.
The resulting program structure of data acquisition, preprocessing and final visualization is concerted to the utilized systems and can be handled by instructed person with a minimum of programming knowledge.
Regarding the outcome of the measurement trial a lack of precision in the three-dimensional representation along the depth, due to the large vertical coverage of the SONAR, can be observed. Also a floating of the sensor carriers acoustic position occurred throughout the trial, on account of interferences between the acoustic sensors.

Tabel of Contents

Abbreviations

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AUV	Autonomous Unterwater Vehicle
COG	Course Over Ground
CTD	Conductivity Temperature Depth - Logger
DIP	Dual In-line Package
DLR	German Aerospace Center
DOM	Dissolved Organic Matter
DVL	Doppler Velocity Log
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IP	Internet Protocol
JSON	JavaScript Object Notation
LBL	Long Baseline
MAD	Median Absolute Deviation
MAVLink	Micro Air Vehicle Communication Protocoll
MI	Institute for the Protection of Maritime Infrastructures
Military Grid Reference System	Military Grid Reference System
NMEA	National Marine Electronics Association
PGM	Portable Grey Map
PLY	Polygon File Format
POI	Point Of Interest
PSU	Practical Salinity Unit
QGC	QGroundControl
ROV	Remotely Operated Vehicle
SBL	Short Baseline
SDK	Software Development Kit
SOG	Speed Over Ground
SONAR	Sound Navigation And Ranging
TC	Topside Computer
TCP	Transmission Control Protocol

UART	Universal Asynchronous Receiver
UAV	Autonomous Underwater Vehicles
UDP	User Datagram Protocol
UGPS	Underwater GPS - Underwater positioning system of the company Water Linked AS
URL	Uniform Resource Locator
USB	Universal Serial Bus
USBL	Ultra Short Baseline
UTM	Universal Transverse Mercator
UUV	Unmanned Underwater Vehicle
WGS 84	Word Geodetic System 1984

Symbols

β	Angular distribution of energy	$\mathrm{dB} \cdot {}^{\circ -1}$
ρ	Surface reflection	dB
ϕ	Roll-angle	$^{\circ}$
θ	Pitch-angle	$^{\circ}$
ψ	Yaw-angle	$^{\circ}$
μ	Mean value	–
α	Critical value	–
μ_0	Mean value of population	–
H_0	Null hypothesis	–
H_1	Alternative hypothesis	–
S	Salinity	ppt
T	Temperature	°C
Z	Z value	–
c	Sound speed	$\mathrm{m} \cdot \mathrm{s}^{-1}$
n	Number of samples	–
p	Empirical quantile	–
s	Standard deviation of sample	–
x	x-direction	m
x_i	Independent normally distributed random variables	–
y	y-direction	m
z	Depth	m

1. Introduction

At the beginning of this thesis the underlying motives, the task as well as the rough course of the work will be outlined. The topic itself was developed in cooperation with the *German Aerospace Center* (DLR) *Institute for the Protection of Maritime Infrastructures* (MI), department 2, group Technology Testing, which is specialized in the application, development and supervision of remote controlled and autonomous vehicles, as a sensor carrier for the acquisition of environmental parameters above and below water. Due to the safety and security background, especially the aspect of harbour sheet pile wall inspection in the German Bight can be seen as the background of this work. Throughout the course of the thesis it was possible to draw on the institute's resources, material and premises.

1.1. Motivation

Harbour sheet pile walls are designed for a lifetime between 50 and 80 years [20, p.39], but may not reach this, due to corrosion and ship collisions. Especially in the low-water zone of tidal ports, pitting and shallow pit corrosion occur, which reduce the sheet pile walls thickness over the course of time and lead in the worst case to a perforation. Same applies to seabed-mounted offshore structures, for example energy parks or oil platforms. Therefore regular inspection and maintenance is necessary to ensure a continuous and uninterrupted course of port business or system operation. Underwater imaging, using optical systems, is a widely spread method for inspection and maintenance in waters, with a good visibility range and minimal sediment content. For low-visibility, which is often found in estuaries and seas with high sediment content, due to tidal movement, a possibility of object detection by means of haptic inspection by divers is rather used instead of optical sensors. This bears the disadvantage of dangerous working conditions and long operations on the divers side. On the offshore business side, a cost-intensive deployment through the use of ships as a base station, transport and, if necessary, accommodation of the operational forces, needs to be expected. Furthermore, the location of the diver and therefore the damaged area is solely based on plumb line measurements, which is dependent on a fixed reference point and detailed documentation by the operation supervisor. Ultimately the inspection depends on the experience and skills of the divers.
The additional use of acoustic systems, such as a multi-beam imaging *Sound Navigation And Ranging* (SONAR) , may facilitate inspection by locating the damaged area before the actual dive. With these systems it is further possible to cover large areas and gain precise information about surface elevation. Three-dimensional capable SONAR systems in the six-figure range form the upper end of the price category are therefore often not affordable for small educational facilities or commercial enterprises. A simplified methodology and measurement setup to acquire spatial, structural information below surface with reasonable priced equipment, using for example a two-dimensional capable SONAR, would provide an increase in knowledge and learning in the educational sector and serve as an additional assistance for divers in port and offshore inspection.

Even though the DLR MI already operates an automated sensor carrier *Autonomous Underwater Vehicle* (AUV), a versatile manoeuvrable sensor carrier with mobility in six degrees of freedom, such as a *Remotely Operated Vehicle* (ROV), expands the range of application in small and narrow areas of investigation and further reduces the amount of operating personal, due to its compact size and lightweight frame. Thus the integration of acoustic sensors and further development and improvement of the ROV is expedited by the institute.

1.2. Task Description

The given task comprises the data acquisition of two-dimensional forward-looking multi-beam SONAR imagery data, the depth captured by a pressure sensor, motion parameters of an ROV, functioning as the sensor carrier, and global positioning data of an underwater acoustic positioning system. All captured data should then be merged, respectively to the SONAR images and visualized using a point cloud representation. For the latter filtering methods regarding the reduction of noise, as well as image smoothing should be applied. Further a correct spatial alignment of the point cloud should be enhanced by a movement compensation as well as correct positioning using additional offsetting from the local and global coordinates of the sensor carrier.

1.3. Thesis Outline

The thesis is divided into three parts: the first part deals with the presentation of principles and technical background regarding underwater acoustics. The second part includes an introduction to the hardware and software being used. Following, in the third part, the implementation, measurement execution and the results are presented. The evaluation of the measurement outcome and recommendations for further work will conclude the thesis.

At the outset the main principles of underwater acoustics (s. sec. 2) are explained and an overview of the functionality and image formation of multi-beam SONAR systems will be given. In the following section the impact of noise and reverberation in water will be explained and possible solutions for its reduction, by means of filter algorithms, will be mentioned. From there on an introduction to georeferencing by short baseline positioning system will be given, followed by the coordinate system being used for data acquisition and the application of a geometric median, in order to evaluate the positioning data quality. The second part is split into a hardware and a software part, spanning over two sections (s. sec. 3 and sec. 4). In the former all to be used systems will be presented, which includes the sensor carrier BlueROV2 (BlueRobotics)[6], the *Underwater GPS* (UGPS) (Water Linked)[49] and the multi-beam SONAR BlueView P900-130 (BlueView Technologies) [11]. In the latter an in depth overview will be given to the software and corresponding self-written programs, being used in the implementation (s. sec 5). Therefore the data flow between the single programs will be explained in section 4.1. The data acquisition of each system will be intensively described in section 4.3, followed by image processing tools in section 4.4. Finally the section is closed with the topic of data fusion and the three-dimensional representation (sec. 4.5). The third part (s. sec. 5) deals with the data acquisition process with self-constructed test objects in a harbour environment. Measurement results of the individuals systems, as well as the fusion of the data are presented and evaluated. The conclusion and proposes to future work will close the thesis in section 6.

2. Principles and Application of Underwater Acoustics

Visual data acquisition in water causes many challenges towards science. With increasing depth not only the over all light penetration diminishes, but also the wavelength distribution lessens. Figure 2.1 shows the light penetration in fresh- or brackish water with a small amount of *Dissolved Organic Matter* (DOM) and sediment (a) and freshwater with a large proportion of DOM, sediment and plankton (b). There are significant differences between both waters. In the topmost layers of 2.1 (a) all wavelength between 400 nm and 680 nm can penetrate and decreases almost linearly toward the spectrum between 400 nm and 550 nm. The red light with wavelength between 650 nm and 680 nm penetrates the water up to 7 m and then decreases continuously. With increasing depth the water will therefore appear to look more greenish, as this wavelength dominates in depth of 25 m and below. The depth in 2.1 (b) is compared to 2.1 (a), with 3 m is significantly smaller. The light penetration from all wavelength between 400 nm and 600 nm ends at around 1 m. A higher penetration amount of the wavelength between 600 nm and 680 nm can be noticed. Combined with the information of high amount of DOM and sediment can be concluded that the water will appear reddish and darker. The visual range in water itself also depends on

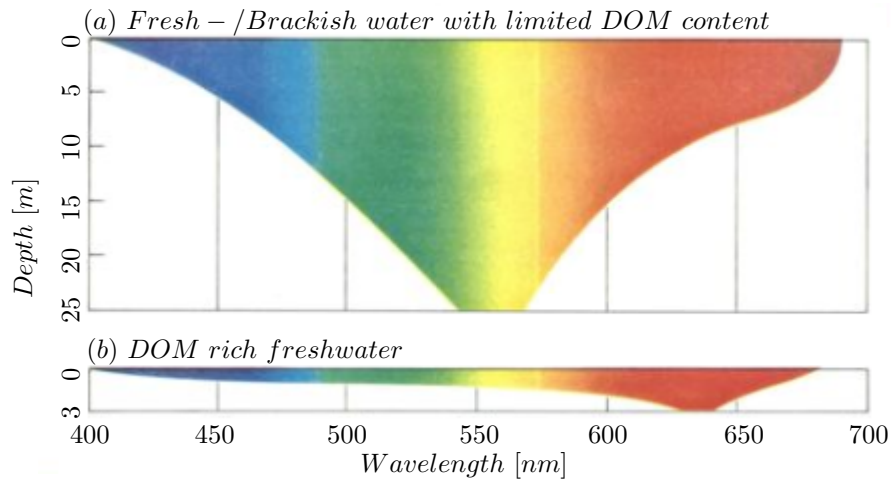


Figure 2.1.: Light penetration in water, adapted from [28]

the sea floor characteristics and suspended matter in the water column, which is caused by biological and anthropogenic effects, such as algae bloom, high population of fish, fertilizer entry (all summarized under the term DOM) and suspension by high boat traffic, but also natural effects such as sedimentary transport by rivers into estuaries as well as tidal movement. Especially in coastal regions, where a high amount of suspended matter is to be expected, optical data acquisition is most likely not applicable, due to limited underwater visibility up to a maximum of 1 m (s. annex A). The application of acoustic waves is an alternative to optical methods and was studied extensively during the first and second world wars research in the investigation of acoustic methods, which are applied and further developed today in oceanography, maritime engineering, naval navigation and geophysical exploration [32].

The starting point for generating valid data from acoustical measurements is the examination of acoustic waves, which are specified as the energy propagation through water [23, p. 1ff.]. By knowing the frequency and wave length, the speed of sound, which is dependant upon temperature, salinity and depth, can be derived. The salinity is usually given in *Practical Salinity Unit* (PSU), which corresponds to parts per thousand. It can be roughly differentiated between marine ocean water ranging from 30 PSU to 35 PSU, brackish water ranging from 0.5 PSU to 29 PSU and fresh water usually around 0.5 PSU. The temperature is dependent on location, season and time of day, the water depth is in the most cases only roughly differentiated in open ocean and coastal waters. However, in order to guarantee the quality of acoustic signals, it is necessary to determine the sound speed in the water, id est the transmission of the acoustic impulse, according to the environmental parameters.

The speed of sound can be estimated to lay in the range of 1446 $\frac{\mathrm{m}}{\mathrm{s}}$ to 1540 $\frac{\mathrm{m}}{\mathrm{s}}$, depending on the type of water [14, p. 13f.]. In estuaries and adjacent ports a low-salinity, shallow waters and moderate temperature are to be expected. Equation 2.1 shows the calculation of sound speed according to Del Grosso and Mackenzie [18]. It is valid for $-2\,^{\circ}\mathrm{C} \leq T \leq 30\,^{\circ}\mathrm{C}$, $25\,\mathrm{PSU} \leq S \leq 40\,\mathrm{PSU}$ and $0\,\mathrm{m} \leq z \leq 8000\,\mathrm{m}$.

$$\begin{aligned} c = 1448.96 + 4.591T - 0.05304T^2 + 0.0002374T^3 + 1.34(S-35) + 0.0163z + \\ 1.675 \cdot 10^{-7} z^2 - 0.01025T(S-35) - 7.139 \cdot 10^{-13} Tz^3 \end{aligned} \tag{2.1}$$

From eq. 2.1 it can be concluded that the temperature influences the sound velocity the most, followed by the salinity and finally the depth. A precise knowledge of the water temperature should be targeted through an appropriately accurate sensor. For the salinity impact the equation 2.1 includes two terms. The first terms, $1.34(S-35)$, impact on the sound speed ranges between -13.4 $\frac{\mathrm{m}}{\mathrm{s}}$ to 0 $\frac{\mathrm{m}}{\mathrm{s}}$, when assuming salinities of 25 PSU to 35 PSU. The other term of salinity and temperature, $-0.01025T(S-35)$, results in a factor between 0.0205 to -0.3075 (for temperatures in the range of $-2\,^{\circ}\mathrm{C}$ to $30\,^{\circ}\mathrm{C}$) to which the impact of the salinity is then multiplied. This leads to a maximal impact of -0.2747 $\frac{\mathrm{m}}{\mathrm{s}}$ to 4.1205 $\frac{\mathrm{m}}{\mathrm{s}}$ for the second term on the sound speed.
Following the impacts of differing salinity levels will be presented. In figure 2.2 the speed of sound, defined by eq. 2.1, is shown according to temperature, depth and varying salinity levels. With rising salinity, here ranging from 20 PSU to 34 PSU, the speed of sound increases by approximately 1.24 $\frac{\mathrm{m}}{\mathrm{s}}$.

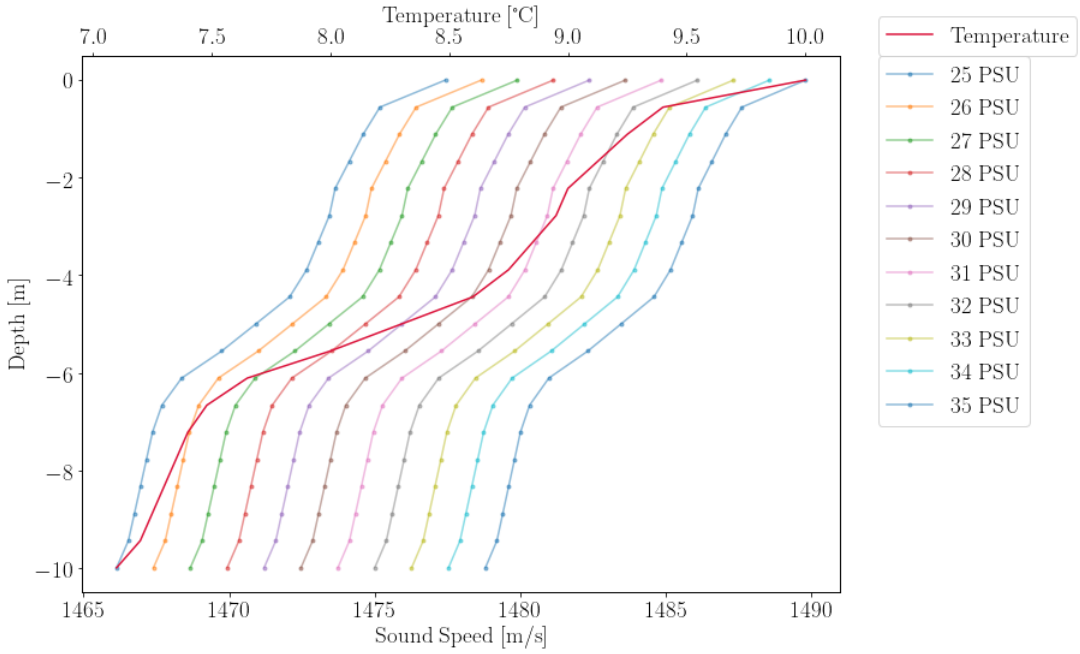


Figure 2.2.: Sound speed Profile in shallow water, according to temperature, salinity and depth

For sending and receiving acoustical waves the sound velocity plays and important role in calibrating acoustic sensors, as deviations in the speed of sound lead to incorrect data or uninterpretable images. Knowledge of the environmental parameters, temperature, salinity and depth, is therefore mandatory.

To acquire image data by using hydro acoustics, specialised active or passive sensors are required, the most commonly category is the *Sound Navigation And Ranging* (SONAR). SONAR describes a technique in detecting and localizing objects underwater. The systems used can be roughly categorized in active and passive SONARs. The active SONAR transforms electric signals to acoustic waves, which is in principle a pressure difference, and listen for the backscattered signal. From the time delay between emitting and receiving, the distance to the object can then be derived. The emitting of an acoustic impulse and the received intensity values are described in the literature as a ping. Active SONAR systems are used for various application such as inspection and maintenance, navigation/obstacle avoidance, in the fishing industry and for geophysical exploration. The active SONAR can be differentiated in echo sounder, which are mainly used vertically and "forward-looking" SONAR systems, which operate horizontally. Regarding the latter especially multi-beam SONAR are used for inspection of underwater structures. These emit sound waves in a fan shape and allow therefore a large swath area to be surveyed. Section 4.3.3 deals intensively with the image formation of such SONAR system. Passive SONARs are only listening for echo transported through the water and used in order to detect for example underwater earth quakes/tsunamis or to observe aquatic mammals (whales/dolphins). The received signal of a SONAR is often distorted by noise and reverberation, caused by acoustic signals from ships or reflection of noise from suspended matter, which will be discussed in section 2.2.

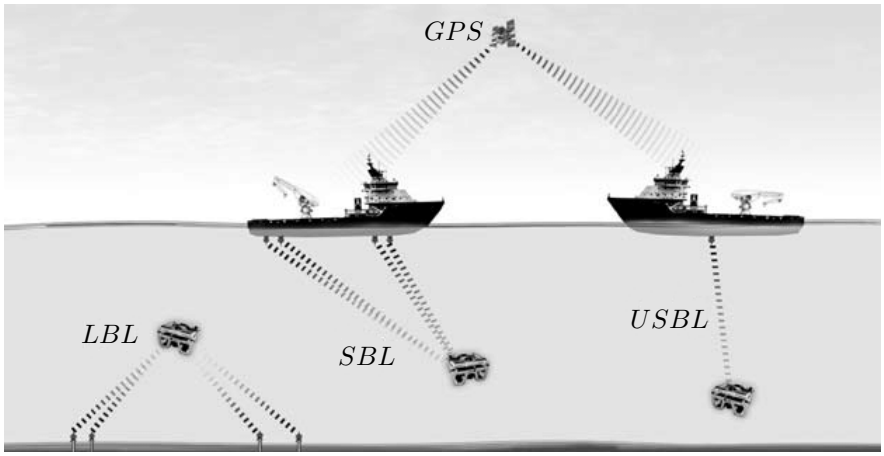


Figure 2.3.: Acoustic navigation systems using *Long Baseline* (LBL), *Short Baseline* (SBL) and *Ultra Short Baseline* (USBL), according to [39]

As already mentioned, acoustic waves are also used for naval navigation. A new area of research and application is the usage for underwater vehicles, which includes scientific used AUV and ROV. Whereas autonomous vehicles allow through their hydrodynamical shape an even data acquisition along large distances, the ROVs advantage lies in their hovering capabilities, achieved through multiple lateral oriented thruster, which can be used to inspect areas from different angles but constant height. An accurate positioning of theses systems is necessary to ensure a consistent and accurate representation of the generated data, by the installed sensors, of underwater surrounding.
Navigation of these vehicles is mostly dependent on a combination of systems. At subsea no reception of *Global Positioning System* (GPS) is possible, due to the absorption of electromagnetic waves from the water. Hence there are different methods to estimate the vehicles position by complex algorithms. An *Inertial Navigation System* (INS) calculates the position from the last known position at the surface, using the vehicles speed and flow parameters. Due to changing currents and drifts this method alone is prone to errors. Therefore further acoustic methods can correct the calculated position and help to

achieve an accurate position over long distances. Acoustic baseline positioning systems work as follows: a transmitter on e.g. an ROV sends a signal which is received by hydrophones submerged in the water, spatially distant from each other. The local position to the receiver can then be calculated through the time difference between sending and receiving. Through measuring the time delays or phase shift of the received signal by the different hydrophones the orientation of the transmitter can be derived. Therefore a relative position of the ROV to the hydrophones is achieved. [45, p. 6f.]
In order to derive the global position, the distance between the hydrophone and a GPS antenna needs to be known and added to the derived local position. Underwater positioning systems can be differentiated in three main categories according to their distance capability as can be seen in fig. 2.3. LBL systems use a transponder network on the sea floor and is used for applications where an accuracy of 1 m or less is necessary. Due to difficult placement of the transponder on the sea floor it is rather inconvenient. An SBL systems consist of three or more connected receivers submerged from a ship or platform, s. fig. 2.3 ship in the centre. The accuracy is dependent from the receiver spacing, but is also used for high accuracy dependent applications. An USBL system consists of a transducer array submerged on a pole and towed behind a ship, s. fig. 2.3 left. The accuracy for this system is rather moderate and therefore used for in a close-ship operation.
For the determination of depth on most underwater vehicles an external depth sensor is used just to access the depth information, due to higher accuracy. In section 2.3 the principle of a short baseline system will be explained in depth.

2.1. Multi-beam SONAR Image Formation

A multi-beam SONAR captures through a hydrophone array multiple narrow beams of $\pm 1^{\circ}$ across a horizontal fan shaped area, specified as the field-of-view. This allows a large area to be surveyed. Hereinafter the multi-beam SONAR, will be referred to by the abbreviation SONAR, if not otherwise stated. The emitter generates a beam pulse from an electric signal and the receiver records the local pressure variation and transforms this back into an electric value. The distance of the reflected object to the SONAR can be calculated using the time delay and the speed of the signal.
A point P_s in a SONAR image with the intensity I_s can either be described in spherical or Cartesian coordinates, as shown in fig. 2.4. The transformation between both coordinate systems follows eq. (2.2a) and respectively (2.2b).

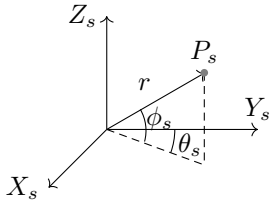


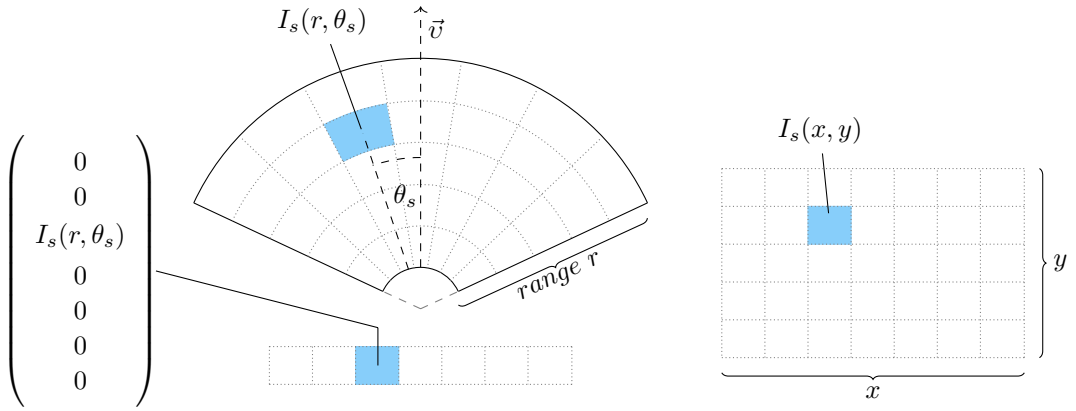
Figure 2.4.: SONAR coordinate system

$$P_s = \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} = r \begin{bmatrix} \cos(\phi_s)\sin(\theta_s) \\ \cos(\phi_s)\cos(\theta_s) \\ \sin(\phi_s) \end{bmatrix} \tag{2.2a}$$

$$P_s = \begin{bmatrix} r \\ \theta_s \\ \phi_s \end{bmatrix} = \begin{bmatrix} \sqrt{X_s^2 + Y_s^2 + Z_s^2} \\ \arctan(\frac{X_s}{Y_s}) \\ \arctan(\frac{Z_s}{({X_s}^2 + {Y_s}^2)}) \end{bmatrix} \tag{2.2b}$$

The range r is dependant upon the sound velocity c as a function of time for the signal to return form the reflective object with $r = \frac{ct}{2}$. In fig. 2.5 the sonar image formation is schematically shown, according to

[19]. A shown in 2.5 (a) all received intensity values $I_s(r, \theta_s)$ are saved in vector format for each transducer array element. $I_s(r, \theta_s)$ is itself dependant upon the range r and the angle between the received intensity value I_s, the direction vector $\vec{v}$ and the angle ϕ_s. The angle ϕ_s describes the vertical beam width of the SONAR, which usually lies around 20°. With the implementation of the third dimension, the intensity value can be described as the integral of the reflective intensities in the range between ϕ_{s1} and ϕ_{s2}, with β as the beam pattern (angular energy distribution) of the SONAR and the reflectors albedo V_s in eq. 2.3 by [19]. In the example in fig. 2.5 only the blue marked area holds a value other than zero, which is saved at the transducer number, which corresponds to the angle θ_s, and corresponding row in an intensity vector. From these intensity vectors an image can be formed, with the width n of the sensor array and the height of the range r (see fig. 2.5 (b)).



(a) Two-dimensional top view of SONAR image aquisition, according to [19]

(b) Resulting two-dimensional image

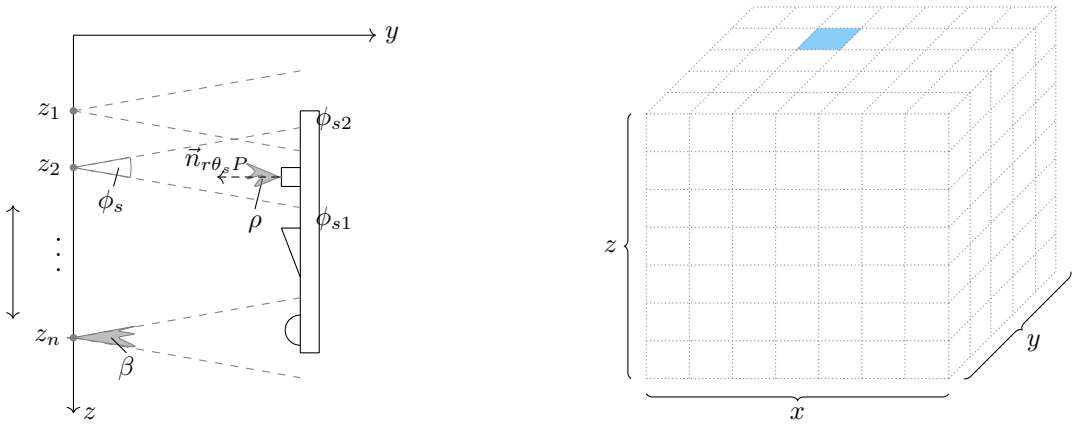
Figure 2.5.: Two-dimensional SONAR image formation

$$I_s(r, \theta) = \int_{\phi_{s2}}^{\phi_{s1}} \beta(\phi_s) V_s(r, \theta_s, \phi_s) \frac{\vec{v} \cdot \vec{n}_{r\theta_s\phi_s}}{\| \vec{v} \| \| \vec{n}_{r\theta_s\phi_s} \|} \tag{2.3}$$

In fig. 2.6 (a) the SONAR image formation regarding the depth is presented. The SONAR will record acoustic intensity values from its position on the z-axis. In the range from ϕ_{s1} to ϕ_{s2} the volume intensity values $I_s(r, \theta_s, \phi_s, z)$ are detected and then saved as already shown in fig. 2.5 (a). With $\Delta z_r \approx r \tan(\frac{\phi_{s2} - \phi_{s1}}{2})$ eq. 2.3, V_s as a binary function (1 for surface and 0 for none) and the assumption that the reflection ρ at point P has a surface normal $\vec{n}_{r\theta P}$ can be rewritten as a function of depth as shown in eq. 2.4 [19].

$$I_s(z, r, \theta) = \int_{z+\Delta z_r}^{z-\Delta z_r} \beta(\phi_s) V_s(r, \theta_s, P) \rho(z, P, \vec{n}_{r\theta_s P}) \tag{2.4}$$

These vectors can then be arranged in a three-dimensional vector space shown in fig. 2.6 (b) with n of the sampled depth measurements.



(a) Data acquisition concerning the depth z, according to [19]

(b) Resulting three-dimensional image

Figure 2.6.: Three-dimensional SONAR image formation

Overlapping beams, such as z_1 and z_2, shown in fig. 2.6 (a), result in a doubled detection of the area in between, which leads to a blurred image along the z-axis in the received three-dimensional representation. Further quality decreasing influences on the backscattered signal can be caused by surrounding noise and reverberation, as presented in the following section 2.2.

2.2. Noise and Reverberation

The cause of underwater noise can be differentiated between three main categories: the "self-noise", anomalous local noise and ambient noise. The "self-noise" is generated by the measurement setup itself, for example sensor platform motion, cable strumming, boat noise etcetera. It is not a constant noise and varies in amplitude and phase, which makes it difficult to reduce in later signal processing. Same applies to the anomalous local noise, which is produced by animals and ships close to the measurement setup. Ambient noise describes background noise of sources far from the detector. Natural phenomena as wave movement, rain and oceanic turbulences, passing fish swarms and mammal pods, but also naval construction and maritime shipping make up a large amount of this noise. Tides, which play a vital role in data acquisition in the North Sea, as a part of the ambient noise, can be disregarded due to their relative small impact of a noise level below 1 Hz [23, p. 125].
In contrast to noise, which is generated by external sources, the reverberation describes the inadvertent reception of echoes from the transmitted signal by the SONAR itself. These are caused by inhomogeneities in the water column, including biological activity, such as marine organisms, but also natural sources, such as locked air bubbles under the sea surface. Therefore it can be roughly differentiated between either the volume and/or surface and bottom reverberation, depending on the prevailing conditions on site.
The former can be divided in Rayleigh scattering by particles much smaller than the wavelength of the transmitted SONAR signal and the geometric reflection by objects larger than the wavelength [23, p. 113]. The Rayleigh scattering describes the scattering of energy in many different directions, which depends on the squared frequency, whereas the shape of the dissolved sediment is irrelevant [23, p.113]. Especially in estuaries and adjacent harbours near the Wadden Sea a high amount of dissolved fine sands (approximately between 125 μm and 250 μm) is to be expected, due to the sediment transport from rivers and through the incoming tides [25, p. 236], which could lead to disturbances. The geometric reflection on the other hand is dependant of the surface reflective properties of the object and independent of the frequency. Usually volume reverberation can be observed in deep ocean environment, resulting from a relative small amount of biological organisms, such as ($\approx 0.05\,\mathrm{m}^{-3}$), but stratification has been observed in shallow water, too [23, p. 121]. The kind of organisms ranges, according to Caruthers, from larger ones, such as squids, to much small crustaceans like krill. In port areas, no high organism density or

stratification is expected, due to the high ship traffic and the limited access of organisms to the port basin through sluices. In the area of offshore energy plants, however, a higher level of biological activity can be expected, since the foundations and the relatively untravelled area around the plants provide a favourable habitat for many marine animals.
The surface reverberation results from the backscattering of the sea surface and the sea floor. The sea surface backscattering strength varies with angle of incidence, frequency and roughness of the surface, which is related to the wind speed. The bottom reverberation on the other hand depends on the structure and material of the sea floor. [23, p. 121f.]

The reduction of noise is an important task in the acoustical image generation and processing, but also in the application of acoustic positioning, as noise sources are unavoidable in inhabited, trafficked and economically utilised measurement areas. In general noise is represented by means of the Gauss distribution and is visible as "white noise" with a flat spectrum, meaning an equal intensity at every frequency, in the respective SONAR images [23, p. 132]. Noise threshold application and a filtering technique, will be explained and applied in section 4.4, specified for use in a closed port area with minimal marine traffic and low biological activity.

2.3. Georeferencing by Short Baseline Systems

Georeferencing in the context of underwater data acquisition is a fundamental requirement for taking into account the exact location of the data and the environmental influences prevailing there. Furthermore the navigation of autonomous and remote-controlled underwater vehicles such as AUV and ROV is greatly facilitated by positioning systems. Also financial and time aspects, which are important for measurements on ships in the high seas, can be reduced by simplified navigation and measurement target execution in the area under investigation. But also in coastal areas with particle-rich water, where navigation by sight is not possible, an underwater positioning system is a good extension of the measurement setup.

The distance between acoustic baselines, which means the submerged sensing elements, is generally used to define an acoustic positioning system. As already mentioned in section 2, it can be differentiated between LBL, SBL and USBL. In contrast to the LBL and USBL, the SBL is used especially for its high accuracy and ease of use, both at sea and in coastal waters. It consists of a main computer unit with integrated *Inertial Measurement Unit* (IMU) and GPS-antenna, usually three or more hydro-acoustic receivers and one locator. The system measures phase differences on arriving pings between the surface mounted receivers. Respectively the bearing, defined by azimuth (orientation) and elevation, is derived from the detection of the relative "time of arrival" as an acoustic signal, transmitted by the locator, passes each of the receivers [45, p. 6].
Its accuracy is specified as a percentage of slant range, which is the direct distance between two points on different altitude, in this case the distance from the receiver to the locator. According to Vickery [45, p. 12] it can be stated that the greater the depth, the greater the slant range and the less repeatable is the position of the underwater vehicle. For SBL-systems the frequency band ranges from low frequency to very high frequency, compare exemplary annex B table B.6. According to [45, p. 8], the selection of the frequency band is in direct correlation to the requisite positioning accuracy for measurement target, as well as the consideration of other used acoustic products, background and self-noise, which might interfere the locator signals. The accuracy is also dependant on additional factors such as installation and calibration, as well as environmental conditions.

Influences on quality can be categorized along two dimension, presented below.

Systematic errors:

- Inaccurate or not precise speed of sound profile
- Inaccurate position determination by the GPS-antenna or false measurement/submission of the distance between the receiver and the computer unit
- Insufficient submersion of the receivers
- Incorrect mounting of the locator on the vehicle
- No or insufficient movement compensation of the measurement platform

Cause of noise:

- Self noise movement of the vessel/measurement platform around the x- and y- axis, resulting in an angular deviation (Roll- and Pitch angle)
- Misleading acoustic noise from bubbles created by waves
- Signal reflection, due to a restricted measurement area (harbour, tank, pool)
- Ambient/background noise from marine mammals, ships
- Unstable position through other acoustic systems, such as SONAR

An enhancement of unstable position data can be achieved by reducing the impact of outliers in the measured data by usage of the geometric median, described in section 2.3.2).

Advantages and disadvantages of an SBL acoustic positioning system, that need to be considered in regard of the measurement target, are compared below (see tab. 2.1).

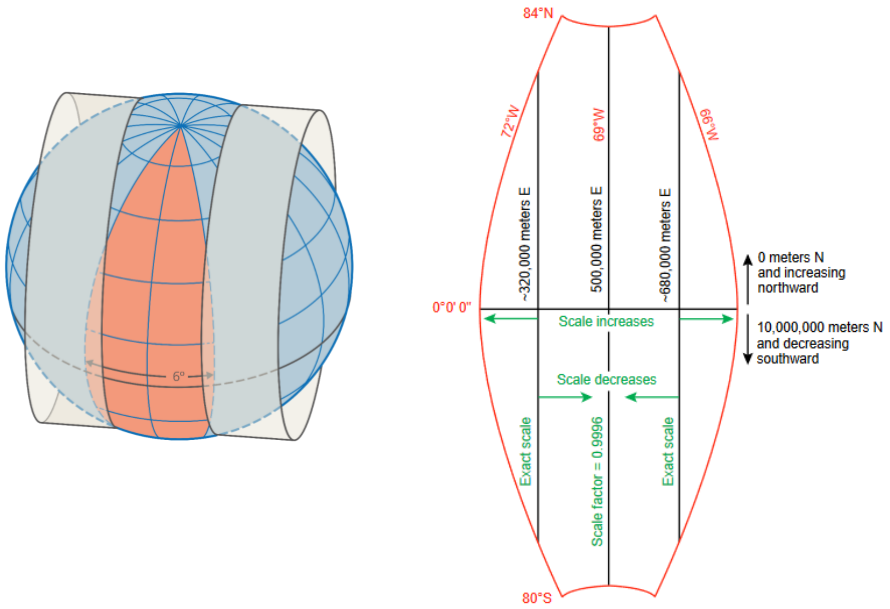
Table 2.1.: Comparison of the advantages and disadvantages of a SBL underwater positioning system

Advantages	Disadvantages
• Low system complexity and therefore easy to use • Small transducer an locator size, often equipped with self sufficient power supply and additional pressure sensor • Good range accuracy with time of flight (determination of the distance from receiver by means of sound velocity and time) • Ship based system (no sea floor transponder)	• Large baseline length for accuracy in deep water • Absolute position accuracy ends on additional sensors (IMU and GPS-antenna) and integration on sensor platform (e.g. ROV or AUV) • Highly accurate dry and offshore calibration required

As already noted, SBL positioning systems determine the relative position of the underwater vehicle to the origin of the main computer. For a global specification, a GPS-antenna, in order to determine the global position of the main computer, must be added to the measurement setup and added as an offset to the local position. Knowledge of the global coordinate system, in which the local position is then transformed is a key element for data accuracy.

2.3.1. Coordinate System

A differentiation between a global and local coordinate system must be considered in order to determine the correct position of a *Point Of Interest* (POI) or acquired data set.
Regarding the described positioning with a SBL system in section 2.3, the local coordinate systems spans according to the position and range of the receivers, with the origin of the computational units position on the measurement platform. Therefore the position of the underwater vehicle and the acquired data is determined regarding the latter.



(a) Universal Transverse Mercator projection with superimposition of the cylinder projection [17]

(b) UTM zone 19, from 84 °N to 80 °S and 66 °W to 72 °W, according to [27], with exaggerated shape and applied scaling factor of 0.9996 to the central median

Figure 2.7.: Schematic Universal Transverse Mercator projection

When transforming the local into a global position the distance from the computer unit to the GPS-antenna, as well as the distance from the GPS reference point is needed. The transformation also includes the motion data of the computer unit (e.g. if mounted on ship or other moving platform).
The projection of two-dimensional data points on a sphere (in this case the earth) requires the application of a further global coordinate system. The UTM, fig. 2.7, in contrast to the *Word Geodetic System 1984* (WGS 84) of latitudes and longitudes, selects the earth surface between 80° South and 84° North and differentiates it in bands of 6° width from west to east. The bands are adjusted and levelled by means of a transverse Mercator projection, which implies the projection onto a horizontal cylindrical surface, see fig. 2.7 (a). This coordinate system allows the application of a Cartesian coordinate system, with the x-axis aligned true-east and the y-axis aligned true-north regarding the zones central meridian. When dealing with eastings (right-handed east coordinates) an off-set value must be defined in order to avoid negative values when entering the left handed scope. This method is called false-easting and sets the x-value of the central median to 500 000 m East. The same principle applies to the vertical values between 84° North and 80° South. Those that are below the equator would be negative, to avoid this the equator is set to 10 000 000 m North on the southern hemisphere, decreasing southward as shown in fig. 2.7 (b) to 1 100 000 m at 80° South. For values on the northern hemisphere the equator is set to 0 m North, increasing northward up to a maximum value of 9 300 000 m at 84° North. Furthermore a scale factor of 0.9996 is applied to the central median, see fig 2.7 (b) in order to project the earth on a plane. When

measuring distances form the map this scaling factor needs to be multiplied in order to obtain the true distance. [27]
Based on the UTM-grid system in the *Military Grid Reference System* (Military Grid Reference System), each zone is separated into 20 latitude bands in alphabetical order starting from "C" at 80° South to "X" ending at 84° North (omitting the letters "I" and "O"). A combination of the zone and letter marks the defined grid zone, for instance the German Bight is situated in the grid zone 32U. In the coordinate transformation presented in section 4.5.4 this grid system is applied to the global coordinates in latitudes and longitudes.

2.3.2. Geometric Median

The conversion of local positions of an acoustic positioning system to global positions can contain errors of different origin. Not only sensor deviations or disturbances of the transmitter/receiver behaviour, but also measuring errors during manual distance measurements between receiver and computational unit and between computational unit and the GPS antenna. Furthermore outliers of the position can occur, due to a reflective measurement environment, ambient noise or anomalous local noise. When using a positioning system without additional support by a *Doppler Velocity Log* (DVL) or INS, the positions must be examined more closely and outliers must be removed from the data set to guarantee the correct positioning of the measured data.
In order to reduce the outlier impact, especially when performing static measurements regarding the x- and y-axis, the geometric median, also known as multivariate L_1-median, can be applied and compared to all positions in the data set.
Sources of different algorithms for the computation of the geometric median haven been proposed by literature, which vary according to precision and algorithm runtime. The most commonly used are the Weiszfeld, Vardi and Zhang or the Non-linear minimization algorithm [13], which calculate the minimal distance between all points iteratively using derivatives and non-negative weights. Due to their complexity the geometric median according to Nelder and Mead [13], also known as downhill simplex method, poses a fast and easy alternative without derivatives.

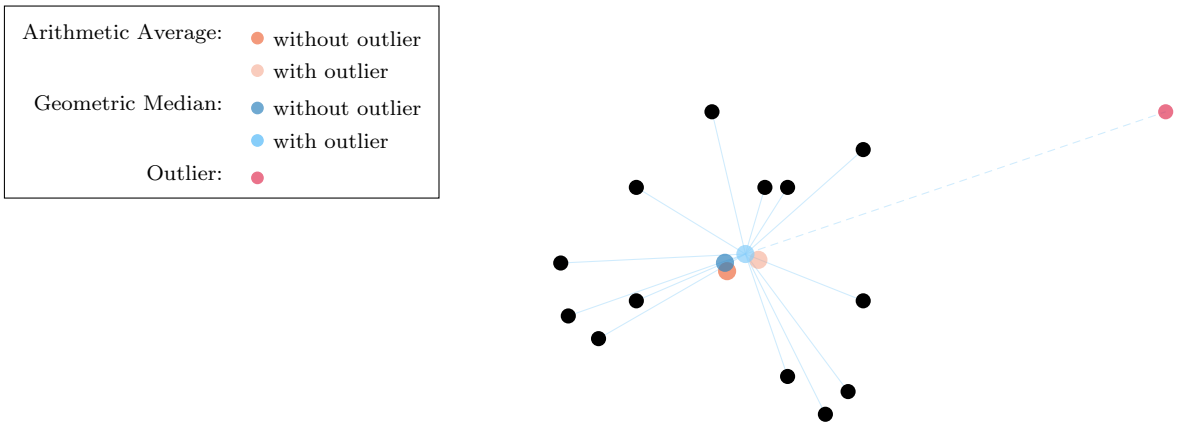


Figure 2.8.: Comparison of Arithmetic average and geometric median in two datasets

At this point, the procedure of the algorithm shall only be briefly described, due to the rather subordinate position in this thesis. Further information can be found in J. H. Mathews and K. D. Fink [30]. The Nelder and Mead algorithm, like the algorithms mentioned above, searches for the local minimum of a function with several variables. In the case of improving the position, two (x, y), respectively three (x, y, z), variables are used. The simplex in $\mathbb{R}^n$ denotes a convex hull with $n + 1$ vertices. In the case of two variables this would correspond to a triangle. The method now consists of a pattern search, which compares the vertices of the triangle to one another. The largest distance $f(x, y)$ between two corner

points is then discarded and replaced by a new vertex, forming a new triangle. This process is repeated until the smallest point is found. Figure 2.8 shows a two-dimensional data plot of coordinates, on which the arithmetic average and the geometric median, according to Nelder and Mead, are applied. Once to the data set excluding and then including the outlier (red) on the far right hand side of the figure. It can be observed that the arithmetic average with outlier moves slightly more towards the outlier, than the respective geometric median, and therefore affects the mean position of the whole data set. The deviation of the geometric median is further calculated by the *Median Absolute Deviation* (MAD), which is respectively defined by the sum of the values minus the median and then divided by the amount of values.

Following the theoretical explanation of relevant factors regarding the ROV based acquisition of SONAR images and underwater vehicle positioning in the presence of noise, now the mean for data acquisition is addressed by a presentation of the the hardware being used.

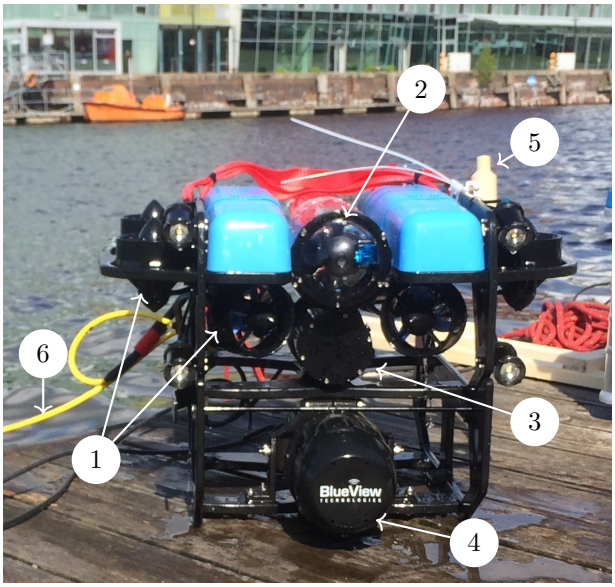
3. Hardware

For the data acquisition of georeferenced SONAR images, three system have been used affiliated by means of carriage, data transfer and electronic merge. The sensor carrier BlueROV2 by Blue Robotics [6] itself is used to navigate the sensor systems to a POI by using a computational unit and remote controller, henceforth designated as *Topside Computer* (TC). The UGPS by the company Water Linked AS [49] is an SBL acoustic positioning system for aided navigation. It facilitates the navigation of the BlueROV2 to the POI and calculates its position underwater, which is necessary for the three-dimensional allocation of the SONAR images. The last system consists of the SONAR itself, where a multi-beam SONAR model P900-130 of the company BlueView Technologies [11] is used. In the following sections each system will be introduced individually. The overall costs, all system specification as well as corresponding references can be found in annex B table B.1 to table B.6.

3.1. Remotely Operated Vehicle BlueROV2

The BlueROV2 [6] by Blue Robotics is an inexpensive, for industry standards (s. table B.1), but high quality ROV, which is characterized by its modular build frame setup, open source electronics and software and high customizability. The configuration used in the measurement trial, presented in section 5.1, uses the "Heavy Configuration Retrofit Kit" with additional "Payload Skid". This instrumentation uses, instead of the regular BlueROV2, eight thrusters in order to allow controlled motion in all six degree-of-freedom. It is therefore possible to manoeuvre more stabilized with varying roll- and pitch-angles and additional payload in the predefined "Stabilization Mode".

Figure 3.1 shows the ROV BlueROV2 in the mentioned configuration and corresponding amount of thrusters thruster, at fig. 3.1①, for the sake of clarity, only marked on the front left hand side. The electronic enclosure fig. 3.1② contains an Pixhawk 1 flight controller from the PX4 open-hardware project, which executes the ArduSub control software of the BlueROV2. It has two acceleration sensors, two gyroscopes, a magnetometer and a barometer, as listed in table B.3. By comparing and validating the gathered data of the first three sensors the roll- pitch and yaw-angle are determined accordingly. For further sensor information, please consult section B table B.4. Below in the battery enclosure fig. 3.1③, an 14.8 V Lithium-Ion battery supplies not only the ROV, but also the SONAR fig. 3.1④. The locator U1, ⑤ in fig. 3.1, which is part of the UGPS, is self energy sufficient and includes an own depth sensor. Therefore the locator U1 is only attached to the sensor carrier, without the necessity of power supply or data transfer integration. Through the tether cable fig. 3.1⑥, using two of the four twisted two-pair cables, the transfer of the ROVs flight parameter and SONAR data to the TC takes place.



1 Thruster
2 Electronic enclosure
3 Battery enclosure
4 BlueView P900-130
5 U1 Locator
6 Tether cable

Figure 3.1.: Customized BlueROV2 using the "Heavy Configuration Retrofit Kit" and "Payload Skid" with additional sensors, locator U1 and BlueView P900-130

In the section B table B.2 an overview of the integrated sensors, corresponding company, version and reference can be found. The data acquisition of the movement parameters, roll- (ϕ) pitch- (θ) and yaw-angle (ψ), as well as the depth data of the barometer is discussed in section 4.3.1.

3.2. Acoustic Positioning System - Underwater GPS

The *UGPS* by Water Linked AS is a SBL, as presented in 2.3. It is developed for small AUVs, ROVs or divers and shows great performance results, especially in shallow waters and reflective environments according to the manufacturer [49].

The measurement setup is installed on a floating pontoon, shown in ① fig. 3.2. The positioning system consists of four receivers (② in fig. 3.2), an external GPS-antenna (③ in fig. 3.2), a main computational unit located in the TC, hereinafter called Master D1, and an locator (④ fig. 3.2). The four receivers are submerged into water, to a minimum of 1.5 m, and connected to the TC (⑤ in fig. 3.2). Inside the TC they are connected to the Master D1, which is implemented in the TC as well, according to fig. 3.3 (a). The locator④ is mounted on the ROV ⑥, both in fig. 3.2, so that the acoustic signal can spread in all directions.

In order to calculate the correct local and global position, the distances d_1 - d_4 in x- and y-direction, as well as the depths z_1 - z_4 from the receiver to the water surface need to be adjusted using the web-based *Graphical User Interface* (GUI) provided by Water Linked AS. In this GUI further adjustments regarding the measurement area, used GPS-antenna, IMU and locator and its frequency channel can be made. The *Internet Protocol* (IP) address of the GUI can be used to access the position data determined by the system. This process of data acquisition will be explained in section 4.3.2. Corresponding listings are presented in the annex C.

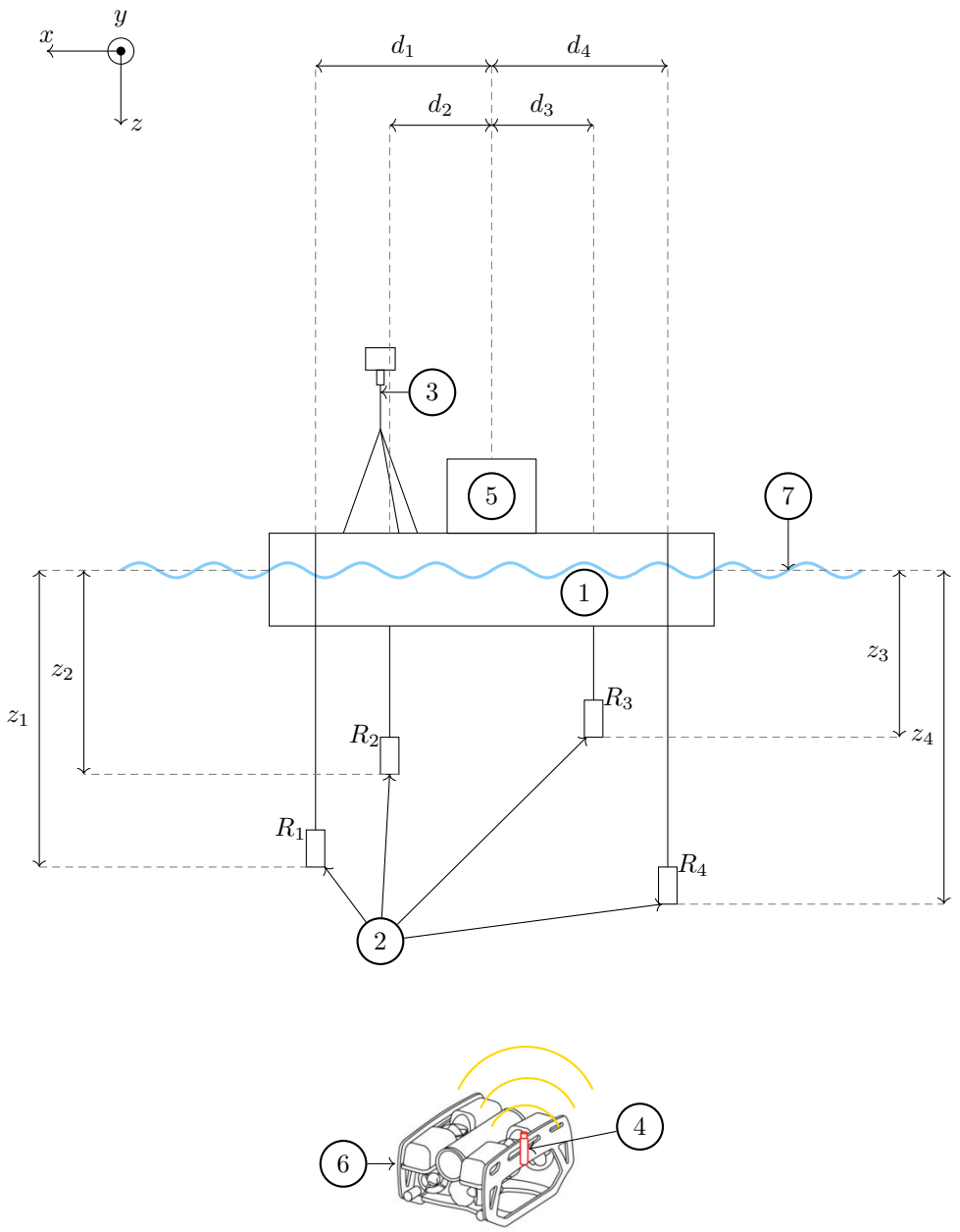
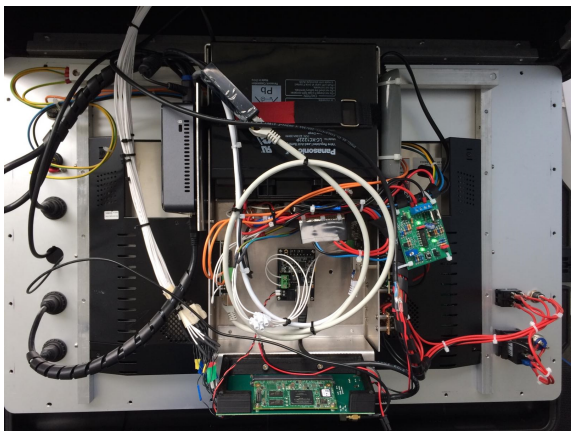


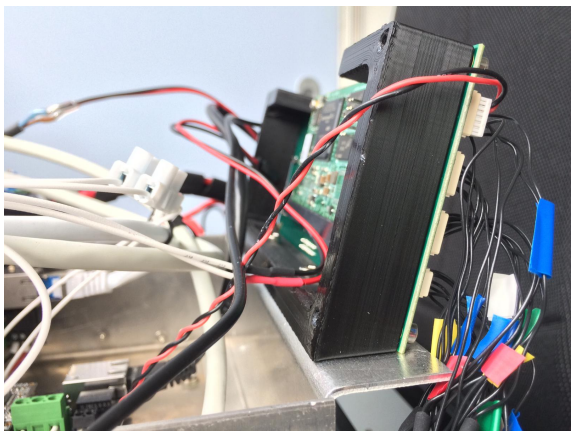
Figure 3.2.: Schematic of Water Linked underwater acoustic positioning system setup (with adapted ROV schematic from [49] and schematically enlarged GPS-antenna); Legend as explained below:
① Pontoon ② Receiver ③ GPS-antenna ④ Locator U1 ⑤ Topside Computer ⑥ ROV
⑦ Water surface

3.2.1. Topside Computer Integration

The TC is a splash-proof case containing all necessary communication interface to receive data from the ROV and SONAR, as well as from the UGPS Master D1 by using Fathom-X Tether Interface Boards [9] (used for the long-distance Ethernet connection over a twisted single pair cable) by Blue Robotics. The Master D1 is installed by a self-designed board holder (s. fig. 3.3 (b)) in such way, that the 15° angle of inclination is compensated when the case is opened, which is necessary to correctly calculate the position of the ROV in the water. Figure 3.3 (a) shows the inside of the TC.



(a) Interior of TC, including a computer unit (top left hand side), battery (black, middel top side), and serial to Ethernet converter (Fathom-X Tether Interface Board in metallic box center) and board holder with Master D1 (bottom)



(b) Slanted circuit board holder for Master D1

Figure 3.3.: Slanted integration of Water Linked UGPS Master D1 electronics into TC

3.2.2. GPS Antenna

Due to higher accuracy the GPS-antenna used in the original setup was replaced by a PmodGPS™antenna by DIGILENT®[15] in combination with an Arduino Nano 3.0 [1], which were provided by the DLR institute MI. The antenna module, shown in ③ in fig. 3.2 is symbolically represented by a large antenna for the sake of clarity. The GPS-antenna in the measurement setup (s. sec. 5.1.1) is located exactly above the Master D1.

3.2.3. Receiver

Four digital low-noise high-sensitivity hydro acoustic receiver D1 (WL-21005-B-010) [50] with the detection range of 100 m are lowered into the water [49]. A minimal distance of 1.5 m below the water surface (⑦ fig. 3.2) should be guaranteed to reduce disturbances through breaking waves at the water surface. Corresponding technical specifications can be found in the annex B.

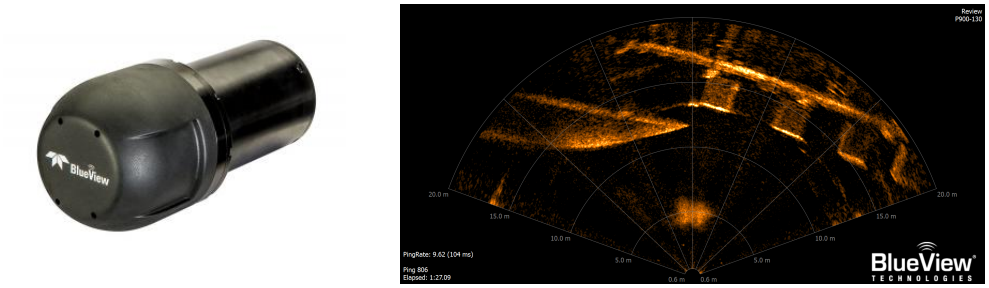
3.2.4. Locator

The U1 locator [48], shown in ② fig. 3.2, is a digital hydro acoustic locator by the company Water Linked AS with a self supplied power source which lasts up to 6 h operation time. The latter is especially beneficial when using the ROV BlueROV2, due to limited cable penetrators to the onboard power supply. There are seven different channels that can be chosen from (see annex B). The seventh channel with the frequency range from 218 Hz to 250 Hz is chosen for the measurement trial presented in section 5, due to reduced masking effects of the simultaneously used multi-beam SONAR frequency of 900 kHz.

3.3. Forward-looking Multi-beam SONAR BlueView P900-130

The BlueView P900-130 by BlueVIEW Technologies (s. fig. 3.4 (a)) appertains to the category of two-dimensional forward-looking multi-beam SONARs, this means establishing the field of view in the direction of travel, as well as emitting multiple acoustic signals from a hydro-acoustic array within the SONAR. The numbers in the designation can be traced back to the system specifications, so the SONAR has a transmission frequency of 900 kHz and a field-of-view of 130°.
A forward-looking SONAR bears the advantage of a horizontal aligned installation, which is necessary in order to inspect vertical aligned structures, and a corresponding large field of view beneficial for covering large areas. Regarding the two-dimensionality of the image generation, please refer to chapter 2.1 figure 2.5.

For every emitted pulse (ping) an image of the backscattered values from objects in the water columns is generated. It must be clarified that not an infinitesimal strip of the water column is detected by the SONAR, but with increasing horizontal distance also a vertically larger area of the water column by the beam angle of 20° (see chapter 2.1 figure 2.6 (a)). In figure 3.4 (b) a typical backscatter image with the range of 20 m, produced by this type of SONAR can be seen. On the left hand side of the image, the outer shape of a ship hull, as well as its keel, can be identified. From the middle to the upper right side of the picture the acoustical reflection of the sheet pile wall of the harbour basin and (in form of the rectangles) buoyancy bodies of a floating pontoon can be detected. The filling of the rectangle buoyancy bodies is due to the large vertical aperture angle of the SONAR, which results in a detection of the buoyancy body not only from the front, but also either from above or below (s. sec. 4.3.3 fig. 2.6 (a)). In the front middle part image a strong cloud shaped noise is present, due to the movement of the sensor carrier itself. The colour coding of the image is only due to better visibility of the objects. Further system specifications can be found in annex B.



(a) BlueView P900-130 forward-looking Multi-beam SONAR [40]

(b) Backscattered image from ship hull and sheet pile wall, Fischereihafen 1, Bremerhaven 2020

Figure 3.4.: Multi-beam SONAR BlueView P900-130 and backscattered data image

The SONAR integration into the ROV itself was completed in the course of an internship at the DLR MI from October 2019 to February 2020 and will only be explained here for the purpose of completeness and for the later comprehension of data acquisition.

The BlueView P900-130 [10] is mounted on the BlueROV2 (s. fig. 3.5 ① and fig. 3.1), and connected to the onboard power supply of 14.8 V (s. fig. 3.5 ②). The data transport is realized via Fathom-X Tether Interface Boards [9], produced by Blue Robotics, located in the electronic enclosure fig. 3.5 ③. These lead the signal by one pair of twisted cable in the tether cable (s. fig. 3.5 ④) to a second Fathom-X Tether Interface Board to the TC fig. 3.5 ⑤. There the signal is fed into the computer by an Ethernet to *Universal Serial Bus* (USB) adapter and can be accessed by the IP-address *192.168.1.45*. The data acquisition using the customer friendly software ProViewer 3™[44] is comprehensively described in section 4.3.3. Data collection using the ProViewer 3™ has the advantage that the current backscattered data is

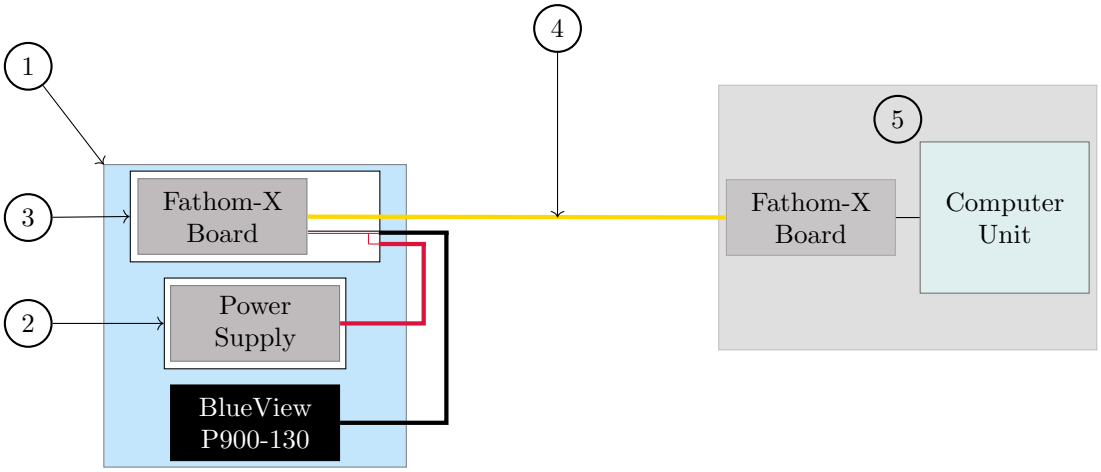


Figure 3.5.: Implementation schematic of P900-130 into system electronics of BlueROV2

displayed on screen of the TC so that it can be used as a navigation assistance additionally to navigation on sight and the acoustic positioning system, presented in section 3.2. Range, intensity and threshold can also be adjusted in real-time to simplify the target object recognition for the user. The recorded data can only be saved in the BlueVIEW proprietary *.son*-format. Additionally snapshots and movies can be generated by calling the previously recorded file in the ProViewer 3™.

4. Software

This section describes the complete process of data acquisition, processing and presentation of the data generated by the three systems BlueROV2, UGPS and P900-130 SONAR. Beginning with a graphical overview of the data processing procedure, the relationships between the programs is presented. In the following, the programming languages and visualization programs used are briefly described followed by the main part of the section, which is divided into the Data Acquisition from Hardware (s. sec. 4.3), Image Processing (s. sec. 4.4) and Sensor Data Fusion and Three-Dimensional Representation (s. sec. 4.5). The first presents the data acquisition from the three systems and gives an overview of the expected data formats by means of examples. Section 4.4 deals in particular with the image enhancement of SONAR images. The application and effects of the filter algorithms used, median 7x7 filter , are presented and the thresholding from a normally distributed noise pixel Z-test is discussed. The last section brings together all the data acquired up to that point and shows the program flow, from timestamp allocation and coordinate transformation to the graphic display of the acquired point cloud.

4.1. Program Overview and Visualization

For the three-dimensional visualization of the SONAR images, data from different systems must be collected, pre-processed and integrated into a main program, which merges and further processes all data. In contrast to software programs available on the market, this does not work automatically yet, but must be carried out manually. A distinction is made between programs that run directly on the topside computer, such as direct data collection from the systems via *User Datagram Protocol* (UDP)/IP bridges (in the case of the motion parameters additionally via a proxy, the so-called MAVProxy by *Micro Air Vehicle Communication Protocoll* (MAVLink) [2]), and the main program. The latter, the BlueView *Software Development Kit* (SDK) SONAR file format conversion program, as well as the threshold algorithm and the program to determine the exact pixel size of the recorded SONAR image range r, are executed on an external computer.
Figure 4.1 shows a schematic of the whole program components and their interconnection. The fading light blue boxes, beginning with the hardware systems on the far left hand side, represent the programs running on the TC. The green boxes represent files that need to be transferred to the external computer, using a storage medium, and processed using programs in the blue boxes. The yellow box *Visualization* represents the result of the program operation, which also runs on the external computer.

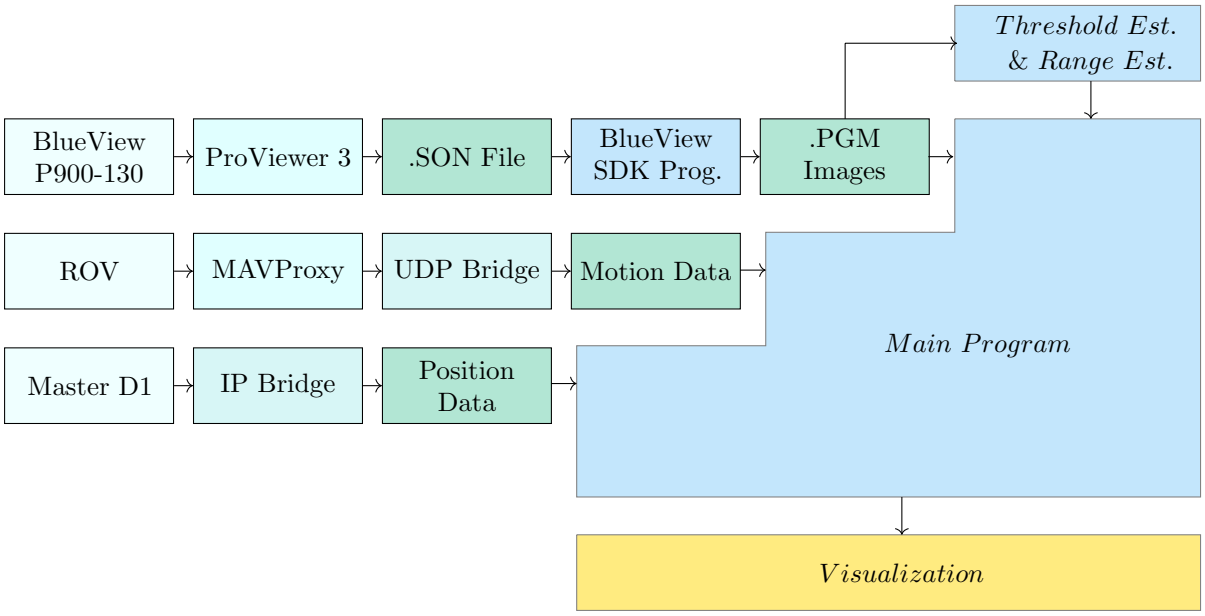


Figure 4.1.: Schematic view of program data flow

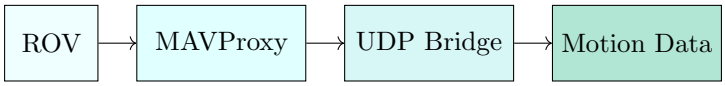
4.2. Software Tools

With regard to the software used, a distinction can be made between internally developed and external programs. The self-developed programs were mainly written in the programming language Python, except for the SONAR processing from the proprietary format of the company BlueView to a more common format, which was realized with the help of C++ and the software development kit of BlueView. *Application Programming Interface* (API) or SDK are designed by the manufacturer to facilitate the implementation of additional hardware or writing new applications for the specific system. In case of the BlueROV2 using the MAVLink protocol and for the BlueView P900-130 using various predefined functions facilitates the data acquisition. As external open source software programs Meshlab [26] and NaviModel Free Viewer [16] are used, which both allow a display of point clouds and their analysis, in contrast to the module open3D [33] used in Python for the generation of the point cloud, which only allows a display but no analysis. Further an external program for the integration of external GPS-data into the web-based Water Linked AS GUI is used.

4.3. Data Acquisition from Hardware

In this section, the steps and programs for data collection from the systems mentioned in the Hardware section 3 are presented and explained. The communication between the system and the TC and possible intermediate operations are discussed. In addition, the scripts used are assigned and partly presented in detail. The complete source code can be reviewed in annex C.

4.3.1. Motion Parameter of Sensor Carrier



To connect the BlueROV2 with the TC, the latter requires a static IP address of 192.168.2.1. The ROV itself sends its data by the IP address 192.168.2.2 of the Raspberry Pi 3 B+ [8]. It is thus located in the same subnet mask as the TC. By default, the data stream arrives at port 14450 and is retrieved when using the control software *QGroundControl* (QGC)[38] (see figure 4.2). The image data of the optical camera is sent to the TC via *GStreamer*, which acts as a distributor of the video data in a local network, which will not be covered in this thesis.

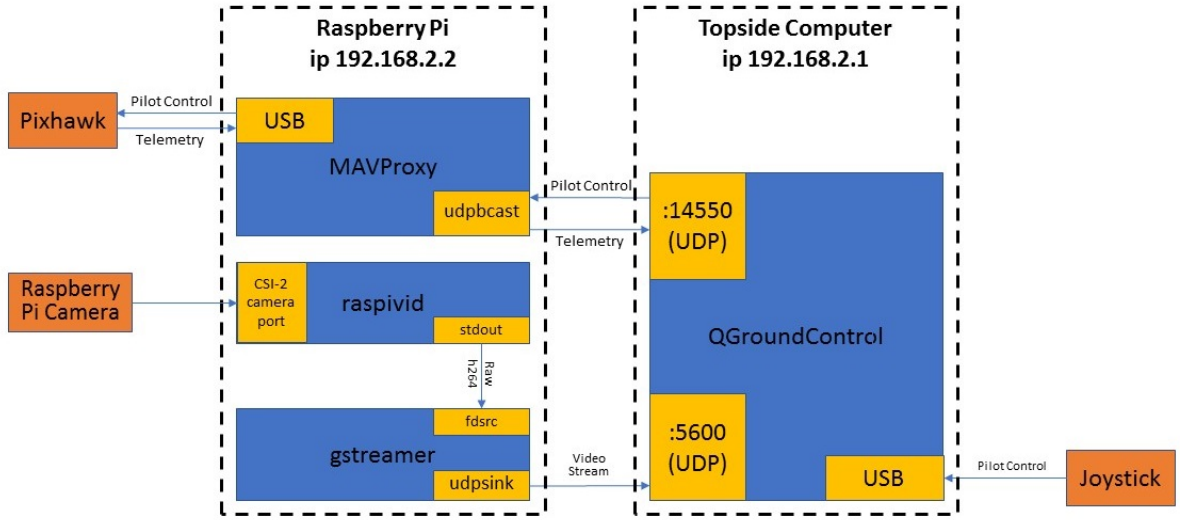


Figure 4.2.: Network communication between BlueROV2 and TC

To acquire the sensor data of the BlueROV2, a transparent proxy must be inserted between the communication between the BlueROV2, more precisely the Pixhawk 1, and the TC defined in figure 4.2. A proxy is a network communication interface between two computers. The address of the transparent proxy remains invisible for both sides, so that not the proxy but the target computer itself is addressed. The proxy can be used to define different interfaces (e.g. UDP or *Transmission Control Protocol* (TCP)) with input and output IPs. The proxy used for the BlueROV2 is MAVProxy. It uses the MAVLink communication protocol, which is used between *Autonomous Underwater Vehicles* (UAV) or *Unmanned Underwater Vehicle* (UUV) and QGC, as well as internally as a communication basis between microcontrollers, such as the Raspberry Pi, and the ArduSub software on the Pixhawk.

The installation and documentation of MAVProxy is available on the official ArduSub GitHub site [2]. In the following the necessary steps are described to use the module in combination with the acquisition of system parameters of the BlueROV2.

In the source code 4.1 the *setup.py* is executed first. This must be repeated each time the module is changed, since only then all modules in the *modules* folder can be loaded by MAVProxy and thus be

executed. In normal operation the execution is sufficient from the definition of inputs and outputs. In the last line a module is loaded which sends certain system parameters from the data stream to a defined UDP interface. In this case it is module *alldata.py* which is described in the following.

```
python setup.py build install
mavproxy.py --master=udpin:0.0.0.0:14550 --out=udpout:0.0.0.0:14551
module load alldata
```

Listing 4.1: MAVProxy setup and loading modules

Server-side Communication

The module *alldata.py*, as already used in lst. 4.1, is an adaptation of the modules *DepthOutput.py* and *example.py* published on the official GitHub site of Blue Robotics under MAVProxy [7]. The former taps depth, temperature data, and the orientation of the ROV. The second serves as a general guideline for the integration of new modules and can be used as a starting point. A list of the control and system parameters can be found in the MAVLink documentation [31].

The module *alldata.py* accesses the following data:

- Depth $[m]$
- Water temperature [°C]
- Orientation [°]
- Roll-, Pitch- and Yaw-angle [°]

The complete source code can be found in the appendix C lst. C.1. For the sake of clarity, the module is only viewed using two source code excerpts.

In source code 4.2, the IP address and the interface, to which the data is to be sent, are specified first. In the following, the socket function is used to send data. A socket is a kind of communication point that is used to exchange data between two programs. AF_Inet describes the type of socket family that the socket can communicate with, in this case Internet Protocol v4 addresses with the host 127.0.0.1 and the port 25102. SOCK_DGRAM, the second argument of the socket function, refers to the use of an UDP interface. In setsockopt, SOL_SOCKET is used as the so-called level argument to change the socket properties. By setting the value 1, which stands for the Boolean *TRUE*, the socket address is reused even if the status of the socket is in TIME_WAIT state and the *timeout* has not expired. If the program is executed too frequently and with too little delay, it may cause the system state to remain in TIME_WAIT. The procedure described above eliminates the error that the address is already in use and therefore cannot be used. Overall, the Socket function acts as a server for the Client, which is described in lst. 4.5. [37]

```
self.ip="127.0.0.1"
self.portnum = 25102
self.port = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
self.port.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
mavutil.set_close_on_exec(self.port.fileno())
print "Outputting data on UDP://%s:%s" % (self.ip, self.portnum)
```

Listing 4.2: Creating the communication interface using socket

In the lst. 4.4 the MAVLink packets are tapped from the data stream. The data is stored in *JavaScript Object Notation* (JSON) format. A simple access to a value is done by calling the declaration, also called key-value pair. In a JSON object, multiple key-value pairs are separated by a comma. The JSON format allows Boolean values, numbers, strings, null values, and arrays (indicated by square brackets), which allow a kind of sub-grouping in the JSON object. In the fictitious example listing 4.3, the values stored in it can be retrieved by querying the key, for example model_no. [36]

```
{
"ROV_type": "BlueROV2",
"model_no": 4356,
"config": [
  {"mode": "heavy duty"
  "thruster": 8},
  {"mode": "normal"
  "thruster": 6}
  ]
}
```

Listing 4.3: JSON Object

The depth and temperature data are displayed by the bar30 and sp2, so the first if-loop is run to store the values in 'temp' and 'depth'. sp3 serves as input if a separate temperature sensor is connected. As described in chapter, only the Bar30 sensor is used for this purpose. In the following one will be asked whether values are available in the keys VFR_HUD, RAW_IMU and ATTITUDE. The corresponding values for the associated subkeys, such as roll-, pitch- and yaw-angle, are stored in variables in self.data. Later in the program, self.data is sent to the defined UDP interface.

```
'Handle mavlink packets, get data'
def mavlink_packet(self, m):
  '''handle mavlink packets'''

  if m.get_type() == 'SCALED_PRESSURE2':
    if self.temp_source == 'sp2':
      self.data['temp'] = m.temperature / 100.0 # m.temperature is centi-C
    if self.depth_source == 'bar30':
      self.data['depth'] = self.get_depth(m.press_diff * 100) # m.press_diff is hPa
  elif m.get_type() == 'SCALED_PRESSURE3':
    if self.temp_source == 'sp3':
      self.data['temp'] = m.temperature / 100.0 # m.temperature is centi-C
  elif m.get_type() == 'GLOBAL_POSITION_INT':
    if self.depth_source == 'filtered':
      self.data['depth'] = -m.relative_alt / 1000.0 # m.relative alt is mm
  elif m.get_type() == 'VFR_HUD':
      self.data['orientation'] = m.heading

  # added *
  elif m.get_type() == 'ATTITUDE':
    self.data['roll'] = m.roll
    self.data['pitch'] = m.pitch
    self.data['yaw'] = m.yaw
```

Listing 4.4: Receiving data packets from the MAVLink protocol

Client-side Communication

In the following the program of the client is described, which receives the data of self.data from the defined IP address 127.0.0.1 and UDP interface 25102 (chapter 4.3.1 source code 4.4), adds a time stamp and saves it to a file.
First, the client-side socket connection is established by binding the socket to the defined IP and the UDP interface (see source code 4.5).

```
UDP_IP = "127.0.0.1"
UDP_PORT = 25102

sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))
```

Listing 4.5: IP address and port definition and Socket for UDP interfaces readout

Afterwards, in listing 4.6, a function is set up that allows the document to be given a unique name. In the command prompt where the program is executed, a file name can be entered directly after calling the program. Otherwise the file is named workfile. With f = open(filename, 'w') a document is opened for writing. As long as the program is running, data received via the UDP interface is written to this document. The command seperator makes small formatting changes to the JSON object in order to append the timestamp with the Unix time time.time() in the next step. Finally, the JSON object is written to the document with f.write.

```
if len(sys.argv) > 1:
print filename
filename = sys.argv[1]
else:
filename= 'workfile'


f = open(filename, 'w')
while True:
  data, addr = sock.recvfrom(1024) #buffer size 1024 bytes
  print "received message:", data
  print time.time()
  dataJson = json.dumps(data, separators=(',',':'))
  dataJson = json.loads(data)
  dataJson["time"] = time.time()
  f.write(str(dataJson))
  f.write("\n")
```

Listing 4.6: Readout of UDP interface

4.3.2. Local and Global Position of Sensor Carrier

In this section the data acquisition process from the acoustic positioning system UGPS by Water Linked AS is presented. Before the parameter access can be started, the following network adjustments need to be made in order to access the data on a specific IP address. On the Master D1 main board the *Dual In-line Package* (DIP)-switch settings must be adjusted to the position as shown in fig. 4.3 and the computers static IP is, as already mentioned in section 4.3.1, set to *192.168.2.1*. This IP address can then be accessed from the computer and shows

Figure 4.3.: DIP switch settings for static IP address on *192.168.2.94*

the interactive Water Linked AS UGPS GUI, where parameters such as used locator, distance between receivers and Master D1 can be adjusted and also accessed by a self-written python script.
Following requirements apply to the data acquisition that need to be taken into account to gather data which can be processed further.

- continuous real-time acquisition from GUI
- local position in x-, y- z-coordinates and global position in latitude and longitude
- timestamp in Unix time format for each data

Based on the python scripts published on the official Water Linked AS git website [46], an own script, which meets the above mentioned requirements, has been developed. In the following, parts of the data acquisition program will be explained. The full listing can be found in annex C C.3.
In listing 4.7 all external functions that are called upon in the main program (see listing 4.8) are defined. The first function *get_data* tries to access the *Uniform Resource Locator* (URL) address given in the main program and prints out error messages if the website is not reachable or connection difficulties occurred.

The second function (line 26) then accesses the local (acoustic) and the third function (line 30) the global position of the locator. The last function enables the user to enter a specific filename for the received data in order to distinguish the measurements properly.

```
12  def get_data(url):
      try:
14      r = requests.get(url)
      except requests.exceptions.RequestException as exc:
16      print("Exception occured {}".format(exc))
        return None
18
      if r.status_code != requests.codes.ok:
20      print("Got error {}: {}".format(r.status_code, r.text))
        return None
22
      return r.json()
24
26  def get_acoustic_position(base_url):
      return get_data("{}/api/v1/position/acoustic/filtered".format(base_url))
28
30  def get_global_position(base_url):
      return get_data("{}/api/v1/position/global".format(base_url))
32
    #check correct filename input for string, float, and file format
34  def set_file_name():
      userinput = input("1. Please enter file name for position data: ")
36    return userinput
```

Listing 4.7: External functions for main programm

In listing 4.8 the main program is shown. The functions previously defined in listing 4.7 are called and their return value is taken. Thus, in the example of user input, this is set as the file name (see line 41). In line 44f. the URL address is passed to the parser and the received data is accessed. In the following a file with the given filename, in which the received data is written, indicated by 'w', is created. Further a timestamp is added before adding the position data to the data string. Finally the data string is transformed into JSON format and written into the file (s. line 60).

```
    def main():
40    print("Underwater GPS Data Acquisition")
      filename = set_file_name()
42    received_data = True

44    parser = argparse.ArgumentParser(description=__doc__)
      parser.add_argument('-u', '--url', help='Base URL to use', type=str, default='http
      ://192.168.2.94')      # getting real
      data
46    args = parser.parse_args()

48    base_url = args.url
      print("Using base_url: %s" % args.url)
50
      #data aquisistion and saving data in json file
52    f = open(filename, 'w')
      data = {}                 #creating dictionary
54    try:
        while received_data == True:
56      data.update({'time' : time.time()})
        data.update(get_acoustic_position(base_url))
58      data.update(get_global_position(base_url))
        dataJson = json.dumps(data, separators=(',',':'))
60      f.write(str(dataJson))
        f.write("\n")
```

Listing 4.8: Main programm of acoustic positioning data acquisition from IP address

In the following listing 4.9 example data in JSON format can be seen.

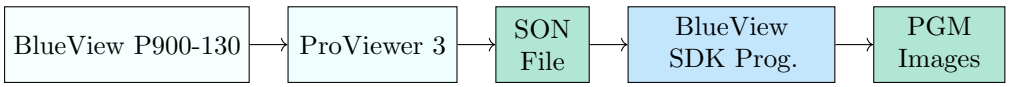

```
{"time":1581086055.0588472,"std":0.6361597587537569,"temp":0,"x":13.61597587537568,"y"
    :30.966764198635598,"z":12.723195175075137,"cog":0,"fix_quality":1,"hdop":1.9,"lat"
    :63.44299179016037,"lon":10.423280187793308,"numsats":2,"orientation":59,"sog":0}
```


Listing 4.9: Example data acquired by program

The time is in Unix time format, local coordinates such as x-, y- and z-coordinates are in meters, global coordinates such as latitude (lat) and longitude (lon) are in decimal degrees. Values for the orientation are gathered from the ROV itself. The values *Course Over Ground* (COG) and *Speed Over Ground* (SOG), are zero as no second GPS-antenna, needed for a referenciation, is implemented for evaluating them. Further, they will not be used in the progressing data fusion.

GPS-Antenna
The source code for sending *National Marine Electronics Association* (NMEA) data by a serial interface, which is executed on the Arduino Nano 3.0 [1], was provided for this thesis and will therefore not be described in depth. Shortly described, the GPS-antenna sends NMEA-0183-strings to the Arduino Nano 3.0, which then transfers the timestamp and GPS coordinates by an *Universal Asynchronous Receiver* (UART)-USB bridge to the TC. Please refer to the annex section C listing C.4, where the code is listed. By an UDP connection, using the NMEA Underwater GPS bridge [47], the data is then transferred to the web-based GUI of the underwater acoustic positioning system. The used NMEA Underwater GPS bridge, for accessing the GPS data and transfer to the IP *192.168.2.94*, was also not a part of this thesis. The executable program code, which was downloaded and connected to both systems by the command line interface, can be found on the official Water Linked git page [47].

4.3.3. SONAR Image Acquisition and Preprocessing



The acquisition of the SONAR images, gathered by the BlueView P900-130, can be divided into the pure data acquisition and the preparation for further processing of the single images. First, the manufacturer's software ProViewer 3™, is used to enable real-time data acquisition with simultaneous display. In the following the SDK is used for further processing and preparation of the data. The latter is described in more detail in section 4.4. The single steps of the preprocessing will be presented hereinafter.
Once the SONAR is lowered into the water, the ProViewer 3™, software is connected to it by its IP address. Once connected the acoustic backscatter is displayed on screen of the TC. Intensity and threshold can be changed manually in order to facilitate object identification for the user. Furthermore the range and field of view can be altered according to the target under investigation. The incoming data can be then recoded and saved into the BlueView proprietary file format with file ending *.SON*. The *.SON* file contains every ping from the multi-beam SONAR saved in binary format and can only be processed by using the BlueView SDK or other programs from the manufacturer. In the following the requirements and program sequence will be presented.

Requirements regarding the transformation of the BlueView proprietary *.SON*-format:

- Timestamp for each ping (same length/unit)
- Transformation of each ping in processable image format (*.Portable Grey Map (PGM)* format is provided by BlueView SDK)
- Unique filename by using the timestamp necessary for the comparison with the data records of motion parameters and positioning data

The BlueView SDK, even though using the programming languages C or C++, uses eight classes (Sonar, Head, Ping, MagImage, ColorMapper, ColorImage, Logger, and Error) in the form of member functions. From these only the Sonar, Head and Ping classes with the prefix BVT are used in order to minimize name space pollution. Hereinafter the main used classes and their sub functions are presented in table 4.1.
These classes and corresponding sub functions were used in the full *.SON* file to *.PGM* image transfer

Table 4.1.: BlueView SDK variables used in listing 4.10 to 4.14 and its functions [12]

BlueView SDK variable	Function
BVTSonar	Top level object, represents the communication to a SONAR or SONAR file and the creation of new *.SON* file
BVTSonar_Create	Creation of SONAR object
BVTSonar_Open	Open *.SON* file by filename or open connection to SONAR
BVTSonar_GetHead	Retrieve head object
BVTHead_GetPing	Retrieve Ping from Head object
BVTHead_GetPingCount	Count the amount of Pings stored in file
BVTPing_GetImageXY	Retrieve XY-format image, according to parameters set in Head
BVTPing_GetTimestamp	Get a timestamp for the recorded ping
BVTMagImage (MagnitudeImage)	Gives access to two-dimensional 16-bit greyscale image
BVTMagImageSavePGM	Saves image in *.PGM*-format

program, which can be found in annex C listing C.5. In the following, only some extracts and the rough program sequence will be explained.
Listing 4.10 describes a short program section on image storage in a specific folder. For this purpose, the extension *.SON* is subtracted (line 27) from the entered file name and a corresponding folder is created (line 28). This sequence was implemented because of the large amount of data, to be more precise the large number of pings of the recorded sequences, in order to ensure easy access to the data by the main two- to three-dimensional conversion program.

```
22      if (argc >= 2){
        strcpy(DataFile, argv[1]);
24      }
        int filenamelength = strlen(DataFile);
26      char foldername[256];
        strncpy(foldername, DataFile, filenamelength - 4);
```

Listing 4.10: Storage location of the sonar images derived from the *.SON* file

In the following a SONAR object will be created and the given *.SON* file will be loaded (see lst. 4.11 line 34 and 44). The filename of the *.SON* file is accessed through the char Datafile, which is linked to the input *.SON* file. BlueView also holds the option to access one of multiple heads, in this case only one head is used, therefore the first head on line will be accessed in line 48. In line 52 the total amount of pings stored in the *.SON* file is counted starting from 0 (line 51). The returned integer will be later used in a conversion progress indicator bar to estimate the total conversion time of the file.

```
    // Create a new BVTSonar Object
36  BVTSonar son = BVTSonar_Create();
    if( son == NULL )
38  {
      printf("BVTSonar_Create: failed\n");
40    return 1;
    }
42
    // Open the sonar file
44  BVTSonar_Open(son, "FILE", DataFile);

46  // Get the first head
    BVTHead head = NULL;
48  BVTSonar_GetHead(son, 0, &head);

50  // Check the ping count
    int pings = -1;
52  pings = BVTHead_GetPingCount(head);
```

Listing 4.11: Creation of Sonar object and following head and ping count

In listing 4.12 the main loop for each ping in the file starts (see line 55). A Ping object is created (line 57) and accessed by BVTHead_GetPing from the head (line 58).

```
54  // Begin loop to create image for each ping
    for (int i = 0; i< pings; i++){
56
      // Now, get a ping!
58    BVTPing ping = NULL;
      BVTHead_GetPing(head, i, &ping);
```

Listing 4.12: Beginning of for-loop for each Ping in *.SON* file

In listing 4.13 a method is shown to implement the timestamp for each ping image to its corresponding filename. At first the timestamp itself is accessed by the function BVTPing_GetTimestamp from the ping (line 61). The timestamps created automatically by the SONAR differ occasionally from each other by means of digits. Which results in an incorrect transformation into milliseconds. Therefore an if-loop is started in order to even out the resulting missing zero inbetween seconds and milliseconds, so that the timeformat is standardized and can be used as a reference in the further allocation of depth, motion paramteres and location coordinates in section 4.5.

```
60    //Get Time stamp for ping
      double time = BVTPing_GetTimestamp(ping);
62    double s;
      int n;
64    s = time;
      s = floor(s);
66    double ms = time - s;
      char buf3[100];
68    n = sprintf(buf3, "%d%d", (int)s, (int)(ms * 1000));
      if (n < 13)
70    {
        if (n = 12)
72      {
          sprintf(buf3, "%d0%d", (int)s, (int)(ms * 1000));
74      }
        else
76        sprintf(buf3, "%d00%d", (int)s, (int)(ms * 1000));
      }
78
      const char* timestampchar = buf3;
```

Listing 4.13: Timestamp implementation into filename of each image

Finally the image is created by the function BVT_GetImageXY and saved into a *.PGM* image by using the function BVTMagImageSavePGM with the filename as a combination of img (standing for image) and the timestamp as well as the ending *.PGM* (see lst. 4.14). Beginning from line 90, the combination of img, timestamp and file format is presented. Therefore the folder name is copied by strcpy in result1, a character with 256 positions, which is then extended by a doubled backslash. This first part declares the path where the image will be stored to. Following, the filename1 and the timestamp (as a char) will be appended, finishing with the specific file format ending *.PGM*. The process bar in line 102 prints the current to be converted ping of all pings in order to show the approximated duration of the program. At the end the of each ping to image conversion BVTMagImage, BVTPing are destroyed and when the full for-loop is finished also the BVTSonar objects is cleaned up in order to prevent memory leaks [12].

```
84      // Generate an image from the ping
        BVTMagImage img;
        BVTPing_GetImageXY(ping, &img);
86
        //Save it to a PGM (PortableGreyMap)
88      char result1[256];
        const char *filename1 = "img";
90      const char *format1 = ".pgm";

92      strcpy(result1,foldername);
        strcat(result1, "\\");
94      strcat(result1,filename1); // copy string filename1 into the result.
        strcat(result1,timestampchar);
96      strcat(result1,format1);// append string format1 to the result.
        BVTMagImageSavePGM(img, result1);
98
        //progress indicator
100     std::cout << "\rProgress: " << i << " out of " << pings << " images converted!";

102     //Clean up
        BVTMagImage_Destroy(img);
104     BVTPing_Destroy(ping);
        }
106
    BVTSonar_Destroy(son);
108 }
```

Listing 4.14: *.PGM* image creation

In the following section, the preprocessing of the recorded and transformed SONAR images will be presented, by means of a threshold application, in order to filter out all pixels with the strongest intensities, and a median 7x7 filter.

4.4. Image Processing

In order to spatially represent the reflection values of each sonar image pixel, different algorithms must be applied to create a detailed image. To reduce the impact of background noise, which results from particle backscatter in the water column and a reflective measurement environment such as harbour areas, a Gaussian Z-Test is applied to the sampled data in order to estimate a threshold for the spatial data plot, according to recommendations presented in [19]. In subsection 4.4.1 the main principle of this test will be presented. Hereinafter the SONAR images will be filtered to reduce uneven edges and outlined pixel. Therefore a median-7x7 filter will be elucidated in subsection 4.4.2.

4.4.1. Threshold Estimation

Threshold Estimation

As shown in chapter 2.2, a high number of underwater structures with high reflectivity (e.g. ships, metal piles etc.), narrow, angled test areas and also the reflection of acoustic waves from suspended matter in the water column lead to a background noise, which distorts the backscattered signal from the investigated area. A threshold represents the value from which the received signal is processed further. In this case, it refers to the pixel value of an 8-bit greyscale image, therefore a value from 0 (black) to 255 (white) as shown in fig. 4.4. To evaluate the noise pixel

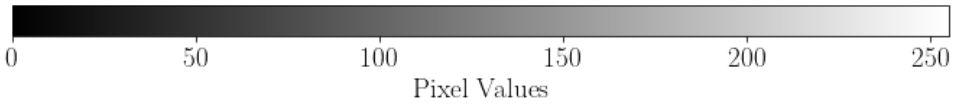
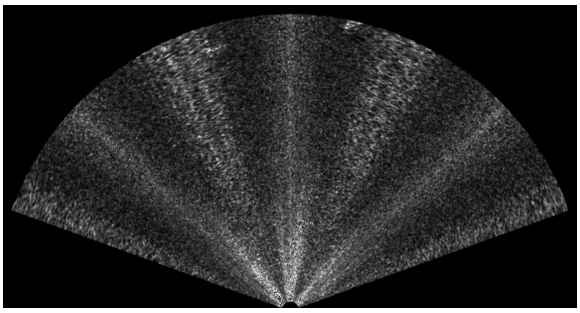
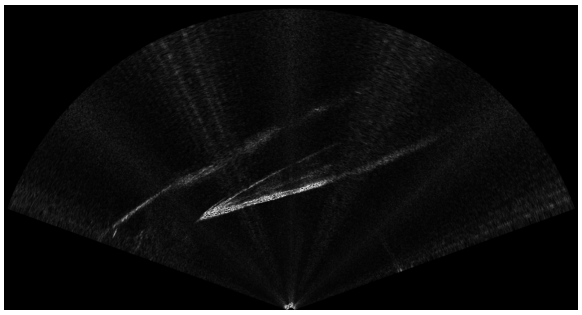


Figure 4.4.: Schematic of 8 bit binary pixel distribution from 0 to 255

distribution, sample data of the test area is collected and transformed from the BlueView P900-130 proprietary format to a 16-bit greyscale image in PGM-format. A further downsampling to 8-bit is necessary for the visual display on a regular computer. For the noise evaluation the same settings (range, threshold and intensity regarding the ProViewer 3™software settings) were used as for the recording of the actual data. In contrast, however, only those images were used on which no underwater objects are visible, that means an open-water scene (such as displayed in fig. 4.5 (a)). (In order not to anticipate the results of the thesis at this point, images, shown in fig. 4.5, from another measurement will be used to explain the procedure) In addition, data from different depths were included in order to eliminate stratification of particles in the water column as well. Comparing the two backscattered images in fig. 4.5, it can be noticed that the open-water image (a) comprises significantly more values in the range of 128 to 256 than the object-oriented image (b). Furthermore it contains strong ray-shaped noise additive to the general noise distribution across the whole image. In order to eliminate most of the noise, the Z-test (Gauss test) is applied. This will be explained in the following.



(a) Backscatteres signal from open-water scene



(b) Backscattered signal of the hull of the "Hansa" (Fischereihafen 1, Bremerhaven, Germany)

Figure 4.5.: Comparison of SONAR 8-bit greyscale images

The Z-Test (Gauss Test)
The Z-Test is a statistical hypothesis test, which uses the mean values of the sample and the population to make a statement about the expected value. It is assumed that the population, in this case the noise pixel allocation, is normal distributed. Reference for the procedure of the Z-Test, for equation 4.1, eq. 4.4, eq. 4.5 and eq. 4.6 is made to Kähler [24], where further information regarding hypothesis testing by means of the Z-Test can be derived.

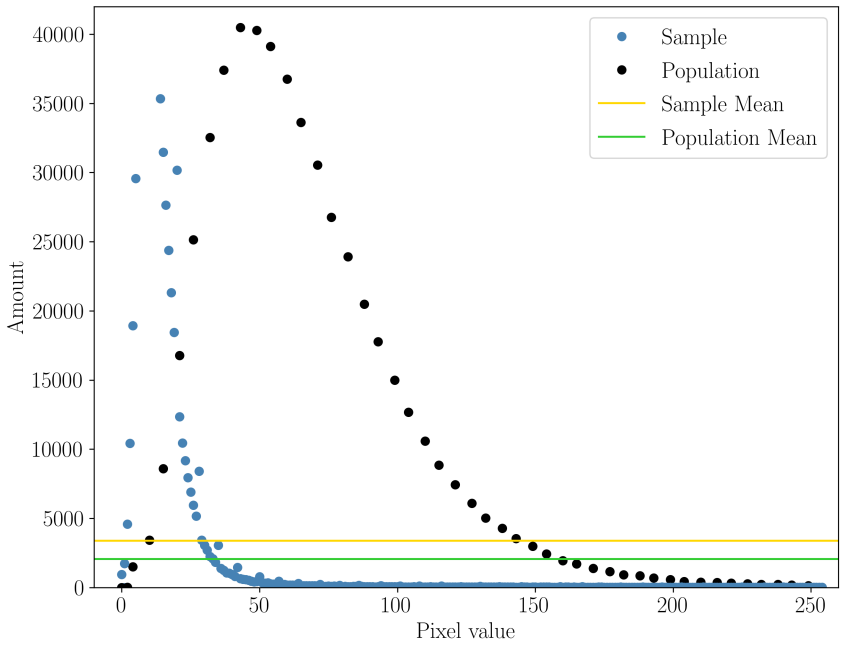


Figure 4.6.: Comparison of open-water and scene image from fig. 4.5 by means of pixel distribution

Figure 4.6 shows the pixel distribution of the noise image 4.5 (a) (black) and the scene image 4.5 (b) (blue). The vertical axis shows the total pixel amount for each pixel value, the horizontal the pixel value in the range from 0 to 255. All pixel values that did not occur were omitted to avoid a massive accumulation of zero values and to preserve the overall impression of the image. Furthermore, the zero value was neglected for both datasets, since only the values other than zero can be used to make a statement about image noise. For the application of the Z-Test, certain assumptions must be made in order to be able to formulate hypotheses about the pixel distribution.
Even though the noise image pixel allocation resembles the normal distribution only on the right hand side of the maximum value, it is nevertheless assumed as normal distributed. It is further assumed that the mean value μ of the sample will be larger than the mean value μ_0 of the population. Hence a right-hand test is applied. The Null hypothesis H_0 in eq. (4.1a) is rejected when the Alternative hypothesis H_1 in eq. (4.1b) is true.

$$H_0 : \mu \leq \mu_0 \tag{4.1a}$$
$$H_1 : \mu > \mu_0 \tag{4.1b}$$

The arithmetic average μ_0 of n populations with the normally distributed variables x_i are first formed and then averaged following eq. 4.2. The arithmetic average μ for the sample is also calculated by eq. 4.2 and for the standard deviation eq. 4.3 is used.

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.2}$$

$$s = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \mu)^2} \tag{4.3}$$

Following, the standard error of the arithmetic average of all samples is calculated by dividing the corresponding standard deviation s by the number of different pixel values n, see eq. 4.4.

$$SE = \frac{s}{\sqrt{n}} \tag{4.4}$$

The Z-value is defined as the difference between the arithmetic average of the sample and the population divided by the standard error, see eq. 4.5.

$$Z = \frac{\mu - \mu_0}{SE} \tag{4.5}$$

For evaluating the one-tailed positive hypothesis H_1 a critical value (or α level) of 0.05 is selected. The

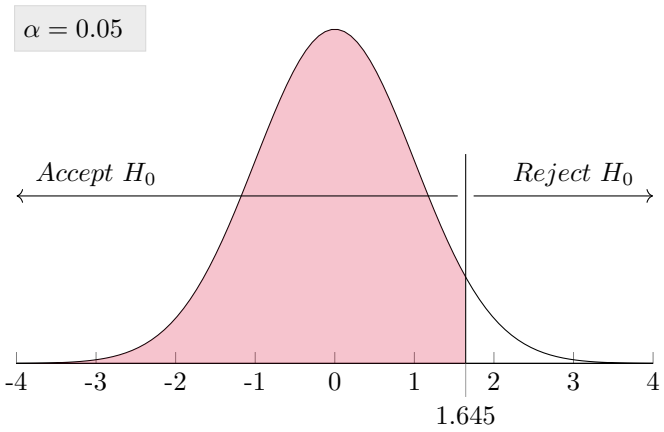


Figure 4.7.: Gaussian normal distribution with of α 0.05, respectively z=1.645

critical value indicates the probability of making a mistake type-I error, which means falsely rejecting the null hypothesis when it is actually true. For example this could be true when using sample images with small objects in a low reflective environment, resulting a small amount of large pixel values and a high amount of zero values. With the use of a Z-table the Z-value can be used to determine the corresponding probability factor p. p is the empirical quantile (as ranging from 0.1 to 0.99 it counts to the percentile) and stands for the value for which $p \cdot 100\%$ of the values are below or $(1-p) \cdot 100\%$ above the Z-value. By using the predefined critical value of 0.05, p must be larger than 0.95 in order to reject the Null hypothesis H_0.

$$p > (1 - \alpha) \tag{4.6}$$

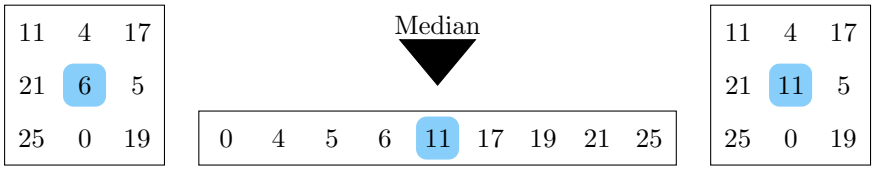
In this example the Z-value of 1.741 leads to a p value of 0.959. Hence the one-tailed positive hypothesis H_1 is true. The threshold value now results from the product of the probability value p and the pixel percentage ratio as shown in eq. 4.7.

$$t \approx 255 \cdot p \tag{4.7}$$

The respective source code can be found in annex C listing C.6.

4.4.2. Filter Algorithms

According to Petillot et al.[34] a simple median 7x7 filter applied on SONAR images, maintains the accuracy of the image while filtering out outliers (noise). In contrast to the mean filter, which distorts edges, the median filter has a better edge sharpness. In figure 4.8, the filter principle is described using a minimized version of a 3x3 median filter. The pixel values in a defined area around the origin pixel, s. fig. 4.8 (a), here marked blue, are sorted by size fig. 4.8 (b). The middle value of this array is then chosen and replaces the origin value, as can be seen in fig. 4.8 (c).
Figure 4.9 compares the original SONAR image of the ship hull of the "Hansa" in (a), the same image



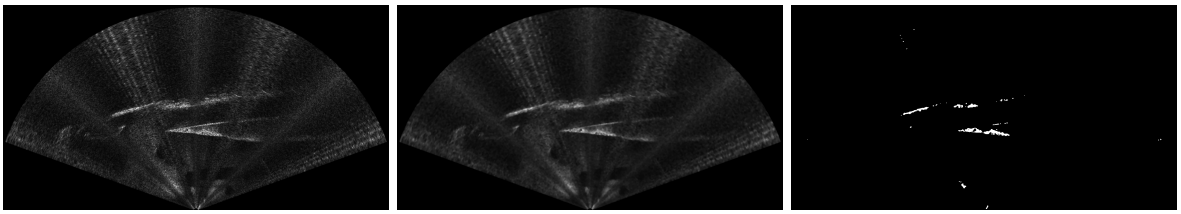
(a) 3x3 image matrix, origin pixel marked blue)

(b) Pixel values sorted by size, median marked blue

(c) New 3x3 image matrix with changed origin value

Figure 4.8.: Median 3x3 filter

processed with an median 7x7 filter (b) and in (c) additionally filtered with a threshold of t=110. It can be noticed, that the ship hull in fig. 4.9 (b) appears smoother with continuous edges. When applying the threshold the ship hull and keel become more visible, by the strong contrast of zero and one values, which is needed later in the spatial representation as a point cloud. SONAR images require practice in interpreting, so it can be said at this point that the upper edge describes the keel, the lower edge the outer wall of the ship. A coloured representation can make interpretation easier. However, this does not play a role in later data processing.



(a) Original image

(b) 7x7 median filter

(c) 7x7 median filter and threshold t = 110

Figure 4.9.: Comparison of the processed, backscattered SONAR image of the hull of the "Hansa"

The advantages of blind deconvolution filter, as recommended according to Guerneve et al. [19], could not produce a significant gain in information, using the in sec. 4.5 described method of spatial representation, and were discarded due to the time-consuming programming run-time.

4.5. Sensor Data Fusion and Three-Dimensional Representation

Main Program

In this section the combination of all three data sets will be presented as well as the algorithms for the two- to three-dimensional representation of the SONAR images. Extracts from the program are given in this section, all fully coherent listings can be found in annex C listing C.6 to C.8.
Figure 4.10 shows the program flow schematically. In the following, the topics presented in the arrows will be dealt with in an anti-clockwise direction, beginning at the top, according to the program sequence.
Before execution of the program, it has to be ensured that all to be considered *.PGM* images, as well as the motion and position data files are located in the same program folder. The images are called by a for-loop for every file ending with *.PGM*, whereas the motion and position data filenames need to be called directly. Careful handling of the correct images of the objects under investigation is required.

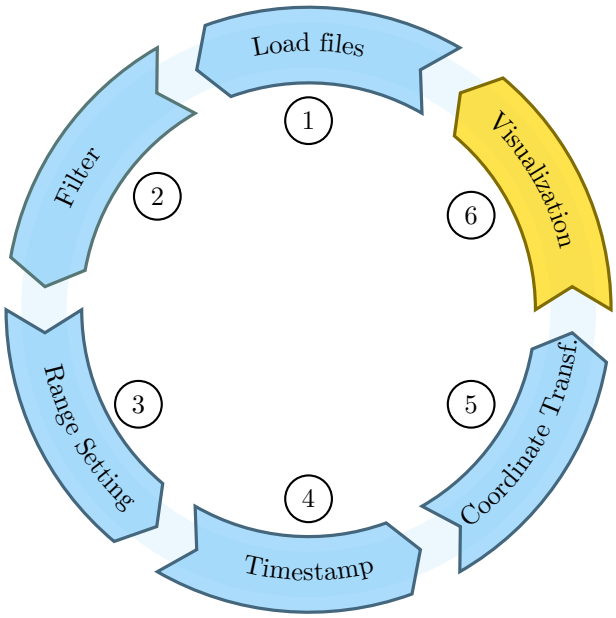


Figure 4.10.: Schematic of the program flow

In the interests of clarity, the main aspects of the program are set out below.

① Loading *.PGM* files into program
② Filter Algorithms (Threshold, Median 7x7)
③ Range setting and pixel to meter conversion
④ Timestamp allocation of all three data sets
⑤ Implementation of the roll-, pitch- and yaw-angle through coordinate transformation of backscattered intensity values
⑥ Utilization of open3d python package for three-dimensional point-cloud visualization of intensity values; preliminary appending of all points until no *.PGM*-files are left and only then display of the point cloud

As the thresholding and filter algorithms, see fig. 4.10 ②, haven been presented in section 4.4 this topic will be skipped here.

4.5.1. Loading of Images

As already mentioned above, the loading of the SONAR images is wrapped by a for-loop, which applies to all files that end with the file format *.PGM* in the respective folder of the program. Each *.PGM* file consists of a header and image data, the former data consist of a magic number, which stands for the file format, the image width and image height all separated by blanks. An example of first row of an converted *.SON* to *.PGM* file can be found in fig. 4.11. The gray-shaded one indicates the first line of the

```
1 P5 1591 843 65535
```

Figure 4.11.: Exemplary .PGM files first row

.PGM file, see fig. 4.11. The magic number P5 stands for portable graymap binary, which is then followed by the width, height and value range from 0 to 65535 (16 bit), which are coded in decimal *American Standard Code for Information Interchange* (ASCII).
To gather image meta data the image is read in binary and then split up to the fourth element of the image see 4.15. The resulting array can be see in lst. 4.15 line 2.

```
file.read().split()[:4]
2 >>[b'P5', b'1591', b'843', b'65535']
```

Listing 4.15: Binary array of meta data

From this array the width and height are accessed by the array index number, see lst. 4.16 line 195f., and transformed into an integer. In the following the logarithm of the value range on base 256 is "ceiled", which means returning the smallest integer not less than the return value of the logarithm. This returns the bytes per value which are then multiplied with the image width, which results in this case in a image width of 3182.

```
194 #access with and height, adjust image width
    width = int(fields[1])
196 height = int(fields[2])
    bytesPerValue = int(ceil(log(int(fields[3]), 256)))
198 nslices = width * bytesPerValue
```

Listing 4.16: Meta data access of *.PGM* file

The actual binary image data is then read, from the position after the header data (position 18 in this example) onwards, into an unsigned 8-bit array b. This array is then reshaped into an array with the height and adjusted width (nsclices) according to the image dimensions, lst. 4.16 line 5.

```
    #get image data
202 imageData = []
    for i in range (0,len(b)):
204   x, maxi = parseLine(b[i])
      if globalMax < maxi:
206     globalMax = maxi
      imageData.append(x)
208
    globalMax = int(globalMax * globalMaxMultiplicator)
```

Listing 4.17: Global maximum compare

For each value in each row of the byte array b it is now checked whether it is the maximum value of the whole image (s. lst. 4.17). If the global maximum (globalMax) is smaller than the value (maxi), maxi is written into the variable of the global maximum. Furthermore the function parseLine gives back an array with the length of the original width of 1591 pixel, which is appended to the array imageData.
Then the globalmax is multiplied by 0.25 (globalMaxMultiplicator), which estimated by scaling the highest intensity value in the 16-bit to a maximum value of 255. This value needs to be adjusted for each data set manually.

In the following a for loop from 0 to the length of the array imageData (also with the value of 843) and a second for loop from 0 to len(imageData[1] which is 1591), runs through every element of the image and divides it through he quotient of globalMax and 256, see lst. 4.18. Finally the image is created by the image module Pillow [29]. The rotation (line 216) needs to take place in order to adjust the differing image origins of the array image (bottom left corner) and Pillow-images (top left image corner). Therefore the images is saved in x-y-coordinate system, such as presented in section 4.3.3.

```
212   data = np.zeros((height, width, 3), dtype=np.uint8)
      for i in range (0, len(imageData)):
214     for j in range (0, len(imageData[i])):
          data[i, j]=(imageData[i][j] / (globalMax / 256), imageData[i][j] / (globalMax /
        256), imageData[i][j] / (globalMax / 256))
216   img1 = Image.fromarray(np.rot90(data,3), 'RGB')
```

Listing 4.18: Save in RGB image format

Both, the ROV motion data as well as the UGPS position data are simply opened and read as JSON files and then read line by line into an array according to their JSON key (s. lst. C.8 line 38ff. and 64ff.).

4.5.2. Range Setting

In contrast to the, in section 2.1 presented, method for obtaining backscattered intensity values from the SONAR in polar coordinates, the main program utilizes rows and columns for obtaining pixel values. In order to match pixel and the set detection range a scaling factor needs to be applied. The method used shall now be explained by means of figure 4.12.

The shown image displays a high noise environment or simply an enhanced backscattered intensity,

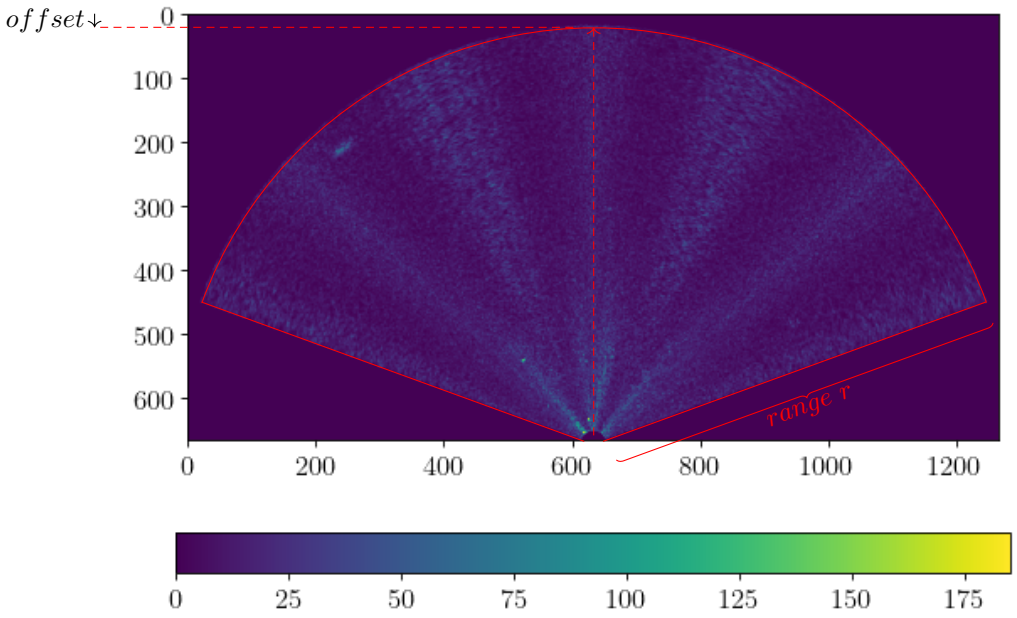


Figure 4.12.: Range Test Image

ranging from 0 to approximately 150 pixel values, which is required to detect the full field of view of the SONAR, here marked in red. In order to enhance the contrast for visualization, the originally grey-scaled image was pseudo-coloured. The pixel count starts at the top left corner of the image (Pillow-image), which can be recognized by the legend direction. To scale the image, the farthest valid pixel value counting from the centre of the image in vertical direction has to be determined. In this example, the farthest pixel

is located in row 19, which can also be defined as the pixel offset. By subtracting the pixel offset from the total image height, the pixel number for the range r is derived. This value can then be entered into the main program. The corresponding script, which reads in multiple images and calculates a mean value to enhance the accuracy, can be found in annex C listing C.7. This step cannot be generalized for all SONAR images, as they differ in dimensions according to their range, therefore the script needs to be executed for all taken measurements, when changing the range settings.
Listing 4.19 line 230ff. shows the implementation of this pixel range into the main program. The range (in meter) itself is entered in line 231, according to the settings during data recording. In line 233 to 234 the origin is moved to the bottom centre of the image. For this purpose, half of the width of the image is subtracted (np.add) and stored in the variable img_x. The height of the image is kept and only assigned to a new variable (img_y). im_xy contains indices of the image pixels that are non-zero (also known as a tuple) and above a threshold, using numpy.nonzero() and the Pillow function Image.point(). The tuple arrays containing the x- (horizontal/width) and y- (vertical/height) coordinates can then be accessed by calling img_[0] and img_[1]. In order to scale each axis according to the scaling factor, derived in line 231, it is multiplied to each tuple array as can be seen in line 240f..

```
230     #pixel amount for range
        real_sonwindow = 165   #enter correct value for each measurement
232
        #move zero to bottom centre of image
234     img_y = img_xy[1]
        img_x = np.add(img_xy[0], - img.size[0])
236
        #transform pixel to meter
238     max_range = 10   #get value data recording (ProViewer)!
        scaling = max_range / real_sonwindow
240     img_x = np.multiply(img_x, scaling)
        img_y = np.multiply(img_y, scaling)
```

Listing 4.19: Range Setting of SONAR image

The complete listing can be found in annex C listing C.7.

4.5.3. Timestamp Allocation

The timestamp of each SONAR image functions as the starting point for allocating additional data with the closest timestamp. As explained in section 4.3.3 the timestamp of each image is stored in the image name, in the motion and positioning files the timestamp is stored as a JSON object. The command line in source code 4.20 is applied to both the motion and the position data sets and returns the JSON object whose timestamp is closest to the image timestamp. Therefore all values in the data set at position 0 (data[:, 0]) will be subtracted by the timestamp of the SONAR image. From each array element the amount is taken (np.abs) and the smallest element (np.argmin) is stored in the variable index. As in this case the JSON object consists just of one single row without any array elements, it can also be defined as an index return function. Listing 4.20 shows the procedure examplary for the ROV motion and depth data, which also applies to the position data from the UGPS.

```
        index = np.argmin(np.abs(np.add(data[:, 0], -timestamp)))
56      ROV_return = data[index, :]
```

Listing 4.20: Acquisition of closest timestamp to SONAR image timestamp and return of motion (or position) data

4.5.4. Coordinate Transformation

Since a movable sensor platform is used for data acquisition, it cannot be assumed that the data is recorded exactly horizontally. Movements of the sensor carrier ROV are described by the roll- (ϕ), pitch- (θ) and yaw-angle (ψ) and applied to the acquired data by a coordinate transformation. Following the mathematical rules for matrix multiplication in coordinate transformation, the rotation matrix R in eq. 4.8 is derived.

$$R = \underbrace{\begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Yaw-Matrix}} \cdot \underbrace{\begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix}}_{\text{Pitch-Matrix}} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix}}_{\text{Roll-Matrix}} \tag{4.8}$$

An implementation of the global position of the ROV leads to the transformation matrix T shown in eq. 4.9. Distances respectively from the Bar30 or locator U1 are adapted manually. For the source code see annex C lst. C.8 line 127ff..

$$T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & UGPS_x + 0.15 \\ R_{21} & R_{22} & R_{23} & UGPS_y + 0.27 \\ R_{31} & R_{32} & R_{33} & depth + 0.25 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

The Rotation matrix R is here defined by its values in rows/columns from R_{11} to R_{33}. The variables $UGPS_x$ and $UGPS_y$ are transformed values from the latitude and longitude to the UTM projection and function as translation values. The translation in z-direction is described through the variable *depth*. It can be chosen from either the depth measurement taken by the UGPS or the Bar30 on te ROV. The values behind $UGPS_x$, $UGPS_y$ and *depth* describe the distance from the SONAR to the respective sensor. To be precise, the SONAR is mounted 0.27 m ahead and 0.15 m left of the U1 locator. Respectively it is mounted 0.25 m below the pressure sensor Bar30, which will be used in the measurement trial in section 5.1. Further it has to be noted that in section 5.1 the position will be presented relative to the Master D1 and globally. An adaption of the $UGPS_{x/y}$ is therefore necessary, when changing the coordinate system.

4.5.5. Three-Dimensional Visualization

For the three-dimensional visualization the python package open3d geometry [33] with the PointCloud class is used. For each backscatter value that is above the threshold value, the x-, y- and z-coordinates are stored in a vector, which is then loaded as a variable into the function plot_pc (see line 14 listing 4.21). First the open3d geometry PointCloud object is loaded into the variable *pcd* in line 15, then the points and finally the vectors. Thereby a transfer of the array takes place (see line 16). In line 17 the points are written into the point cloud pointcloud.ply. In the following this point cloud will be accessed as soon as all point are written into it and then visualized.

```
14  def plot_pc(pcvector):
    pcd = o3d.geometry.PointCloud()
16  pcd.points = o3d.utility.Vector3dVector(np.transpose(pcvector))
    o3d.io.write_point_cloud("pointcloud.ply", pcd)
18  pcd_load = o3d.io.read_point_cloud("pointcloud.ply")
    vishandle = o3d.visualization.draw_geometries([pcd_load])
```

Listing 4.21: Function for visualization with open3d

In the following step the MeshLab software is used to convert the point cloud from *Polygon File Format* (PLY) format to the commonly used *.xyz* format. The final display and analysis is performed using the NaviModel Free Viewer. A further representation in a two-dimensional coordinate is created, in order to access the exact positioning of the point cloud in the spatial representation, as in the NaviModel Free Viewer no presentation on an underlying coordinate grid is possible.

Understanding for the data acquisition of the motion and depth values, acquired by sensors on the Pixhawk 1, transfer by the MAVProxy, access by an UDP-bridge and saving into a common JSON file was presented. Further the acquisition of the positioning parameters, determined by the UGPS, by means of IP-address access and also storage in JSON-format followed. Finally the access and transformation of the SONAR images was discussed. The image enhancement by means of thresholding and the application of a median 7x7 filter demonstrated the image preparation before moving on to the final merging of all gathered data. This last step not only motion compensated, but also allocated the SONAR at the correct spatial position in a local and global coordinate system. Now moving on to how this complete software procedure is translated to the data, acquired in a measurement trial, executed in the Fischereihafen 1, Bremerhaven, Germany.

5. Implementation

In this section, the previously described hardware and software are tested within a measurement campaign. First of all, the measurement setup and the measurement target are presented. In addition, the test objects are presented, which are to be used for the following three-dimensional construction. The results of the individual sensor systems can be found in section 5.2. Finally, the three-dimensional representation of the test objects is presented by the combination of all sensor data.

5.1. Methodology

The aim of the measurement is to replicate the spatial sonar image formation presented in section 2.1. Therefore reference objects are mounted on a board and lowered into the harbour water. In order to specify the quality of the three-dimensional capabilities of the presented program in section 4.5, four different measurements on reference objects will be carried out:

- Cuboid
- Prism 10°
- Prism 20°
- Rectangle

The color coding of the individual reference objects is taken up in the course of the measurement results in order to facilitate the clear assignment in data plots.

5.1.1. Outdoor Measurement Setup

In fig. 5.1 (a) and (b) the measurement setup on a pontoon at the Fischereihafen 1 in Bremerhaven, Germany is shown. In the following description the reference point for all coordinate information of the test setup is the TC.
The TC in connected via the tether spool to the ROV. Furthermore the ROV is equipped with the in section 3 described SONAR and U1 locator, mounted on the left side in the direction of movement. The receiver of the underwater positioning system are lowered into the harbour water to a depth of 2 m, 2.5 m, 1.5 m and 2.2 m (from receiver one to four in that order) with a distance to the Master D1 board (located in the TCP) with respectively 4 m to the sides and 0.75 m to the front and 1.65 m to the back. The GPS-antenna is located close to the Master D1 in the TC and can therefore not be seen. On fig. 5.1 (b) the distance holder, which enables the reference objects to be recorded with the field of view in the direction of the open port basin can be seen looming over the edge of the pontoon. If the objects were lowered directly on the outer wall of the pontoon (resulting in the field of view facing in the direction of the harbour wall/pontoon), the high ambient reflectivity would lead to high noise on the SONAR images. The board with the reference object is lowered at 2.83 m, 1.55 m to a depth of approximately 0.35 m (top edge of board) facing with the characteristic side towards the pontoon, in order to execute the measurement as mentioned above. In fig 5.1 (b) the field of view of the SONAR would therefore take measurements from bottom (near) to top (far), the ROV would be manoeuvred under the pontoon, facing the reference object.

(a) Measurement setup from side view, ROV in this image not at the measurement position

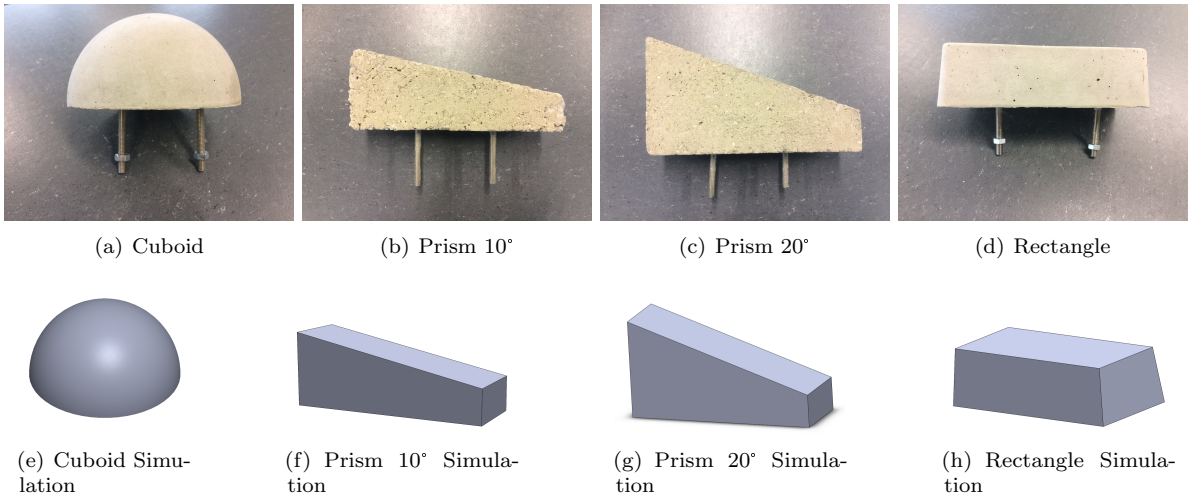
(b) Measurement setup from top view, ROV in this image not at the measurement position

Figure 5.1.: Outdoor measurement setup at the Fischereihafen 1

The measurement is carried out as follows: the four receivers of the acoustic positioning system are lowered into the harbour basin at certain distances from the TC and fixed to the pontoon to prevent the cable from slipping. Then the cables are connected to the TC and the power supply of the master D1 is ensured. The locator U1 is also switched on to establish a GPS lock. The TC will be switched on and the control software QGroundControl will be started as soon as the ROV is connected with a battery and to the TC by the tether cable. The Water Linked UGPS GUI is opened and all relevant distances from the receivers to the Master D1 will be inserted, as well as information about the measurement area (reflective environment) and the chosen locator and frequency channel. Furthermore the external GPS-antenna needs to be connected by the NMEA Underwater GPS bridge application and the GUI settings need to be changed to external GPS-antenna and onboard IMU. Care must be taken to ensure that the external GPS-antenna is connected only after the internal GPS-antenna of the Master D1 has received a GPS lock. This is required for time synchronization as a software prerequisite. After the calibration of the Bar30 depth sensor and a functional check of all thrusters, the ROV can be lowered into the water. Thereafter the corresponding programs for SONAR data acquisition, acoustic position and movement/depth parameters of the ROV can be started. The ROV is then manoeuvred towards the reference object using the GUI or by visual contact using the ProViewer 3™.
The actual test drive consists of recording the test object as frontally as possible at a constant distance at approximately x: 2.83 m, y: −0.45 m with a deviation of ±0.25 m in both directions. Then the ROV is lowered until the reference object is no longer recorded by the SONAR. This is repeated for all remaining objects.

5.1.2. Reference Objects

The reference objects as shown in fig. 5.2 (a) to (d) are constructions out of concrete using self-constructed wooden moulds for the Prism 10°and 20°and plastic moulds for the Rectangle and Cuboid. The material was chosen in such way to resemble the materials used underwater constructions in order to mimic a similar backscattering behaviour. Threaded rods were cast into the moulds with a form stabilising perforated tape to fix them during testing to a plastic board with nuts and locknuts. The board is lowered 0.35 m (top edge of board) into the water. For the virtual comparison also three-dimensional models were constructed as shown in fig. 5.2 (e) to (h). The dimensions of each reference object can be taken from



(a) Cuboid (b) Prism 10° (c) Prism 20° (d) Rectangle

(e) Cuboid Simulation (f) Prism 10° Simulation (g) Prism 20° Simulation (h) Rectangle Simulation

Figure 5.2.: Reference Objects and corresponding simulation

annex D table D.1. Due to the shape distortion of the plastic mould of the rectangular test object, the figures 5.2 (a) and (e) show rather a polyhedron. It is assumed that the minimal angular deviation has no significant influence on the image formation of the SONAR, thus this test object is still referred to as a rectangle.

Figure 5.3 (a) shows a close up view of the distance holder of the reference object, which is facing towards

(a) Distance holder of the reference objects from the pontoon

(b) Plastic board with reference object Cuboid

Figure 5.3.: Reference object suspension and mounting on plastic board

the pontoon. In fig. 5.3 (b) the plastic board, on which the reference object Prism 20°is mounted, is presented. The same alignment is also used for the Prism 10°. The Rectangle reference object is mounted also vertically (long side parallel to the board). The exact location of the reference objects can be taken from the annex D table D.2. The differences in depth are due to the different attachment to the plastic board.
In the following sections, the reference objects are continued as proper names in the continuous text in order to guarantee a clear identification of these.

5.1.3. Measurement Expectation

In this section the measurement expectation regarding the horizontal alignment of the backscattered SONAR images is presented.
Figure 5.4 shows the measurement setup from the perspective of the ROV and the expected SONAR image, with the origin located at the position of the Master D1 in the TC. For the global position the same origin applies, only in UTM-coordinates at 472 415.51 m, 5 930 441.29 m. It is expected that the resulting point cloud scene from the data sets of the individual reference objects will lie in the range from ca. −0.10 m to 6 m in x-direction and from ca. 0.15 m to 3 m in y-direction, depending on the intensity of backscatter from particles in the water column (s. fig. 5.4). The coloured axis markings are intended for later comparison with spatial representations of the NaviModel Free Viewer. Regarding the orientation of the point cloud in a coordinate system it is assumed that the representation along the x-y-axis will show the top view, along the x-z-axis a front view and along the y-z-axis a side view of the scene. Due to the close proximity of the SONAR to the receivers of the acoustic positioning system UGPS, the disturbance of the position data might be increased. Moreover, a deviation of the GPS position is estimated, in addition to the local position, within the range of the sensor accuracy.
Furthermore, it is anticipated that the descent of the ROV without the support of the thrusters and only the use of the stabilisation mode will have a sufficiently slow rate of descent so that the point cloud generated by the sonar at a frame rate of about 15 Hz, will have a corresponding number of horizontal image planes. It is assumed that the vertical length of the reference object will not match the recorded depth sequence of the sonar, as described in the theoretical background section 2.1, due to the large vertical beam width. Additionally the buoyancy chamber of the pontoon is also expected to be located in the field of view of the SONAR, due to the angled measurement area.

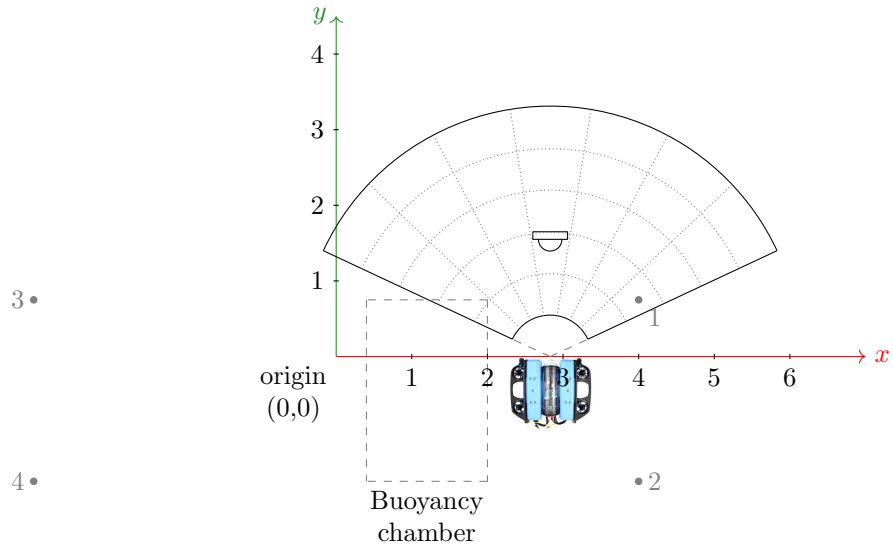


Figure 5.4.: Two-dimensional representation of whole scene expectation in top view along x-,y-axis; image of BlueROV2 from [6], reference object Cuboid enlarged, receiver numbered from 0 to 4

5.2. Measurement Results

In this section the results from the measurement trial in the Fischereihafen 1 described in section 5.1 will be presented. In section 5.2.2, sec. 5.2.3, sec. 5.2.1 and sec. 5.2.4 all four data sets will be compared graphically to each other by colour coding the individual measurements. Tables concerning the mean values and standard deviations of the movement parameters, number of points in each point cloud and point cloud dimensions are given in annex E table E.1 to tab.E.4.

5.2.1. Depth Measurements

In this section the measurement results from the pressure sensor Bar30 in the Pixhawk 1 on the BlueROV2 and from the acoustic positioning system UGPS are graphically displayed and described.
Fig. 5.5 shows the pressure depth profile and fig. 5.6 the acoustically determined and calculated depth, both on the (y-axis) according to the time (x-axis), colour coded in relation to the measured object.
The data excerpt for the Cuboid, Prism 20°and Rectangle dataset shows a descent from ca. 0.18 m to 0.22 m to a depth of 1.45 m. In the Prism 10°dataset the depth progresses parabolically from −1.2 m at 0 s over a maximum of 0.05 m at 8.5 s to −1.1 m towards the end of the measurement. It can be also noted that the descent in the Cuboid and the Prism 20°dataset are faster then the descent of the Rectangle dataset. The course of depth of the four datasets has been recorded almost without gaps, except for single outliers in the positive area of the graph. The curse of depth in fig. 5.5 is aligned to the curse of the pressure depth, but with significant steps of approximately 0.2 m every 1 s to 3 s, depending on the dataset.

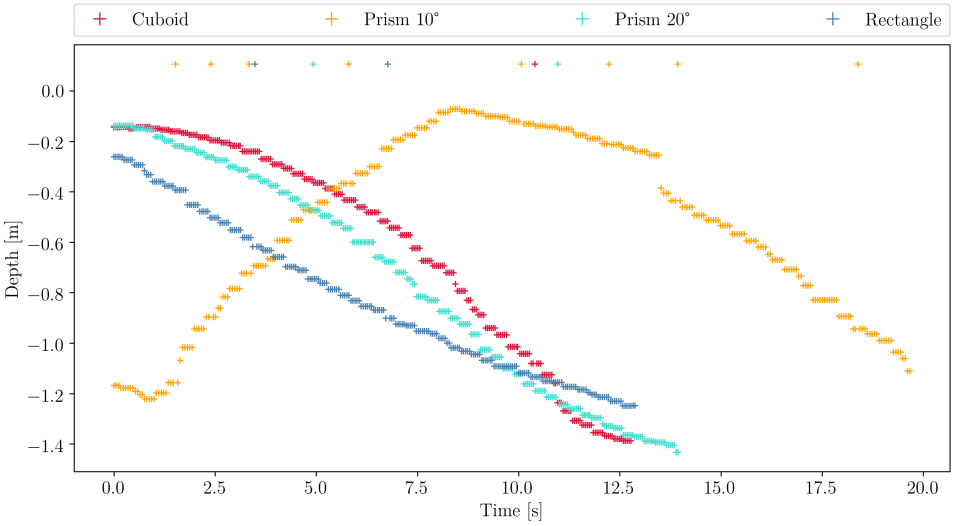


Figure 5.5.: Depth measured by pressure sensor Bar30

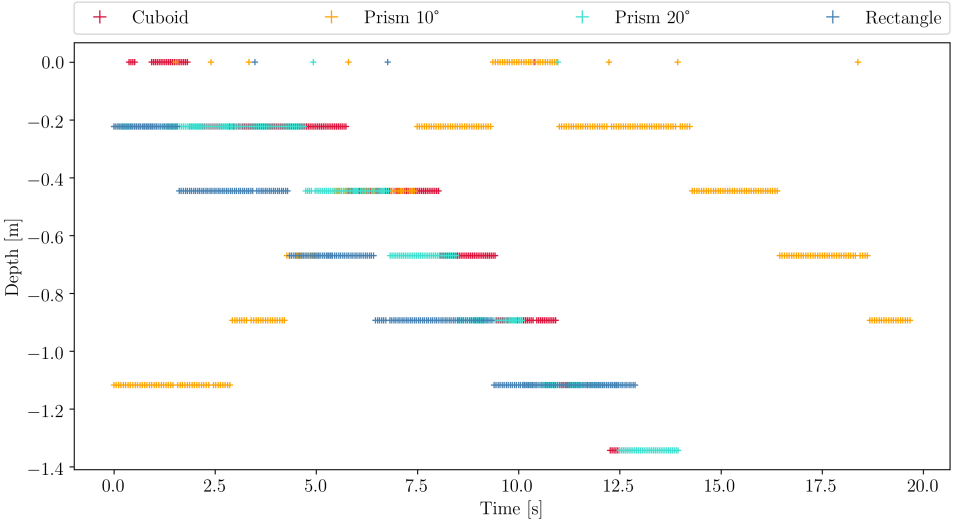


Figure 5.6.: Depth calculated by acoustic positioning system UGPS

5.2.2. Motion Parameters

In this chapter the motion parameters of the ROV during the recorded SONAR sequences of the four reference objects are presented.
As shown in fig. 5.7 the roll-angle indicates the deviation from the x-, the pitch-angle from the y- and the yaw-angle from the z-axis, all in degrees. Movements in the direction of rotation are considered positive, those against the direction of rotation are considered negative.

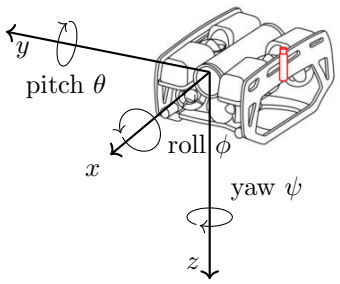


Figure 5.7.: Roll-, pitch- and yaw-angle in relation to the sensor carrier BlueROV2, according to [35] (adapted ROV schematic from [49])

All three angles in fig. 5.8, fig. 5.9 and fig. 5.10 are relative to the BlueROV2 (s. fig. 5.7) and gathered by the corresponding sensors on the Pixhawk1 Flight Controller (s. annex B tab. B.3). In all three angle plots the y-axis shows the the corresponding angle in degrees, the x-axis shows the timespan of the recorded dataset in seconds. In each plot, the four data sets are compared with each other, which is facilitated through the colour coding mentioned in sec. 5.1.
In the roll-angle plot in fig. 5.8 the angle of the Cuboid dataset is constant for the first ten seconds of the data set. For the last three seconds it comes to a short roll in the direction of rotation on 5.8°, which adjusts itself again to zero towards the end of the measurement. The Prism 20°and the Rectangle datasets roll-angles show a similar behaviour, as the former reaches a parabolic maximum of 2.3° at 5.5 s and the latter of 4° at 11 s. Especially in the datasets of the Rectangle and the Prism 10°a roll movement in negative direction can be observed when the measurement is started. In the case of the Prism 10°measurement this is much stronger with −5.6° in contrast to the Rectangular measurement with a minimum of −1.7°. Furthermore, fluctuations of the roll-angle from 0 to 10 seconds can be detected in the Prism 10°dataset.
The pitch-angle in fig. 5.9 follows mainly the same time pattern, but with higher deviations. For the Cuboid dataset this results in an angle of 8.8° at 12 s, for the Prism 20°dataset an maximum of 5.4° at 11 s and for the Rectangle dataset a deviation of 3.2° at 5 s. Also the absolute minima of the Prism 20°is increased by −7.2° and shows the course of an increasing negative exponential from 0 s to 8.5 s. The fluctuation of the Prism 10°dataset is also extended to second 13 s with a maximum of 1.7° and a minimum of −5.5°.
Figure 5.10 shows the yaw-angle, also commonly described as the orientation of movement towards magnetic north. It is noticeable that all angles of the four measurements are negative in the range from −10° to −22°. Further, each measurement does not end at the same degree where it has started. For example the Prism 10°dataset starts at 0 s with −10°, fluctuates around the mean value −15° with a standard deviation of −1.6° and ends at ca. 19.7 s with −15.4°. For further averages and standard deviations see annex E table E.1.

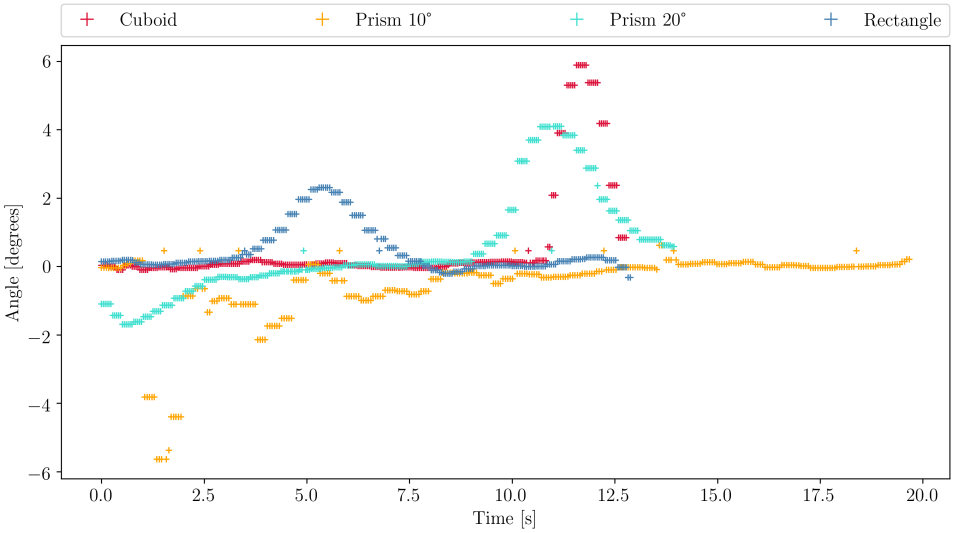


Figure 5.8.: Roll-angle comparison of the four datasets

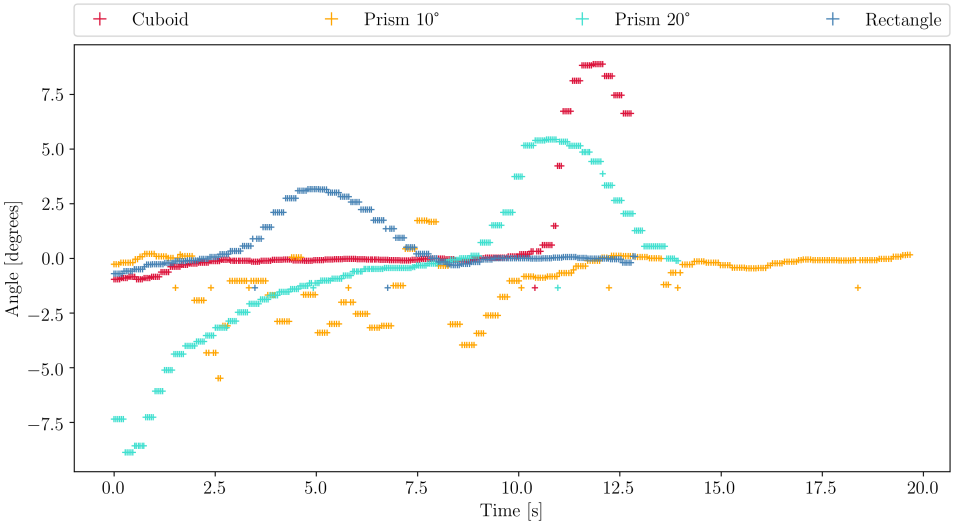


Figure 5.9.: Pitch-angle comparison of the four datasets

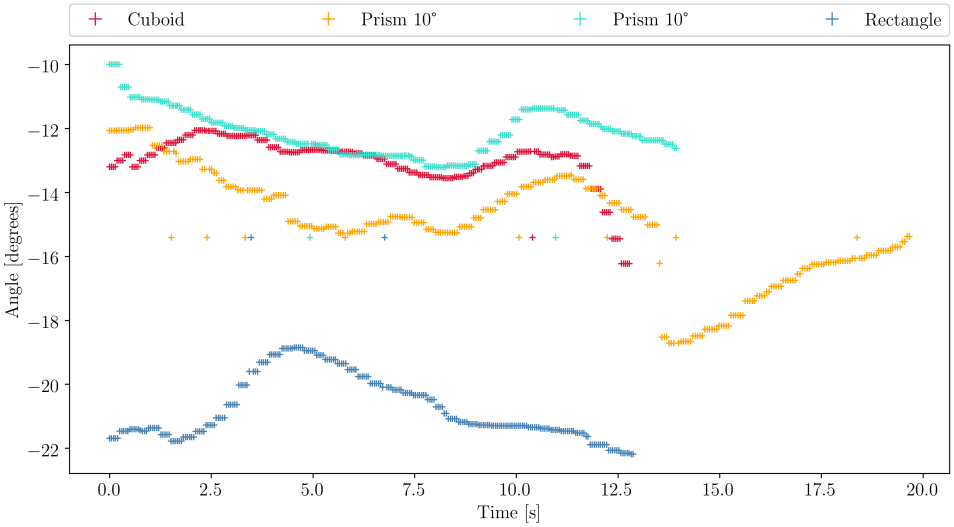


Figure 5.10.: Yaw-angle comparison of the four datasets

5.2.3. Short Baseline Positioning Data

In this section the gathered local and global positioning data of the acoustic positioning system UGPS will be presented by group plots of all four data sets. The three plots shown in this section (s. fig. 5.11 and fig. 5.12) present the data in a two-dimensional coordinate system, where the x- and y-axis represents the distance in meter, for the local position (s. fig. 5.11) this means the distance to the Master D1 in the TC, for the global position (s. fig. 5.16 and fig.5.12) it referrers to the global position of the external GPS-antenna on the TC additionally to the local calculated position. It has to be mentioned that the global position is presented in UTM-format and therefore deviates from the commonly know WGS 84-format of latitudes and longitudes.

In fig. 5.11 the local acoustic position and corresponding geometric median of the locator U1 on the ROV is presented in four different measurement sets distinguishable by colour coding. Furthermore, the temporal recording can be taken schematically from the colour gradient of the individual plots. From light to dark/intensive, the time increases over the course of the measurement. It can be stated that the positions are mainly distributed in the approximate range between 2.2 m and 3.7 m horizontally and 0.7 m and 1.4 m vertically, with the exception of one outlier at (0.8 m, 1.6 m), which was extracted at this point for the sake of clarity. Regarding the amount of outliers it can be noticed in Cuboid dataset it is the smallest with 1, followed by 2 outliers in the data sets of Prism 20°and the Rectangle (s. annex E tab. E.2.). The accumulation of outlier in the Prism 10°dataset is highest with 8. In the datasets of the Cuboid, Prism 20°and Rectangle, a horizontally shift of the local position occurs from larger to smaller x-coordinates, in contrast to the Prism 10°dataset, whose is opposite.

The global position in UTM 32 x- and y-coordinates, shown in fig. 5.12 presents an enlarged excerpt of the data set, with the identical colour coding, as in the local coordinate representation. An outlier at approximately $x = -505\,645$ m, $y = 1.6$ m was removed, in order to show the course of the position data. In particular, the large spatial distance between the four data sets and their floating, irregular displacement from the origin (lighter line) of each measurement can be seen. In the lower left corner a black rectangle indicates the true position of the global position of the Master D1.

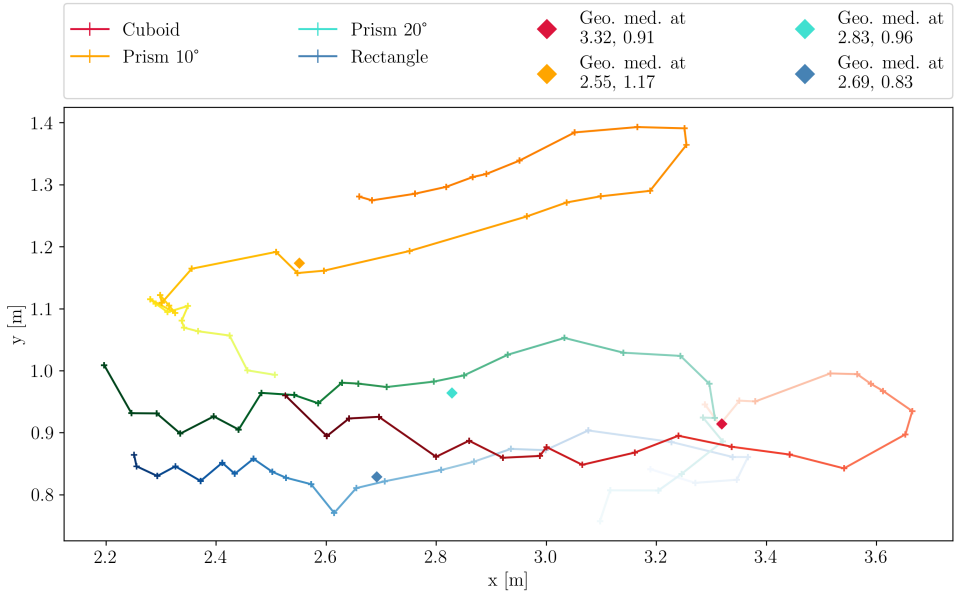


Figure 5.11.: Local acoustic position two-dimensional, with removed outlier at 1.55 m,1.6 m; true position of ROV estimated at 2.83 m,−0.45 m

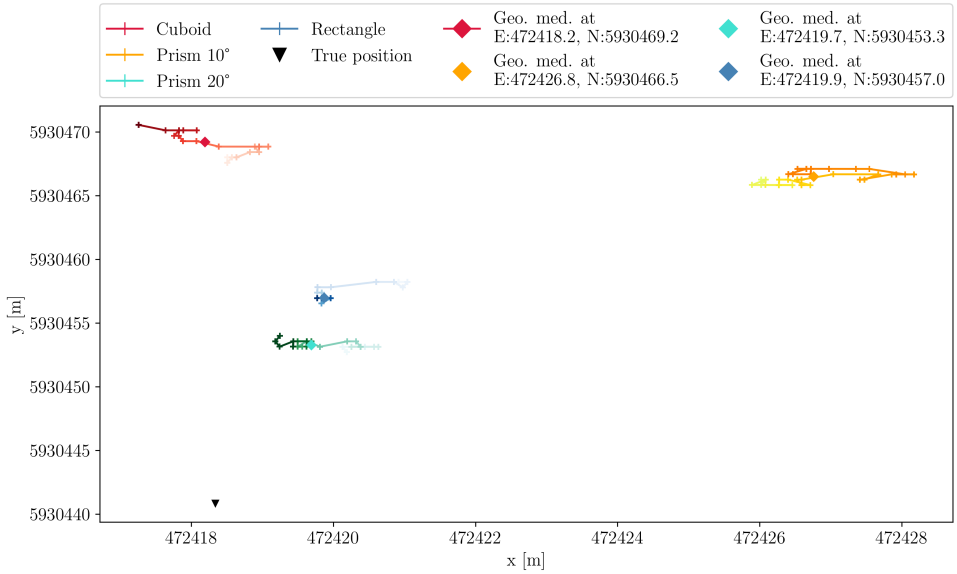
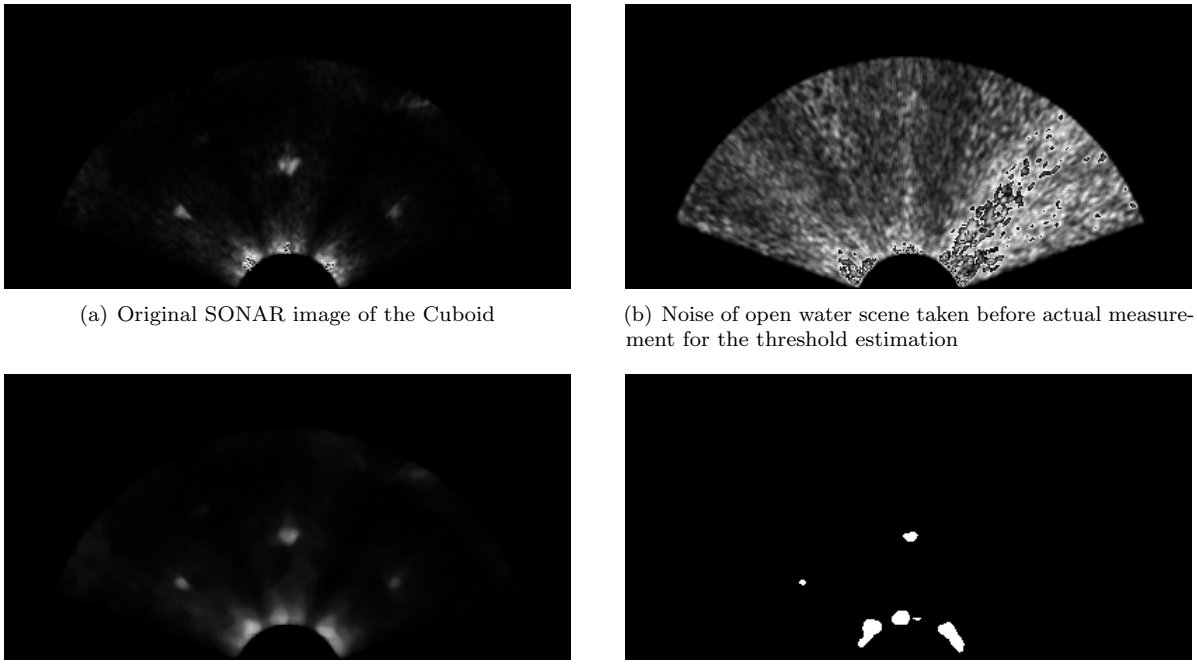


Figure 5.12.: Global acoustic position fitted; true position at 472 418.17 m, 5 930 440.84 m; excluding outlier at −505 645 m, 1.6 m

5.2.4. Sonar Images

In fig. 5.13 (a) the image of the Cuboid is shown as it is supplied by the BlueView P900-130 SONAR, representative for all other measurements. It must be noted that the figure only present parts of the Cuboid, in this case approximately the middle section. As can be seen in fig. 5.13 (b) the image noise



(a) Original SONAR image of the Cuboid

(b) Noise of open water scene taken before actual measurement for the threshold estimation

(c) Application of median 7x7 filter

(d) SONAR image after thresholding with t = 120

Figure 5.13.: SONAR image results of cuboid reference object

is respectively low compared to the example shown in section 4.4, but nevertheless produces constant background noise that distorts the backscattered signal, which can be noticed in the lower centre of fig. 5.13 (d) close to the source of the sound emission. The probability factor p, as well as the derived noise reduction thresholds are listed in table 5.1. It can be seen that the thresholds are fairly close with a mean

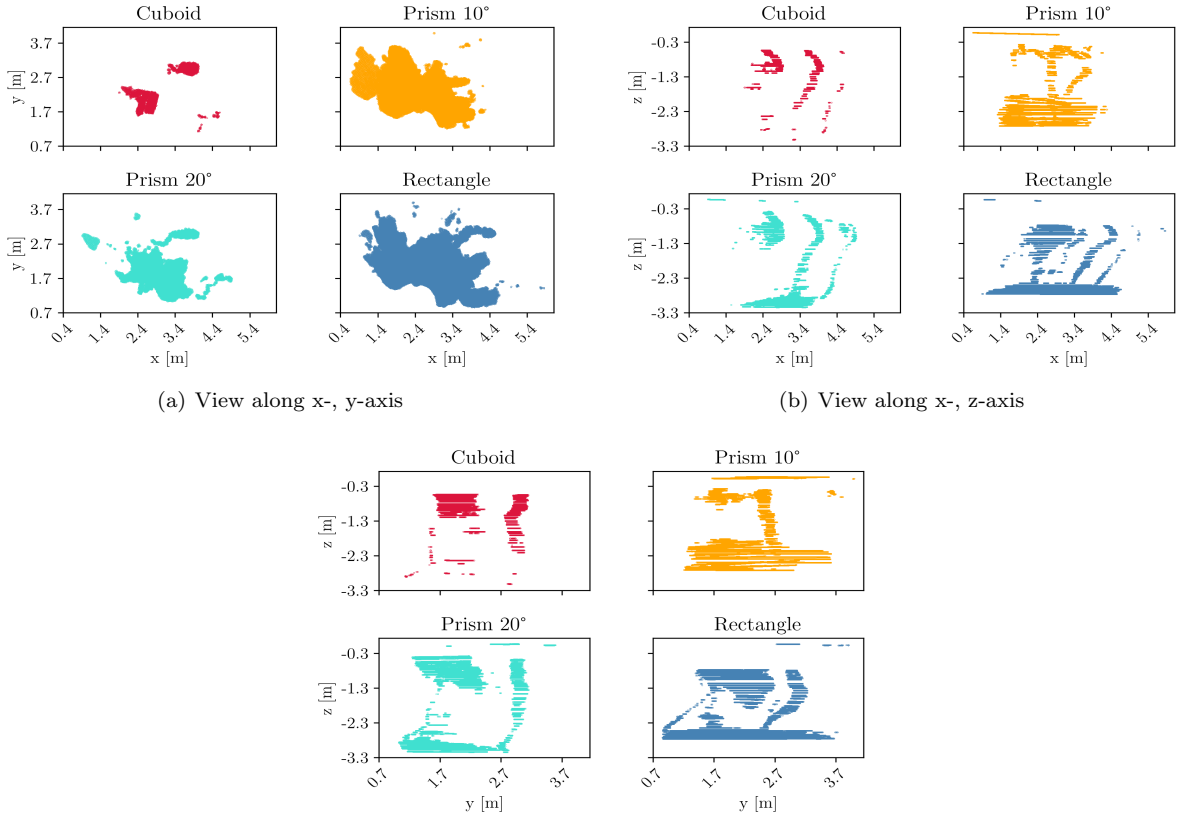
Table 5.1.: Thresholds applied to the four data sets

Reference object	Probability value p	Threshold t
Cuboid	0.472	120
Prism 10°	0.448	114
Prism 20°	0.476	121
Rectangle	0.436	111

value of 116.5 and a standard deviation of ±4.2.

5.2.5. Data Fusion and Spatial Representation

In this section the resulting point cloud representations of all four reference objects will be presented according to the two representation methods local and global, further divided into a depth representation at the expected depth of the test object. Furthermore all global outlier were disregarded for the latter. First of all, the position of the point cloud is shown in a simple two-dimensional plot, once from above (view on x-y) and then from the x-z- and the y-z-perspective. The point clouds are then presented in detail in the spatial representation by the NaviModel Free Viewer.
Figure 5.14 shows the individual local point clouds from the four measurements Cuboid, Prism 10°, Prism 20°and Rectangle, colour coded in x-y view (a), x-z view (b) and y-z view (c) in order to estimate the position and dimension of the point clouds presented in fig. 5.15. In fig. 5.14 (a) the Prism 10°, Prism 20°and Rectangle datasets show a large accumulation of points in the middle section of the pot between x:1 m to 4.4 m and y: 1.7 m to 3.7 m. In the Cuboid dataset a triangular shape can be identified in the lower left area and an oval shape in the top, middle section at (3.4 m, 2.8 m). In fig. 5.14 (b) a lateral deviation, as well as a doubled depth course is detected in the case of all datasets from 3.4 m to ca. 4.4 m. Further a triangular objects is detected in fig. 5.14 in the Prism 20°and Rectangle dataset. Figure 5.14 (c) also clearly shows a large object on the left hand side of all plots and a smaller one on the left hand side, over the full depth of 3.3 m. In fig. 5.15 an overview of the entire local scene is given in a three-dimensional



(a) View along x-, y-axis

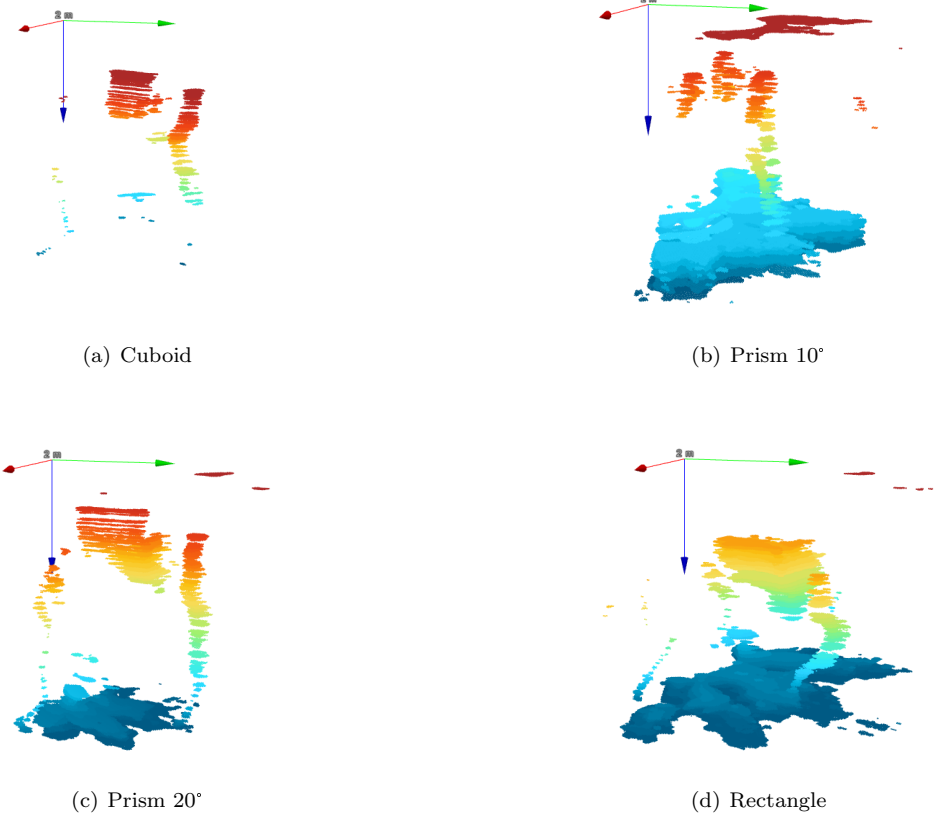
(b) View along x-, z-axis

(c) View along y-, z-axis

Figure 5.14.: Local two-dimensional representation of whole scene

representation using the software NaviModel Free Viewer. In the further representations fig. E.2 and fig. E.3 specific shots from the viewing direction in x-y- and y-z-direction are given. As these only provide a different way of presenting the data, already shown in fig. 5.14, they are listed under sec. E in the annex. The given coordinate system specifies the alignment towards y (green), x (red) and the z/depth (blue). Distances can be estimated using the specified length of the specific coordinate axis or can be compared

to the distances in figure 5.14
In fig. 5.15 the displayed scene shows a narrow oval shape ranging from top to bottom of the representation but also a outer edge of a rectangle/triangle (left in fig. 5.15 (a), (c) and (d)) and a radial point cloud (top right corner in fig. 5.15 (b) and at approx. $-4\,\mathrm{m}$ to $-5\,\mathrm{m}$ in fig. 5.15 (b), (c) and (d)). Furthermore small point cloud segments can be found in fig. 5.15 (a), (c) and (d) on the right side of the reference object.



(a) Cuboid

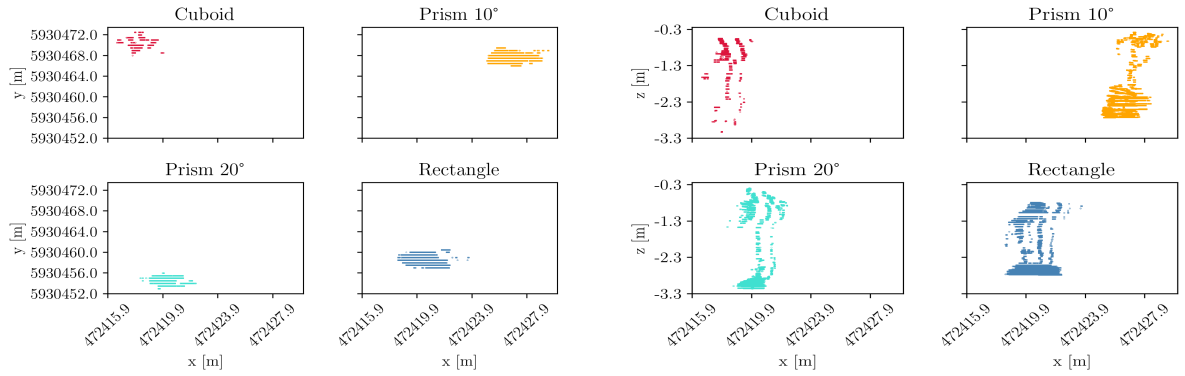
(b) Prism 10°

(c) Prism 20°

(d) Rectangle

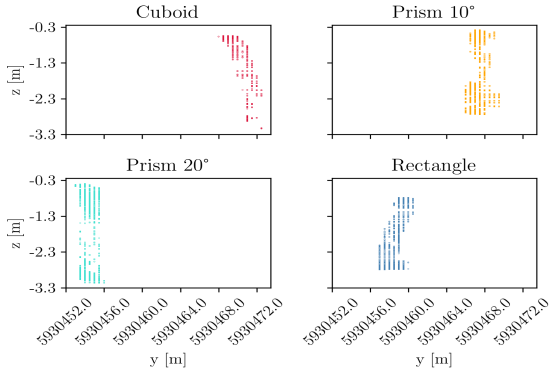
Figure 5.15.: Local three-dimensional representation of whole scene; length of coordinate axes 2 m

Figure 5.16 displays the same datasets in global UTM-coordinates, colour coded in x-y-view (a) and two plots along the z-axis (b) and (c). These plots also only allow an overview of the coordinate system, in which they are plotted, as well as their dimensions. An accumulation of points can be observed in 5.16 (a) in the range between x: 472 416.0 m to 472 428.5 m and y: 5 930 454.5 m to 5 930 474.5 m. Along the x-z- and y-z-axis the point cloud spans in respect of the depth from 0.5 m to 3.3 m. For a more detailed view,



(a) View along x-, y-axis

(b) View along x-, z-axis

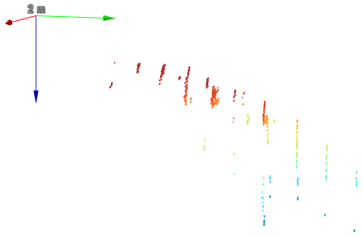


(c) View along y-, z-axis

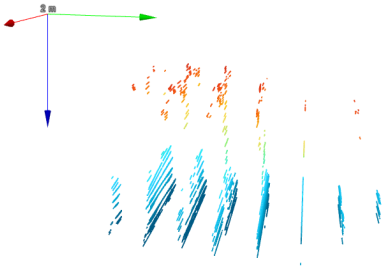
Figure 5.16.: Global two-dimensional representation of whole scene

images were also created with the NaviModel Free Viewer, see fig. 5.17 and also annex fig. E.4 and fig. E.5. Noticeable is the strong distortion and low point density of each point cloud in contrast to the local representation. Point segments are aligned horizontally, with a distance of ca. 0.5 m, vertically only a few centimeter. A slight suggestion of the objects from the local representation can be observed nevertheless. In fig. 5.17 especially the radial backscatter in (b) to (c) can be recognized (blue points at the bottom of all images as well as red points in (b) in the topmost layers). It has to mentioned that the other objects are plotted distorted along the y-axis. The horizontal distance between the almost slice shaped point accumulation is particularly clearly visible in fig. 5.17 (c) at the bottom of the image.

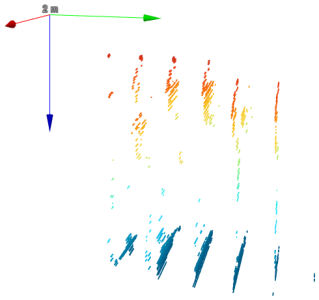
In figure 5.18 the point clouds, presented in fig. 5.15 and fig. 5.17, are only represented in the specific depth of the reference objects, which can be taken from table 2.1 or in the description below each individual image. The left figure column shows the local (s. fig. 5.18 (a),(c),(e) and (g)) the right column the global representation (fig. 5.18 (b),(d),(f) and (h)). Its has to be mentioned that the colour code has been adapted to fit the maximum depth in this presentation and therefore does not match with the depths assigned to a specific colour in the earlier presentations. All data sets, except the Prism 10°with a duration of 19.731 s, lie within the range 12.826 s to 13.987 s. The amount of points in each point cloud does not differ between the total global and local representation in all cases. Regarding the largest number of points (global/local) the Rectangle point cloud shows 1 327 977 global and 1 327 977 local data points with



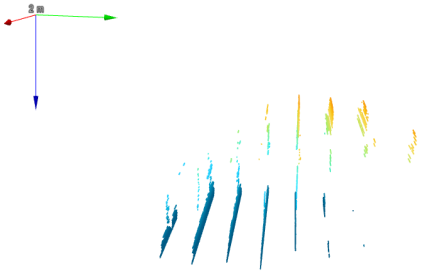
(a) Cuboid



(b) Prism 10°

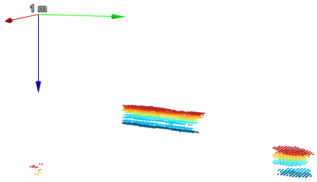


(c) Prism 20°

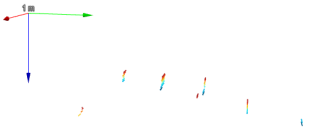


(d) Rectangle

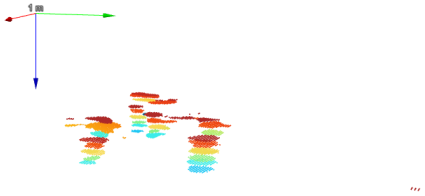
Figure 5.17.: Global three-dimensional representation of whole scene; length of coordinate axes 2 m



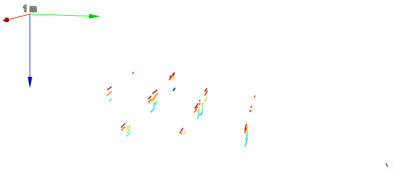
(a) Cuboid local, depth corrected on z between 0.555 m and 0.71 m



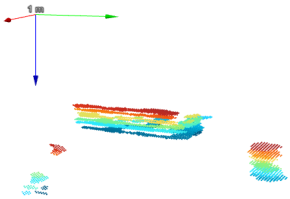
(b) Cuboid global, depth corrected on z between 0.555 m and 0.710 m



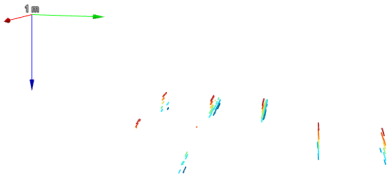
(c) Prism 10°local, depth corrected on z between 0.586 m and 0.820 m



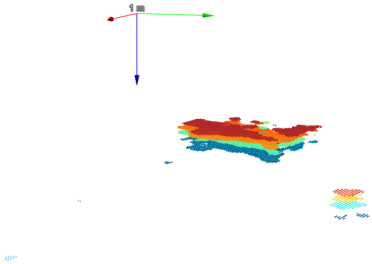
(d) Prism 10°global, depth corrected on z between 0.586 m and 0.820 m



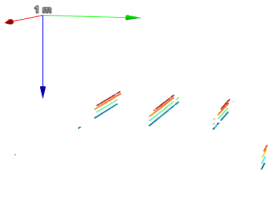
(e) Prism 20°local, depth corrected on z between 0.599 m and 0.710 m



(f) Prism 20°global, depth corrected on z between 0.599 m and 0.830 m



(g) Rectangle local, depth corrected on z between 0.695 m and 0.900 m



(h) Rectangle global, depth corrected on z between 0.695 m and 0.900 m

Figure 5.18.: Global three-dimensional representation of the point cloud

a recording time of 12.928 s, the smallest number of points is in the Cuboid representation (56 076 global and 56 076 local) with a recording time of 12.826 s. For the data set of the Prism 20°no pixel amount of the possible reference in the depth restricted representation could be determined, due to a high number unidentifiable pixel accumulation.

5.3. Evaluation of Measurement Results

In this section, individual aspects from the areas of software implementation, measurement setup and execution are now analysed with regard to their quality on the basis of the resulting measurement results. The measuring accuracy of the individual sensors and their interdependencies with each other as well as the interaction with environmental influences are discussed. First of all, it must be stated that the entire measurement setup is complex and the individual sensors and systems are highly dependent on each other. This results in a large accumulation of continuing errors, which can have a strong influence on the measurement results. It is beyond the scope of this study to examine which influence is the most dominant one and how this is to be reduced. Nevertheless, suggestions for further improvement are given in section 6.

5.3.1. Evaluation of Depth Measurement

A similar course of the calculated, acoustic depth and the depth from the pressure measurement can be observed, but the accuracy differences are clearly visible in the two data plots, compare fig. 5.5 and fig. 5.6. In view of this, the number of different depth values from the depth segmented representation (s. fig. 5.18) of the reference objects were compared for both depth measurements in table 5.2. It shows that in the case of the acoustic positioning system only one depth value was recorded for the span of the reference objects in the specific depth range (see annex D table D.2). In contrast, the pressure sensor Bar30 recorded an average of 7.5 different values in the same depth range. Although the frequency with which the UGPS generates depth data seems to be about 1.4 times the recording frequency of the pressure sensor, however, this cannot compensate its low accuracy. The specified measuring accuracy of 1 % per range form receiver to locator therefore cannot be confirmed in the case of the depth measurement regarding the UGPS. With the expectation of a depth value every 2 cm for the pressure sensor, the measuring accuracy, based on the average of the four measurements, can be confirmed. This supports the presumption that an external depth sensor should be used for depth measurement if the data is depth-dependent in order to achieve a higher accuracy. In the deep segmented representation (s. fig. 5.18), as in annex E table E.4, partly less depth layers are identified, which is due to the reduction only to the reference object. Thus the possibility is given that the missing depth layers are to be found in the overall representation, or that depths lying close together at a distance of a few millimetres have been combined into one layer by the representation software NaviModel Free Viewer. In the case of the Cuboid, the number of horizontal planes in table E.4 is smaller than the number of different depth measurements in table 5.2. In this case, it is assumed that a not sufficiently deducted pitch angle may have led to another plane in the point cloud representation. Based on the depth progressions in the case of Rectangle, it can be seen in fig. 5.8 to 5.10 that a slow descent and a gentle stop, leads to smaller deviations of roll-, pitch- and yaw-angle in comparison to those of Cuboid and Prism 20°with a fast descent and an abrupt deceleration at z_{max}, resulting in a strong angular change towards the end of the measurement.

5.3.2. Evaluation of Motion Parameters

In three out of four cases the datasets are a descent (s. fig. 5.5), therefore the thrusters are not actively used for the sinking of the ROV, but only for a slight stabilization by using the stabilization mode. During these measurements only the yaw-angle is manually adjusted, in order to capture the reference object from the front. In an ascent, such as the Prism 10°dataset, the thrusters are actively

Table 5.2.: Depth values in depth segmented point cloud

Reference object	Amount of depth values		Amount of differing values	
	UGPS	Bar30	UGPS	Bar30
Cuboid	30	24	1	4
Prism 10°	57	47	1	12
Prism 20°	36	40	1	6
Rectangle	57	47	1	8

used to elevate the ROV. In the course of all four measurements, the ROV drifts from its original orientation/yaw-angle. Due to the minimal readjustment of the orientation, the roll- and yaw-angle (lateral tilting to the right and forward) change in all four data sets. For example in the data set of the Rectangle there is a change of direction from 0 s to 5 s by about 4° (s. fig. 5.10), at the same time the roll-angle changes by 2° (s. fig. 5.8) and the pitch-angle by about 3° (s. fig. 5.9). In the case of the Prism 20°, even during a descent, there is a negative change in direction from approximately 0 s to 9 s by −3°, followed by a positive change in direction by 2° and then again negative to −12° (s. fig. 5.10). In the Prism 10°data set, the drift of the ROV from its original orientation shows a very irregular course in roll- and pitch-angle (s. fig. 5.8 and fig. 5.9) during the ascend. When sinking the roll- and pitch-angle remain almost unaffected by the relatively slow change of the yaw-angle of 4° in 7 s. It can be deduced that a descent allows a more stable positioning of the ROV, and that ascents should be avoided.

Roll- and pitch-angle react to changes in direction of the yaw-angle simultaneously with similar strength. This means that the ROV will roll- to the right and tilt forward at the same time, which indicates an uneven taring of the ROV. The weight of the SONAR is not compensated by additional buoyancy bodies in the rear of the ROV, nor is the lateral weight of the locator U1. This is not noticeable when the ROV is held in a static position, but as soon as it is in motion a movement around the two axes mentioned above can be detected. This can also be derived from annex E table E.1, where the highest standard deviation from the pitch-angles arithmetic average amounts 2.8° in the Cuboid and even 3.5° in the Prism 20°measurement. It can be concluded that as long as the yaw-angle varies only about 1.1°, it has no effect on the roll- and pitch-angle. For example, compare data set Cuboid, where both angles remain almost constant for 11 s until shortly before the end of the measurement. A significant change in angle from 0° to 5.8° only occurs with a yaw deviation of 4° within 0.5 s.

Even though the position parameters are deducted in the visualization program (s. sec. 4.5 eq. 4.8), the data set of the Prism 10°(s. fig. E.2) still shows very inclined horizontal point cloud segments (se. fig. E.2 (b)) in the top layers (equal to the beginning f the measurement). All other point clouds in fig. E.2 appear to align horizontally. This is considered as evidence that the correction of the sensor data by the rotation matrix is successful, even though the distance measurement from Pixhawk 1 to the SONAR was executed by hand and includes a deviation of ±1 cm.

5.3.3. Evaluation of Short Baseline Positioning Data

In the case of the acoustic positioning system various sources of error can be mentioned, for example, the manual measurement of the distances from the TC to the individual receivers. The measurement itself is done manually using a simple measuring stick, where the measurement accuracy per receiver is about 5 cm. Also the attachment of the receivers to bollards does not allow an absolutely correct positioning with regard to the x- and y- axis, due to cable bending. The receivers themselves were weighted down with lead weights, in order to allow a straight cable while being submerged into the water, but also here a slight cable bend could still be detected, so that the depth can also be estimated at 3 cm to 5 cm less, depending on the current strength and cable bend, which is influenced by the rolled up storage. However,

the factors just mentioned are considered to have a relatively small effect on the position in the context of using the positioning system together with another acoustic system.

Although the highest frequency channel was already used for the locator U1 during the measurement, a clear drift of up to 1.2 m in the x-direction can be seen in the position results in fig. 5.11, which is evident in all four measurements, albeit at different heights in the y-direction. In the case of the datasets Cuboid, Prism 20°and Rectangle, i.e. the sinking movements, a drift of the position from more positive to more negative x values can be detected. In the case of the Prism 10°measurement (first ascent, then descent) it can be speculated that the ascent leads to a drift from more negative to more positive values. In case of the descent the drift develops the other way round. In the y-axis of the Cuboid, Prism 20°and Rectangle datasets the ROV geometric medians are relatively close, with a deviation below 1.10, s. annex E table E.2..

Furthermore, the calculated, acoustic position deviates strongly from the true (but also only estimated) position. However, it can be seen that there is a clear difference between the estimated position of 2.83 m,−0.45 m with a generous deviation of 0.5 m meters in both directions, and the calculated geometric median of the measurements. The position estimation and the high deviation is due to the fact that the ROV was positioned under the pontoon without direct visual contact. It was only positioned on sight according to the backscattered SONAR image, which results in a relative position to the reference object. Furthermore, no accumulation of measuring points can be found in all four data sets, which would be an indicator for a static measurement. This is in contradiction to the measurement procedure where only small changes in movement in x- and y-direction were made by the ROV. Consequently, a correct calculation of the position of the ROV, and therefore consequently the SONAR image, is not provided. The specified measurement accuracy of 1 % per range from the receiver to locator is not given with the SONAR and the environmental conditions prevailing on site, such as strong reflectivity due to an angled and enclosed measurement area (harbour basin). With regard to the global positioning, it can be assumed that the outliers at 1.55 m,−505 645 m in figure 5.12 are probably due to an interrupted reception from the locator, as the same outlier appears in the local data set at x: 1.55 m. As the the number of points is only limited, but consistent in all four datasets (see annex E table E.2.) this abnormality will be neglected for the time being, as they are not considered in the final three-dimensional representation.

5.3.4. Evaluation of Image Processing

The SONAR data is collected using a standard value for the speed of sound of 1500 $\frac{\text{m}}{\text{s}}$, as no *Conductivity Temperature Depth - Logger* (CTD), in order to determine the salinity level, was available. As the theory in section 2 shows, lower sound velocities are expected in brackish water (including harbour water in fishing port 1) due to lower salinity than salt water and temperature stratification with depth. Consequently, it cannot be entirely ruled out that the objects were presented differently than with a correct sound velocity value.

The results of the threshold determination show that the mean value of the sonar images with object is only slightly larger than the mean value of the noise images. Consequently, this results in a low threshold (see table 5.1) and a rejection of the H_0 hypothesis. The expectation that the hypothesis will be rejected is therefore confirmed when using images with small reflective objects. Despite this fact, the threshold was applied to the images and shows a relatively good representation of the objects in consideration of the deviating positioning. The previous processing of the images with the 7x7 median filter leads to an amplification of the edge, as in the example of the Cuboid in fig. 5.13. However, it can also be stated that with an object width of 7.9 cm to 15.5 cm and a vertical beam angle of 20° the median filter leads to a "filling" of the objects. Due to the inaccurate position determination of the ROV, the SONAR images are plotted inaccurately with respect to the x-axis, which leads to a smearing of the object (oval shape) in the three-dimensional representation.

5.3.5. Evaluation of Spatial SONAR Representation

In the locale whole representations (see fig. 5.15) the edge of the described buoyancy body of the pontoon can be clearly identified. It is located in the figures 5.14 (a) on the left hand side, (b) top left and (c) top left, as well as in figure 5.15 (a), (c) and (d) on the left hand side of the picture. The object on the right (s. fig. 5.15) must be the reference object, which is already detected by the SONAR's 20° vertical beam angle at depths above and below the actual installation depth. In addition, noise can be identified directly in front of the SONAR, which indicates a high particle density in fig. 5.15 (b) and (d) bottom. In the example of fig. 5.15 (b) however, the noise at a lower depth is more likely to be due to the air bubble turbulence caused by the thrusters of the ROV as being too close to the surface. In the side view of the whole scene in E.2 it is again clear that the vertical length of the reference objects is not identical to the actual object size (compare table D.1 to the maximum depth of the point cloud in 5.14 (b)). The reason for this is again the vertical beam angle of the SONAR.

The measurement expectation of the alignment and position of the reference object is partially fulfilled (s. fig 5.14 (a)). Deviations of 0.5 m in x- and 1 m in y-direction from the to be expected position can be assessed as good preliminary result, given the error propagation due to multiple subsystems in the measurement setup. The lateral profile of the reference objects is distorted over the depth and cannot be clearly attributed to the geometry of the objects. Further the representation shows a detection up to a depth of 3.3 m, even though the depth of the Bar30 measurement (s. fig. 5.5) does not exceed below 1.4 m, no precise clue of the error can be stated at this point, but it is rather likely to be found in the coordinate transformation algorithm. This can be proved as all depths inserted in the calculation, and additionally saved in an external JSON-file, are above -1.4 m.
Although in the example of the two Prisms (s. fig. E.2 (b) and (c)) the elongated slope in the deeper part of the point cloud can be clearly seen, the similar course of the Rectangle in fig. E.2 (d) puts it into question. Even in the close-up of the depth-related representation (s. fig 5.18), no laterally clearly identifiable shape can be determined. Also in the top view the representations does not show any characteristic similarity with the reference objects see table E.4, as all objects seem to have a round to oval shape. Therefore no digital comparison of simulated and three-dimensional constructed objects (in fig. 5.2 (e) to (h)) can be made. In order to be able to determine the effect of the acoustic positioning on the three-dimensional representation, there are no corresponding comparative measurements at this point, where the ROV is fixed to a rod by means of a rope and moves up and down, so that it remains at the same position. Thus, the influence of the SONAR itself on the acoustic positioning and therefore the position of the point cloud, cannot be determined. It is not possible to say whether the lateral offsets in the frontal view (fig. 5.14 (b)) are due to the geometry of the object or whether they are additionally distorted by the acoustic positioning system.

With regard to the global display representation, similar outlines as in the local display can be seen (s. fig. 5.17). Otherwise, the data is clearly distorted with respect to the x-,y-axis representation. This is mainly due to the insufficient accuracy of the GPS-antenna, which, with an accuracy of ± 3 m in addition to the local data, significantly reduces the clarity. It can be seen, however, that both the local and the global point cloud contain the same number of points. This is true for both the total and the depth-segmentation for the entire point cloud, as well as for a section of the reference object in the depth-segmented point cloud, compare table E.3. The only difference is that they are plotted in the global variant as vertical lines along the y-axis (s. fig. 5.16 and fig. E.5), which is due to insufficient accuracy.

6. Conclusion and Further Work

In the presented thesis a methodology for the spatial representation and georeferencing of two-dimensional SONAR images taken from a underwater sensor platform with six degrees of freedom was proposed. Three subsystems, the ROV sensor platform, as the navigational unit underwater, the acoustic positioning system UGPS, to calculate the position of the sensor carrier and the SONAR itself for image generation, were operated simultaneously in compound. The data acquisition of the individual sensors and their conversion into a standardized and common data format and further processing were carried out. A scaling of the image size to the adjusted beam range as well as threshold filtering of background noise and edge enhancement by a median 7x7 filter was performed. According to the recording time of each SONAR image, corresponding data sets of the movement parameters of the ROV and acoustic position system were assigned and a corresponding movement compensation of the Euler angles, as well as an allocation of the global and the local position were carried out. Each referenced point was then graphically represented in a point cloud. The evaluation of the validity of the data fusion was carried out by means of a test setup consisting of four different reference objects, with a size greater than the measurement accuracy of the SONAR, attached to a plastic board and submerged in a harbour basin.
The resulting program network is designed to enable instructed persons to collect, process and display two-dimensional sonar data with minimal additional programming effort, such as value adjustment. Further, it is assumed that programs can be called up via the command line and in appropriate development environments by the user.

The measurement trial with the reference objects has shown that in particular the simultaneous use of SONAR and acoustic positioning system in an angled and enclosed measurement area can lead to partly strong deviations of the calculated position with a relative positioning to the reference point. With the integration of an external GPS-antenna, the general measurement deviation is added to the local position, which is further distorted by this. Corresponding investigations must be carried out in the future to improve the alignment of the point cloud according to the expected position. Furthermore, the aperture angle of the SONAR and an programming error in the depth allocation posed a problem for the detection of the correct length of the object in z-direction. Further, intensifying noise, when measuring too close to the water surface or at depths below 3 m, occurred possibly due to the depth hold and therefore intensive use of the thrusters by the ROV. Nevertheless, the reference object, even if distorted over the depth, and its immediate surroundings could be visualized three-dimensionally from only two-dimensional images.

For future measurements regarding inspection and maintenance of sheet pile walls, both the object distortion in z-direction and the positional deviation in x- and y-direction should be considered. At the current state of the measurement setup, it can be assumed that it has a similar accuracy as the regular plumb line measurement once a damage has been found. However, the system can help to analyse a large area in advance and minimize the inspection area. For diving inspection teams working with the presented system setup, the deviation in the x- and y-direction results in a larger area that needs to be surveyed, especially along the depth, and consequently a longer operation, than with an accurate positioning and smaller vertical beam angle of the SONAR. With regard to the inexpensive nature of the measurement setup, it can be said that an already used, but as good as new, SONAR was used. It is still possible that a corresponding new alternative or other model can change the total price. However, the overall setup is considerably less expensive than a three-dimensional SONAR. As a possible alternative to the ROV based sensor transport it can be considered whether the SONAR can be carried by a diver to further reduce the price. Accordingly, the SONAR should be equipped with a waterproof gyroscope, for motion compensation, a pressure sensor and the locator U1, as well as a data logger.

In continuing work it will be necessary to make improvements to the software and the measurement setup in order to increase the user-friendliness of hardware and software, the accuracy of the acquired data and the final point cloud display. Beginning with the software improvement it is recommend to create two intuitive GUI or batch files (containing all programs to be executed), in order to reduce the manually call up sequence for the data acquisition on the TC, as well as the processing and fusion of the data on the external computer. Further, in the context of risk management in port areas, an improvement of the vertical resolution of the point cloud needs to be examined. This could be either achieved by creating a machine learning algorithm for reducing the influence of the vertical beam angle by change-detection over time or as a mechanical alternative an implementation of a rotating SONAR mounting on the ROV. The SONAR could also be replaced by one with a smaller vertical aperture angle. Also the additional integration of a CTD logger, would be recommended in order to customize the sound velocity, needed for a correct data acquisition of the SONAR. Regarding the interferences between the SONAR and the acoustic positioning system, measurement trials in an absolute predictable environment, without any objects, need to be carried out, in order to examine if the floating position can be adjusted by a partial regression line or through a moving geometric median. A more advanced proposal would be a further integration of a DVL or INS in the sensor carrier. To enhance the global position it might be necessary to invest in a higher accuracy GPS-antenna, such as those commonly used in drone navigation. The final recommendations will have to be examined for profitability and affordability, depending on the application and desired accuracy, in order to stay in the economically priced range for the measurement setup.

Regarding the measurement setup and execution it is recommended to operate the system for the time being in low-flow areas until a sufficient positioning with a position hold mode for the ROV operation is possible. Further a more an expanded layout of the UGPS receivers with higher distance to harbour walls, as well as a further lowering of the latter should be tested in regard of enhancing the acoustic positioning.

Bibliography

Book Sources

[14] Roger Dashen, Walter H. Munk, and Kenneth M. Watson. *Sound transmission through a fluctuating ocean*. Cambridge University Press, 2010.

[18] H. G. Gierloff-Emden et al. *Landolt-Börnstein, Numerical data and functional relationships in science and technology*. Ed. by J. Sündermann. Vol. Volume 3: Oceanography. Springer-Verlag, Berlin Heidelberg New York, 1986.

[23] Caruthers Jerald W. *Fundamentals of Marine Acoustics*. Elseviewer Scientific Publishing Company, 1977.

[25] R. Köster et al. *Ökosystem Wattenmeer/The Wadden Sea Ecosystem: Austausch-, Transport-und Stoffumwandlungsprozesse/Exchange Transport and Transformation Processes*. Springer-Verlag, 2013.

[30] J. H. Mathews and K. D. Fink. *Numerical Methods Using Matlab*. Prentice-Hall, 2004, pp. 430–436. ISBN: 0-13-065248-2.

[39] Springer Reference. *Encyclopedia of Robotics*. Ed. by Marcelo H. Ang, Oussama Khatib, and Bruno Sicilliano. Springer Berlin Heidelberg, 2018. ISBN: 978-3-64-241610-1.

Manuals

[2] ArduPilot. *MAVProxy 1.6. A UAV ground station software package for MAVLink based systems*. 2019. URL: `https://ardupilot.org/mavproxy/index.html` (visited on 07/01/2020).

[10] BlueView Technologies. *P-Series User Handbook*. 2012. URL: `http://dascoei.ca/wp-content/uploads/2013/09/Data_Sheet_P900_Series_v2.pdf` (visited on 06/12/2020).

[11] BlueView Technologies. *P900 Series. 2D Imaging Sonar*. BlueView Technologies, Inc. 2151 N. Northlake Way, Suite 214 Seattle, WA 98103 USA, 2005.

[12] BlueView Technologies. *ProViewer Sonar Development Kit. User's Guide*. 2012.

[15] Digilent®. *PmodGPS™Reference Manual*. 2016.

[22] InvenSense. *MPU-6000 and MPU-6050 Product Specification 3.4*. Aug. 19, 2013. URL: `https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf` (visited on 05/24/2020).

[31] MAVLink. *MAVLINK Common Message Set*. URL: `https://mavlink.io/en/messages/common.html` (visited on 06/13/2020).

[41] STMicroelectronics. *Digital gyroscopes for enhanced motion-control realism and image stabilization. ST's new digital gyroscopes are the perfect synthesis of accuracy and design flexibility*. 2015. URL: `https://www.st.com/content/ccc/resource/sales_and_marketing/promotional_material/flyer/be/1f/c9/09/c2/4b/45/b0/fl3axdigitalgyro.pdf/files/fl3axdigitalgyro.pdf/jcr:content/translations/en.fl3axdigitalgyro.pdf` (visited on 06/24/2020).

[42] STMicroelectronics. *LSM303DLM. Sensor module: 3-axis accelerometer and 3-axis magntometer*. 2011. URL: `https://www.st.com/content/st_com/en/products/mems-and-sensors/e-compasses/lsm303dlm.html` (visited on 06/28/2020).

[43] TEConnectivity. *MS5611-01BA03 Barometric Pressure Sensor, with stainless steel cap.* URL: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS5611-01BA03%7FB3%7Fpdf%7FEnglish%7FENG_DS_MS5611-01BA03_B3.pdf%7FCAT-BLPS0036 (visited on 06/28/2020).

[47] Water Linked. *Water Linked NMEA Underwater GPS bridge.* 2020. URL: https://github.com/waterlinked/ugps-nmea-go (visited on 06/05/2020).

[50] WaterLinked. *WL-21005. Rceiver-D1.* 2017.

Papers, Articles, Technical Reports and Inproceedings

[13] Christophe Croux, Peter Filzmoser, and Heinrich Fritz. "A comparison of algorithms for the multivariate L1-median". In: *CentER Discussion Paper Series* (2010).

[19] Thomas Guerneve, Kartic Subr, and Yvan Petillot. "Three-dimensional reconstruction of underwater objects using wide-aperture imaging SONAR". Englisch. In: *Journal of Field Robotics* 35.6 (2018), pp. 890–905.

[20] Dipl. Geol. Anne Heeling. "Ermittlung und Bewertung des Korrosionszustandes von Stahlspundwänden in Häfen und an Wasserstraßen - Determination and Assessment of the Corrosion State of Sheet Pile Walls in Harbours and Waterways". In: *Kompetenz für die Wasserstraßen—Heute und in Zukunft. Forschungs-und Entwicklungsprojekte der BAW* 100 (2017), pp. 39–53. URL: https://izw.baw.de/publikationen/mitteilungsblaetter/0/mb_100_04_Heeling_Ermittlung.pdf.

[21] Sherwood B Idso and R Gene Gilbert. "On the universality of the Poole and Atkins Secchi disk-light extinction equation". In: *Journal of Applied Ecology* (1974), pp. 399–401.

[24] Wolf-Michael Kähler. "Prüfung von Zentren (z-Test, t-Test)". In: *Statistische Datenanalyse*. Springer, 2004, pp. 237–278.

[27] Richard B. Langley. "The UTM grid system". In: *GPS world* 9.2 (1998), pp. 46–50.

[28] J. Levin and E. McNichol. "Color Vision in Fish". In: vol. 246. Scientific American, 1982, pp. 108–117.

[32] Thomas G. Muir and D. L. Bradley. "Underwater acoustics: A Brief Historical Overview Through World War II". In: *Acoustics Today* 12.3 (2016).

[34] Y. Petillot et al. "Underwater vehicle path planning using a multi-beam forward looking sonar". In: *IEEE Oceanic Engineering Society. OCEANS'98. Conference Proceedings (Cat. No. 98CH36259)*. Vol. 2. IEEE. 1998, pp. 1194–1199.

[45] Keith Vickery. "Acoustic positioning systems. A practical overview of current systems". In: *Proceedings of the 1998 Workshop on Autonomous Underwater Vehicles (Cat. No. 98CH36290)*. IEEE. 1998.

Online Sources

[1] Arduino. *Arduino Nano.* 2020. URL: https://store.arduino.cc/arduino-nano (visited on 06/05/2020).

[3] Blue Robotics. *Bar30 High-Resolution 300m Depth/Pressure Sensor.* 2020. URL: https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar30-sensor-r1/ (visited on 06/17/2020).

[4] Blue Robotics. *BlueROV2 Heavy Configuration Retrofit Kit.* 2020. URL: https://bluerobotics.com/store/rov/bluerov2-upgrade-kits/brov2-heavy-retrofit-r1-rp/ (visited on 06/28/2020).

[5] Blue Robotics. *Payload Skid.* 2020. URL: https://bluerobotics.com/store/rov/bluerov2-accessories/brov-payload-skid/ (visited on 06/28/2020).

[6] BlueRobotics. *BlueROV2*. 2020. URL: https://bluerobotics.com/store/rov/bluerov2/ (visited on 06/12/2020).

[7] BlueRobotics. *MAVLink Proxy and Comman Line Ground Station*. GitHub Open Source Code Sharing Platform. URL: https://github.com/bluerobotics/MAVProxy (visited on 06/13/2020).

[8] BlueRobotics. *Raspberry Pi 3 Model B*. 2020. URL: https://bluerobotics.com/store/comm-control-power/elec-packages/rpi3-r1/ (visited on 06/28/2020).

[9] BlueRobotics. *Single Fathom-X Tether Interface Board*. URL: https://bluerobotics.com/store/comm-control-power/tether-interface/fathom-x-tether-interface-board/ (visited on 06/07/2020).

[16] EIVA. *NaviModel Free Viewer*. 2020. URL: https://www.eiva.com/products/navisuite/navisuite-processing-software/navimodel-producer (visited on 06/05/2020).

[17] Federal Office of Topography swisstopo. *Universal Transverse Mercator (UTM)*. URL: https://www.swisstopo.admin.ch/en/knowledge-facts/surveying-geodesy/reference-systems/map-projections.html (visited on 06/05/2020).

[26] Visual Computing Lab. *MeshLab*. 2020. URL: http://www.meshlab.net/ (visited on 06/05/2020).

[29] Fredrik Lundh. *Pillow*. 2020. URL: https://python-pillow.org/ (visited on 07/01/2020).

[33] Open3D. *Open3D. A Modern Library for 3D Data Processing*. 2020. URL: http://www.open3d.org/ (visited on 07/01/2020).

[35] PX4. *Pixhawk 1 Flight Controller*. Ed. by Dev Team. 2019. URL: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html (visited on 06/28/2020).

[36] Python Software Foundation. *json - JSON encoder and decoder. JSON encoder and decoder*. 2020. URL: https://docs.python.org/3/library/json.html (visited on 06/15/2020).

[37] Python Software Foundation. *socket. Low-level networking interface*. 2020. URL: https://docs.python.org/3/library/socket.html (visited on 06/14/2020).

[38] QGroundControl. *Intuitive and Powerful Ground Control Station for the MAVLink protocol*. 2020. URL: http://qgroundcontrol.com/ (visited on 06/23/2020).

[40] seatronics. *Teledyne Blueview P900 Series Sonar*. 2020. URL: https://seatronics-group.com/rov-sensors/teledyne-blueview-p900-series-sonar/ (visited on 06/21/2020).

[44] Teledyne Marine. *ProViewer™. -BlueView-*. URL: http://www.teledynemarine.com/ProViewer (visited on 06/07/2020).

[46] Water Linked. *getposition.py*. Ed. by jaxxzer and wlkh. Sept. 25, 2018. URL: https://github.com/waterlinked/examples/blob/master/getposition.py (visited on 06/21/2020).

[48] WaterLinked. *Locator U1*. URL: https://waterlinked.com/product/locator-u1/ (visited on 06/12/2020).

[49] WaterLinked. *Underwater GPS*. URL: https://waterlinked.com/underwater-gps/ (visited on 06/10/2020).

Annex

A. Laws and Principles regarding Light

Range of vision in port water with optical systems
For the determination of the visual range in turbid, sediment/DOM rich waters, a Secchi-disk, which is a round, white coated disc, to which a rope is attached in the middle for easy lowering into the water, is often used in practice. The depth from which the disk cannot be seen any more is defined as the Secchi-depth z_{Secchi}. For calculating the attenuation length, that means the distance at which the light should be attenuated to the value $\frac{1}{e}$, which typically corresponds to the definition of "visibility", the reciprocal of the attenuation coefficient needs to be determined.
The attenuation coefficient from the Secchi-Depth z_{Secchi} is defined by A.1.[21]

$$c = \frac{1.7}{z_{Secchi}} \tag{A.1}$$

In general, Lambert-Beer's law applies to the ratio of emitted intensity I_0 to transmitted intensity I_1 at distance d with the attenuation coefficient c.

$$\frac{I_1}{I_0} = e^{-c \cdot d} \tag{A.2}$$

The definition of the visual range, with $\frac{1}{e} = \ln(1)$, then is valid for $c \cdot d = 1$. This then results in eq. A.3.

$$d\left(\frac{I_1}{I_0} = \frac{1}{e}\right) = \frac{1}{c} \tag{A.3}$$

During a measurement trial at the DLR Institute for the Protection of Maritime Infrastructure from the 15th of October 2019 to the 17th January 2020 performed by the sensor technology group in the Fischereihafen 1, an average value of 0.92 m for the visibility and 2.05 for the attenuation coefficient were determined. It can be concluded that optical measurements will probably be of low quality and rather have to be recorded for distances of less than 0.92 m to show the necessary detail. It should also be noted that the measurements were strongly dependent on the weather and the activities in the harbour basin.

B. Technical Specifications

Table B.1 lists the costs of the individual system, as described in section 3, for the measurement setup, presented in section 5.1.

Table B.1.: Costs of the measurement setup individual systems in euro (dollar to euro exchange rate on 06/28/20)

System	Specification	Price	Resource
ROV BlueROV2, Blue Robotics	Heavy Configuration Retrofit Kit with Payload Skid, 4 Lights, Water Linked UGPS / BlueROV2 Integration Kit, additional 2 Fathom-X boards, 100 m tether cable	9588.12 €	[9], [6], [4], [5]
BlueView P-Series SONAR, Teledyne Marine	P900-130	24 000 €	reconditioned loan by external company
UGPS, Water Linked AS	U1 Locator, 4 Receiver each 10 m	4902.81 €	[48] [49]
Total		38 739.64 €	

Table B.2 lists the specific sensors integrated on the sensor carrier as shown in sec. 3.1 fig. 3.1 and their corresponding company, version and reference.

Table B.2.: Sensors on the BlueROV2 sensor carrier

Sensor	Company	Version	Resource
Depth sensor	BlueRobotics	Bar30	[3]
Temperature sensor	BlueRobotics	Bar30	[3]
Flight Controller Unit	Holybro	Pixhawk 1	[35]
Multi-beam SONAR	Teledyne Marine	BlueView P900-130	[11]
Acoustic locator U1	Water Linked AS	U1	[48]

Table B.3 lists the integrated sensors on the Pixhawk 1 Flight Controller Unit, as implemented in the BlueROV2, see sec. 3.1.

Table B.3.: Sensors on the Flight Controller Unit Pixhawk 1

Sensor	Company	Version	Resource
Gyroscope	ST Microelectronics	L3GD20H	[41]
	Invensense	MPU 6000	[22]
Accelerometer	ST Microelectronics	LSM303D	[42]
	Invensense	MPU 6000	[22]
Magnetometer	ST Microelectronics	LSM303D	[42]
Barometer	TE Connectivity	MEAS MS5611	[43]

Table B.4 gives an overview of the to be expected accuracy of the individual sensors integrated in the measurement setup.

Table B.4.: Accuracy of BlueROV2 Sensors

Sensor	Accuracy
Bar30 MS5037-30BA Pressure sensor	±0.2 cm
Bar30 Temperature sensor	±1 °C
L3GD20H Gyroscope	±2000 dps
LSM303D Accelerometer	±16 g
LSM303D Magnetometer	±2 gauss
MPU 6000 Accelerometerr	±16 g
MPU 6000 Gyroscope	±2000 dps
MEAS MS5611 Barometer	±1.5 mbar
PmosGPS™	±3 m
WaterLinked UGPS	±1 %*range receiver to locator

Table B.5 shows the explicit sensor specification of the BlueView P900-130 forward-looking Multibeam SONAR used in the implementation of the thesis.

Table B.5.: System Specifications of the BlueView P900-130

Specification	Value
Field-of-View	130°
Max. Range	100 m
Beam Width	1° x 20°
Beam Spacing	0.18°
Number of Beams	768
Range Resolution	2.54 cm
Frequency	900 kHz

Table B.6 shows the frequency channels of the underwater acoustic positioning system UGPS of the company WaterLinked. In the thesis implementation channel 7 was used.

Table B.6.: Frequency band and according channel from Water Linked acoustic products

Frequency band		Channel
from	to	
31.25 kHz	65.25 kHz	1
62.25 kHz	93.75 kHz	2
93.75 kHz	125 kHz	3
125 kHz	156 kHz	4
156 kHz	187 kHz	5
187 kHz	218 kHz	6
218 kHz	250 kHz	7

C. Listings

In this section all listings used during the course of the thesis will be illustrated. Those with an other source than the examinee will be marked. Moreover the corresponding files, sorted by occurrence in the annex, can be found on the storage medium in the folder *Listings*, which is enclosed with the thesis.

```
'''
Example Module by Peter Barker, September 2016
from https://github.com/ArduPilot/MAVProxy/blob/master/MAVProxy/modules/
  mavproxy_example.py
AND
https://github.com/bluerobotics/MAVProxy/blob/master/MAVProxy/modules/
  mavproxy_DepthOutput.py
adapted by Lea Meyer in order to access:
- heading
- depth
- scaled pressure of Bar30
- Raw IMU (not used)
- Euler angles (roll, pitch and yaw)

added lines are marked with a star * in the comment, main code was adopted from
  sources above
'''
import os
import os.path
import sys
from pymavlink import mavutil
```

```
import errno
import time
import socket
import json

from MAVProxy.modules.lib import mp_module
from MAVProxy.modules.lib import mp_util
from MAVProxy.modules.lib import mp_settings


class AllData(mp_module.MPModule):

  BUFFER_SIZE = 65536

def __init__(self, mpstate):
  """Initialise module"""
  super(AllData, self).__init__(mpstate, "AllData", "All Data support")
  #self.add_command('AllData.port', self.cmd_port, 'Port selection', ['¡25102¿'])
  #self.add_command('AllData.depthSource', self.cmd_depth_source, 'Depth source selection', ['¡bar30—filtered¿'])
  #self.add_command('AllData.tempSource', self.cmd_temp_source, 'Temperature source selection', ['¡sp2—sp3¿'])

  self.last_update = 0
  self.depth_source = 'bar30'
  self.temp_source = 'sp2'

  self.data = {
  'depth' : 0, # Depth in meters
  'temp' : 0,    # Water temperature in degrees Celsius
  'orientation' : 0
  }

  self.ip="127.0.0.1"
  self.portnum = 25102
  self.port = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  self.port.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  mavutil.set_close_on_exec(self.port.fileno())
  print "Outputting data on UDP://%s:%s" % (self.ip, self.portnum)

  #self.status_callcount = 0
  #self.boredom_interval = 10 seconds
  #self.last_bored = time.time()

  #self.packets_mytarget = 0
  #self.packets_othertarget = 0

  #self.example_settings = mp_settings.MPSettings(
  # [ ('verbose', bool, False),
  # ])
  #self.add_command('example', self.cmd_example, "example module", ['status','set (LOGSETTING)'])

'Handle mavlink packets, get data'
def mavlink_packet(self, m):
  '''handle mavlink packets'''
  #print (str(type(m)) + ": " + m.get_type())

  if m.get_type() == 'SCALED_PRESSURE2':
    if self.temp_source == 'sp2':
      self.data['temp'] = m.temperature / 100.0 # m.temperature is centi-C
    if self.depth_source == 'bar30':
      self.data['depth'] = self.get_depth(m.press_diff * 100) # press_diff is hPa
  elif m.get_type() == 'SCALED_PRESSURE3':
    if self.temp_source == 'sp3':
      self.data['temp'] = m.temperature / 100.0 # m.temperature is centi-C
  elif m.get_type() == 'GLOBAL_POSITION_INT':
    if self.depth_source == 'filtered':
      self.data['depth'] = -m.relative_alt / 1000.0 # m.relative alt is mm
  elif m.get_type() == 'VFR_HUD':
    self.data['orientation'] = m.heading
```

```
    # added *
    elif m.get_type() == 'ATTITUDE':
      self.data['roll'] = m.roll
      self.data['pitch'] = m.pitch
      self.data['yaw'] = m.yaw

  def send_data(self):
    if time.time() < self.last_update + 0.25:
    return
    self.last_update = time.time()
    datagram = json.dumps(self.data)
    self.port.sendto(datagram, (self.ip, self.portnum))

  def idle_task(self):
    '''called in idle time'''
    self.send_data()

  def cmd_port(self, args):
    'handle port selection'
    if len(args) != 1:
      print("Usage: port <number>")
    return
    self.port.close()
    self.portnum = int(args[0])
    self.port = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.port.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    mavutil.set_close_on_exec(self.port.fileno())
    print "Outputting depth on UDP://%s:%s" % (self.ip, self.portnum)

  def cmd_temp_source(self, args):
    'handle temperature source selection'
    if len(args) != 1:
      print("Usage: tempSource <source>")
    return
    source = args[0]
    if source == 'sp2':
      self.temp_source = source
    elif source == 'sp3':
      self.temp_source = source
    else:
      print ('Unknown temperature source %s' % source)

  def cmd_depth_source(self, args):
    'handle depth source selection'
    if len(args) != 1:
      print("Usage: depthSource <source>")
    return
    source = args[0]
    if source == 'bar30':
      self.depth_source = source
      elif source == 'filtered':
      self.depth_source = source
    else:
      print ('Unknown depth source %s' % source)

    def get_depth(self, pressure_pa):
    specific_gravity = self.get_mav_param("GND_SPEC_GRAV", 1.0)
    return pressure_pa / 9800.0 / specific_gravity

  def init(mpstate):
    '''initialise module'''
    return AllData(mpstate)
```

Listing C.1: MAVProxy module required for data aquisition from sensors of the Pixhawk 1 on the ROV

```
import socket
from time import gmtime, strftime
import time
```

```
import json
import sys
import re #

UDP_IP = "127.0.0.1"
UDP_PORT = 25102

# socket from ROV
sock = socket.socket(socket.AF_INET, # Internet
socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

# socket to Docker
rov_sock = socket.socket(socket.AF_INET, # Internet
socket.SOCK_STREAM) # TCP
#sock.bind((UPD_IP, 5001))

if len(sys.argv) > 1:
  filename = sys.argv[1]
  path = "ROVMotion_Data/"
  print filename
  filename = path + sys.argv[1]
else:
  filename= 'workfile'


f = open(filename, 'w')
while True:
  data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
  print "received message:", data
  print time.time() #time.localtime()
  dataJson = json.dumps(data, separators=(',',':'))
  dataJson = dataJson[:-2]
  dataJson = dataJson[1:]
  dataJson+=", \"time\": " + str(time.time()) + "}\n"
  dataJson1 = dataJson.replace('\\', '')
  f.write(dataJson1)
else:
  print "error"
```

Listing C.2: UDP access of motion parameters from IP address of ROV

```
"""
Programm fetches position of underwater vehicle from url http://192.168.2.94
and saves data in file
"""
#modules
from __future__ import print_function
import requests
import argparse
import time
import json

def get_data(url):
try:
  r = requests.get(url)
  except requests.exceptions.RequestException as exc:
  print("Exception occured {}".format(exc))
  return None

if r.status_code != requests.codes.ok:
  print("Got error {}: {}".format(r.status_code, r.text))
  return None

return r.json()

def get_acoustic_position(base_url):
  return get_data("{}/api/v1/position/acoustic/filtered".format(base_url))
```

```
  def get_global_position(base_url):
    return get_data("{}/api/v1/position/global".format(base_url))

  #check corect filename input for string, float, and file format
    def set_file_name():
    file_format = ".json"       #desired file format
    path = "UGPS_Data/"
    userinput = input("1. Please enter file name for position data (without .json): ")
    filename = path + userinput + file_format
    return filename

  def main():
    filename = set_file_name()
    received_data = True

    parser = argparse.ArgumentParser(description=__doc__)
    parser.add_argument('-u', '--url', help='Base URL to use', type=str, default='http
    ://192.168.2.94')
    args = parser.parse_args()

    base_url = args.url
    print("Using base_url: %s" % args.url)

    #data aquisistion and saving data in json file
    f = open(filename, 'w')
    data = {}                   #creating dictionary
    try:
      while received_data == True:
      data.update({'time' : time.time()})
      data.update(get_acoustic_position(base_url))
      data.update(get_global_position(base_url))
      dataJson = json.dumps(data, separators=(',',':'))
      f.write(str(dataJson))
      f.write("\n")
    except KeyboardInterrupt:
      print("Recording ended!")
      pass
    f.close()

  if __name__ == "__main__":
  main()
```

Listing C.3: IP access of UGPS positioning data

```
'''
Author: Malte Struck
Institute: DLR-MI
Program is executed on Arduino Nano 3.0 and provides transfer
of serial GPS NMEA string by UART-USB-Bridge
'''
SoftwareSerial mySerial(2, 3); // RX, TX

void setup() {
  Serial.begin(9600);
    while (!Serial) {
    }

  mySerial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (mySerial.available())
    Serial.write(mySerial.read());
}
```

Listing C.4: Executed skript on Arduino Nano for accessing GPS-antenna data

```
/*
* SONAR .SON to .PGM Conversion
* Opening .SON Files, retrieve ping and convert image
* Storage of the file under path with filename consiting of img
* timestamp and .pgm ending
*/
#ifdef _WIN32
# define _CRT_SECURE_NO_WARNINGS
#endif
#include <cstdio>
#include <cstring>
#include <cmath>
#include <iostream>
#include <bvt_sdk.h>
#include <io.h>
#include <string.h>

char DataFile[] ="";

int main( int argc, char *argv[] )
{
  if (argc >= 2){
  strcpy(DataFile, argv[1]);
  }
  int filenamelength = strlen(DataFile);
  char foldername[900];
  strncpy(foldername, DataFile, filenamelength - 4);
  printf("%s\n", foldername);
  if (mkdir(foldername) != 0)
    {
      printf("New folder could not be created\n");
    }

  printf("Datafile: %s\n", DataFile);
  // Create a new BVTSonar Object
  BVTSonar son = BVTSonar_Create();
  if( son == NULL )
    {
      printf("BVTSonar_Create: failed\n");
      return 1;
    }

  // Open the sonar file
  BVTSonar_Open(son, "FILE", DataFile);

  // Get the first head
  BVTHead head = NULL;
  BVTSonar_GetHead(son, 0, &head);

  // Check the ping count
  int pings = -1;
  pings = BVTHead_GetPingCount(head);

  // Begin loop to create image for each ping
  for (int i = 0; i< pings; i++){
    // Now, get a ping!
    BVTPing ping = NULL;
    BVTHead_GetPing(head, i, &ping);

    //Get Time stamp for ping
    double time = BVTPing_GetTimestamp(ping);
    double s;
    int n;
    s = time;
    s = floor(s);
    double ms = time - s;
    char buf3[100];
    n = sprintf(buf3, "%d%d", (int)s, (int)(ms * 1000));
```

```
        if (n < 13)
70      {
        if (n = 12)
72        {
            sprintf(buf3, "%d0%d", (int)s, (int)(ms * 1000));
74        }
        else
76        sprintf(buf3, "%d00%d", (int)s, (int)(ms * 1000));

78      }

80      const char* timestampchar = buf3;


82
        // Generate an image from the ping
84      BVTMagImage img;

86      BVTPing_GetImageXY(ping, &img);

88      //Save it to a PGM (PortableGreyMap)

90      char result1[256];
        const char *filename1 = "img";
92      const char *format1 = ".pgm";

94      strcpy(result1,foldername);
        strcat(result1, "\\");
96      strcat(result1,filename1); // copy string filename1 into the result.
        strcat(result1,timestampchar);
98      strcat(result1,format1);// append string format1 to the result.
        //printf("result1:BVTMagImage_SavePGM(img, result1);
100
        // progress indicator
102     std::cout << "\rProgress: " << i << " out of " << pings << " images converted!";

104     //* Clean up
        BVTMagImage_Destroy(img);
106     BVTPing_Destroy(ping);
    }
108
    BVTSonar_Destroy(son);
110 }
```

Listing C.5: .SON to PGM program based on BlueVIEW SDK

```
    #modules
2   import numpy as np
    from PIL import Image
4   from math import log, ceil
    from scipy import stats
6   from os import listdir
    import pathlib
8   import matplotlib.pyplot as plt

10  def constantsGlobVars():
      #constants
12    globalMaxMultiplicator = 0.25;

14    #global vars
      globalMaxN = 0
16    globalMaxS = 0
      return globalMaxN, globalMaxS, globalMaxMultiplicator
18
    def parseLine(a): # todo: give nbytes as parameter!
20    local_max = 0
      length = len(a) / 2
22    length = int(length)
      value = np.zeros(length)
```

```
    for i in range(0, length):
      j = 2*i
      value[i] = a[j]*256 + a[j+1]
      if value[i] > local_max:
      local_max = value[i]
    return value, local_max

  def getNoisedata():
    globalMax, notused, globalMaxMultiplicator = constantsGlobVars()
    noiseMean = []
    noiseSD = []
    imgnum = 0   # initialize image number for creation of dummy data
    # process all images in working directory
    for f in listdir(pathlib.Path().absolute()):
      if f.endswith('.pgm') and f.startswith('noise'):   # only process .pgm files
        imgnum += 1   # assign image number for creation of dummy data
        file = open(f, 'rb')
        fields = file.read().split()[:4]
        b = bytearray()

        # find first binary data (after the 4th space)
        startOfBinData = 0
        maxStartOfBinData = 100
        spacesBeforeBinData = 4
        foundSpaces = 0

        #fileContent = file.read()

        for pos in range(0, maxStartOfBinData-1):
          file.seek(pos)
          byte = file.read(1)
          if ord(byte) == 0x20 and foundSpaces < spacesBeforeBinData:
            foundSpaces = foundSpaces + 1
            if foundSpaces == spacesBeforeBinData:
              startOfBinData = pos + 1

        #print("binary data starts at positon " + str (startOfBinData))
        # startOfBinData = 18

        file.seek(startOfBinData)          #go to position 18 in file were image data information starts

        # read the binary data into unsigned 8-bit array b
        b = np.fromfile(file, dtype='uint8')


        width = int(fields[1])
        height = int(fields[2])
        bytesPerValue = int(ceil(log(int(fields[3]), 256)))
        nslices = width * bytesPerValue

        b = np.array(b).reshape(height, nslices)

        #find global maximum pixel value of image
        imageData = []
        for i in range (0,len(b)):
        x, maxi = parseLine(b[i])
        if globalMax < maxi:
        globalMax = maxi
        imageData.append(x)

        globalMax = int(globalMax * globalMaxMultiplicator)

        data = np.zeros((height, width, 3), dtype=np.uint8)
        for i in range (0, len(imageData)):
        for j in range (0, len(imageData[i])):
        data[i, j]=(imageData[i][j] / (globalMax / 256), imageData[i][j] / (globalMax /
  256), imageData[i][j] / (globalMax / 256))
        img = Image.fromarray(data, 'RGB')
```

```
92          #get single channel of nosie image
            r, g, b = img.split()
94          noiseImagedata = b.histogram()
            newNoiseImage = zeroElimination(noiseImagedata)
96          NoiseMean = getMean(newNoiseImage)
            noiseMean.append(NoiseMean)
98          NoiseSD = getNoiseSD(newNoiseImage, NoiseMean)
            noiseSD.append(NoiseSD)
100     return noiseMean, noiseSD, imgnum, newNoiseImage

102   def getSampledata():
        notused, globalMax, globalMaxMultiplicator = constantsGlobVars()
104     sampleMean = []
        sampleSD = []
106     imgnum = 0   # initialize image number for creation of dummy data
        # process all images in working directory
108     for f in listdir(pathlib.Path().absolute()):
          if f.endswith('.pgm') and f.startswith('img'):   # only process .pgm files
110       imgnum += 1   # assign image number for creation of dummy data
          file = open(f, 'rb')
112       fields = file.read().split()[:4]
          b = bytearray()
114
          # find first binary data (after the 4th space)
116       startOfBinData = 0
          maxStartOfBinData = 100
118       spacesBeforeBinData = 4
          foundSpaces = 0
120
          for pos in range(0, maxStartOfBinData-1):
122         file.seek(pos)
            byte = file.read(1)
124         if ord(byte) == 0x20 and foundSpaces < spacesBeforeBinData:
            foundSpaces = foundSpaces + 1
126         if foundSpaces == spacesBeforeBinData:
              startOfBinData = pos + 1
128

130       file.seek(startOfBinData)           #go to position 18 in file were image data information starts

132       # read the binary data into unsigned 8-bit array b
          b = np.fromfile(file, dtype='uint8')
134


136       width = int(fields[1])
          height = int(fields[2])
138       bytesPerValue = int(ceil(log(int(fields[3]), 256)))
          nslices = width * bytesPerValue
140
          b = np.array(b).reshape(height, nslices)
142
          #find global maximum pixel value of image
144       imageData = []
          for i in range (0,len(b)):
146         x, maxi = parseLine(b[i])
            if globalMax < maxi:
148           globalMax = maxi
            imageData.append(x)
150
          globalMax = int(globalMax * globalMaxMultiplicator)
152
          data = np.zeros((height, width, 3), dtype=np.uint8)
154       for i in range (0, len(imageData)):
            for j in range (0, len(imageData[i])):
156           data[i, j]=(imageData[i][j] / (globalMax / 256), imageData[i][j] / (globalMax
      / 256), imageData[i][j] / (globalMax / 256))
          img = Image.fromarray(data, 'RGB')
158       #get single channel of noise image
          r, g, b = img.split()
```

```
160         sampleImagedata = b.histogram()
            newSampleImage = zeroElimination(sampleImagedata)
162         SampleMean = getMean(newSampleImage)
            sampleMean.append(SampleMean)
164         SampleSD = getNoiseSD(newSampleImage, SampleMean)
            sampleSD.append(SampleSD)
166     return sampleMean, sampleSD, imgnum, newSampleImage

168     def zeroElimination(sample):
          count = 0
170       data = []
          for i in sample:
172         if(count > 0):
              data.append(i)
174           count = count + 1
            else:
176           count = count + 1
        return data
178
        def getMean(sample):
180       sampleSize = len(sample)
          sumTotal = sum(sample)
182       mean = sumTotal/sampleSize
        return mean
184
        def getNoiseSD(sample, mean):
186       sampleSize = 255
          mean = mean
188       sumOfSquares = 0
          for element in sample:
190         deviationScore = element - mean
            sumOfSquares = sumOfSquares + deviationScore**2
192       variance = sumOfSquares/sampleSize
          SD = variance**0.5
194     return SD

196     def getSampleSD(sample):
          sampleSize = len(sample)
198       mean = getMean(sample)
          sumOfSquares = 0
200       for element in sample:
            deviationScore = element - mean
202         sumOfSquares = sumOfSquares + deviationScore**2
          variance = sumOfSquares/(sampleSize-1)
204       SD = variance**0.5
        return SD
206
        def getNoiseStandardError(NoiseAverageSD):
208       sampleSize = 255
          sampleSD = NoiseAverageSD
210       #Standard error of the mean for the pixel value
          SEM = sampleSD/(sampleSize**0.5)
212     return SEM

214     def getSampleStandardError(sample):
          sampleSize = len(sample)
216       noiseSD = getSampleSD(sample)
          #Standard error of the mean for the pixel value
218       SEM = noiseSD/(sampleSize**0.5)
        return SEM
220
        # Z = (sampleMean - noiseMean / StandardError)
222     def getZScore(sample, SampleAverageMean, NoiseAverageMean, NoiseAverageSD):
          sampleMean = SampleAverageMean
224       noiseMean = NoiseAverageMean
          SEM = getSampleStandardError(sample)
226       Z = (sampleMean - noiseMean) / SEM
        return Z
228
```

```
def getProbabilityFromZ(Z):
  Z = round(float(Z), 2)
  probability = 0
    if Z < -4:
      return 0.000
    elif Z > 4:
      return 1.000
    else:
      probability = stats.norm.cdf(Z)
return probability

def testHypothesis(p, alpha = 0.05, testType = 'one-tailed positive'):
  if testType == 'two-tailed':
    if p < (alpha/2) or p > (1-(alpha/2)):
      return True
    else:
      return False
    if testType == 'one-tailed negative':
      if p < alpha:
        return True
      else:
        return False
    if testType == 'one-tailed positive':
      if p > (1 - alpha):
        return True
      else:
        return False

def getThreshold(p):
  p = p * 100
  threshold = int(round((255/100)*p))
return threshold

def main():
  #get gloabl variables and constants
  globalMaxN, globalMaxS, globalMaxMultiplicator = constantsGlobVars()

  NoiseMean, NoiseSD, imgnum, newNoiseImage = getNoisedata()
  SampleMean, SampleSD, imgnumsample, newSampleImage = getSampledata()

  NoiseAverageMean = sum(NoiseMean)/imgnum
  NoiseAverageSD = sum(NoiseSD)/imgnum
  SampleAverageMean = sum(SampleMean)/imgnumsample

  # Now the actual Z-Test
  zScore = getZScore(newSampleImage, SampleAverageMean, NoiseAverageMean,
  NoiseAverageSD)
  p = getProbabilityFromZ(zScore)
  if testHypothesis(zScore):
    print('Z-TEST')
    SampleMean = getMean(newSampleImage)
    print('NoiseAverageMean', NoiseAverageMean)
    print('Sample Mean: ', SampleAverageMean)

    print('Z:' , zScore)
    print('p: ', p)
    threshold = getThreshold(p)
    print('Threshold', threshold)
    testhyp=testHypothesis(p)
    print("The hypothesis is: ", testhyp)
  else:
    print("error")
if __name__ == '__main__':
main()
```

Listing C.6: Gaussian Z-Test applied to sample images by means of noise images

```
  import numpy as np
  from matplotlib import image as mpimg
  import matplotlib.pyplot as plt
  from os import listdir
  import pathlib
  from matplotlib.pyplot import figure
  figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')

  def getMean(sample):
    sampleSize = len(sample)
    sumTotal = sum(sample)
    mean = sumTotal/sampleSize
  return mean

  def main():
    # process all images in working directory
    window = []
    for f in listdir(pathlib.Path().absolute()):
    if f.endswith('.pgm'):  # only process .pgm files
    print("filename: ", f)
    # open image
    img = np.array(mpimg.imread(f))
    img = img[:, :, 0]  #Pseudocolor
    # create points from nonzero pixels
    img_xy = np.nonzero(img)          # returns xy-vector of every nonzero pixel, not pixel value itself
    real_sonwindow = img.shape[0] - img_xy[0][0]
    print('real_sonwindow', real_sonwindow)
    window.append(real_sonwindow)

    mean = getMean(window)
    print("mean", mean)

if __name__ == '__main__':
main()
```

Listing C.7: Capture of the furthest pixel in the sonar image for the pixel to meter conversion in C.7

```
  # imports
  import numpy as np
  import math
  from PIL import Image
  from PIL import ImageFilter
  from os import listdir
  import pathlib
  import open3d as o3d
  import json
  from pyproj import Proj
  from math import log, ceil

  # function definitions
  def plot_pc(pcvector):
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(np.transpose(pcvector))
    #pcd.colors = o3d.utility.Vector3dVector(intensity)
    o3d.io.write_point_cloud("pointcloud.ply", pcd)

    #comment when .ply cropped; point cloud visualization with o3d
    pcd_load = o3d.io.read_point_cloud("pointcloud.ply")
    vishandle = o3d.visualization.draw_geometries([pcd_load])

    #un-comment when ply cropped
    # ============================================
    # vol = o3d.visualization.read_selection_polygon_volume("croppedpointcloud.json")
    # object = vol.crop_point_cloud(pcd)
    # o3d.io.write_point_cloud("pointcloudcropped.ply", object)
    # vishandle = o3d.visualization.draw_geometries([object])
    # ============================================
    #print("")
```

```
32
34  def get_rov_data(timestamp): # rov_data = [time, depth, orientation, roll, pitch, yaw]
      try:
36      with open('motion08_06_20.json') as json_file:
        for line in json_file:
38
          # read each line as json data
40        json_data = json.loads(line)

42        # create numpy array out of json data to allow faster processing //time in milli seconds
          newrow = np.array([[(json_data["time"]*1000), json_data["depth"], json_data["
    orientation"], json_data["roll"], json_data["pitch"], json_data["yaw"]],])
44
        # if this is the first line, create array 'data' and add first row
46        if 'data' not in locals():
          data = newrow
48      # if data already exists add new row on bottom of array
        else:
50        data = np.append(data, newrow, axis=0)
      except FileNotFoundError:
52      print('depth_log.json file could not be read or does not exist')
        return 0
54    # find index of time closest to given timestamp
      index = np.argmin(np.abs(np.add(data[:, 0], -timestamp)))
56    ROV_return = data[index, :]
      return ROV_return
58

60  def read_ugps_data():
      try:
62      with open('ugps08_06_20.json') as json_file:
          for line in json_file:
64          # read each line as json data
            json_data = json.loads(line)
66
            # create numpy array out of json data to allow faster processing
68          np.set_printoptions(suppress=True, formatter={'float_kind':'{:f}'.format})
            newrow = np.array([[json_data["time"]*1000, json_data["x"], json_data["y"],
    json_data["z"], json_data["lon"], json_data["lat"]],])
70          # if this is the first line, create array 'data' and add first row
            if 'data' not in locals():
72          data = newrow
          # if data already exists add new row on bottom of array
74        else:
            data = np.append(data, newrow, axis=0)
76        return data
      except FileNotFoundError:
78      print('ugps_data.json file could not be read or does not exist')
        return []
80

82  def get_ugps_coordinates(ugps_data, timestamp):       # ugps_data = [time, x, y, z, lon, lat]
      # find index of time closest to given timestamp
84    index = np.argmin(np.abs(np.add(ugps_data[:, 0], -timestamp)))
      #local test
86    local_x, local_y, local_z = ugps_data[index, 1], ugps_data[index, 2], ugps_data[
      index, 3]
      # projection from lon/lat to xy (UTM)
88
      # create projection object with settings for Bremerhaven
90    myProj = Proj(proj='utm', zone=32, ellps='WGS84', preserve_units=False) #"preserve_units
      = False" ensures units = meters
92    # apply projection global;
      #utm_x, utm_y = myProj(ugps_data[index, 4], ugps_data[index, 5], inverse=False)
94
      #projection local; note:comment global projection
96    utm_x, utm_y = ugps_data[index, 1], ugps_data[index, 2]
```

```
98      #without utm-projection
        #utm_x, utm_y = ugps_data[index, 4], ugps_data[index, 5]
100
        return utm_x, utm_y, local_x, local_y, local_z
102

104   def coordinate_transform(pc_raw, roll, pitch, yaw, ugps_x, ugps_y, depth):
        # transform coordinates
106     yawMatrix = np.array([
        [math.cos(yaw), -math.sin(yaw), 0],
108     [math.sin(yaw), math.cos(yaw), 0],
        [0, 0, 1]
110     ])

112     pitchMatrix = np.array([
        [math.cos(pitch), 0, math.sin(pitch)],
114     [0, 1, 0],
        [-math.sin(pitch), 0, math.cos(pitch)]
116     ])

118     rollMatrix = np.array([
        [1, 0, 0],
120     [0, math.cos(roll), -math.sin(roll)],
        [0, math.sin(roll), math.cos(roll)]
122     ])

124     # rotation matrix
        R = np.matmul(np.matmul(yawMatrix, pitchMatrix), rollMatrix)
126
        # measured impementation distance ugps to multibeam
128     ugps_x = ugps_x + 0.18
        ugps_y = ugps_y + 0.27
130     depth = depth + 0.25 #1. distance ugps locator to multibeam :- 0.35 or 2.pixhawk to multibeam: -0.25
        # transformation matrix
132     T = np.zeros((4, 4))
        T[:-1, :-1] = R  # rotation
134     T[:-1, 3] = [ugps_x, ugps_y, depth]
        T[-1, -1] = 1  # for right shape
136
        pc_img = np.matmul(T, pc_raw)[:-1, :]
138     return pc_img

140   def parseLine(a):
        local_max = 0
142     length = len(a) / 2
        length = int(length)
144     value = np.zeros(length)
        for i in range(0, length):
146       j = 2*i
          value[i] = a[j]*256 + a[j+1]
148       if value[i] > local_max:
            local_max = value[i]
150     return value, local_max

152   def main():
        # try to read UGPS data
154     ugps_data = read_ugps_data()  # returns array of array [time, x, y, z, lon, lat]
        # assumption: xyz relative to gps receiver, lon & lat absolute
156     imgnum = 0  # initialize image number for creation of dummy data
        # process all images in working directory
158     for f in listdir(pathlib.Path().absolute()):
          if f.endswith('.pgm'):  # only process .pgm files
160         imgnum += 1  # assign image number for creation of dummy data
            timestamp = float(f[3:-4])  # get timestamp from filename
162
            #constants
164         globalMaxMultiplicator = 0.25;
```

```
        #global vars
        globalMax = 0

        file = open(f, 'rb')
        fields = file.read().split()[:4]
        b = bytearray()

        # find first binary data (after the 4th space)
        startOfBinData = 0
        maxStartOfBinData = 100
        spacesBeforeBinData = 4
        foundSpaces = 0

        for pos in range(0, maxStartOfBinData-1):
          file.seek(pos)
          byte = file.read(1)
          if ord(byte) == 0x20 and foundSpaces < spacesBeforeBinData:
            foundSpaces = foundSpaces + 1
            if foundSpaces == spacesBeforeBinData:
              startOfBinData = pos + 1


        file.seek(startOfBinData)          #go to position in file were image data information starts


      # read the binary data into unsigned 8-bit array b
      b = np.fromfile(file, dtype='uint8')

      #access with and height, adjust image width
      width = int(fields[1])
      height = int(fields[2])
      bytesPerValue = int(ceil(log(int(fields[3]), 256)))
      nslices = width * bytesPerValue

      #reshape b into np array with adjusted height and width
      b = np.array(b).reshape(height, nslices)

      #get image data
      imageData = []
      for i in range (0,len(b)):
        x, maxi = parseLine(b[i])
        if globalMax < maxi:
          globalMax = maxi
        imageData.append(x)

      globalMax = int(globalMax * globalMaxMultiplicator)
      data = np.zeros((height, width, 3), dtype=np.uint8)

      for i in range (0, len(imageData)):
        for j in range (0, len(imageData[i])):
          data[i, j]=(imageData[i][j] / (globalMax / 256), imageData[i][j] / (globalMax
    / 256), imageData[i][j] / (globalMax / 256))
      img1 = Image.fromarray(np.rot90(data,3), 'RGB')

      #median 7x7
      img = img1.filter(ImageFilter.MedianFilter(7))

      #threshold
      threshold = 119 #get value from gauss threshold estimation
      im = img.point(lambda p: p > threshold and 255)
      img_xy = np.nonzero(im)

      #insert calculation for range
      real_sonwindow = 166 #see getrangepixelnumber.py: value generated by taking the min value of image rows
    with non-zero value subtracted from total image rows

      # move zero to bottom center of image
      img_y = img_xy[1]
      img_x = np.add(img_xy[0], - img.size[0] )
```

```
        # transform pixel to meter
        max_range = 3   # get value from multibeam!
        scaling = max_range / real_sonwindow
        img_x = np.multiply(img_x, scaling)
        img_y = np.multiply(img_y, scaling)

        # add third dimension
        ROV_data = get_rov_data(timestamp)  # function needs timestamp from filename

        ROV_z = ROV_data[1]

        img_z = np.full(len(img_x), ROV_z)  #create empty z-column of fitting length
        ones = np.full(len(img_x), 1)  # create row of ones for transformation
        pc_raw = np.array([img_x, img_y, img_z, ones])  # ready to transform vector

        roll = ROV_data[3] # in radian
        pitch = ROV_data[4] # in radian
        yaw = ROV_data[5] # in radian

        ugps_x, ugps_y, local_x, local_y, local_z = get_ugps_coordinates(ugps_data,
    timestamp)

        #Export .json files for all used timestamps
        #motion data
        export_ugps = {}
        export_ugps['time'] = timestamp
        export_ugps['x'] = local_x
        export_ugps['y'] = local_y
        export_ugps['z'] = local_z
        export_ugps['lat'] = ugps_x
        export_ugps['lon'] = ugps_y
        with open('ugpsminimized.json', 'ab+') as f:
            f.seek(0,2)                                 #Go to the end of file
            if f.tell() == 0 :                          #Check if file is empty
                f.write(json.dumps([export_ugps]).encode())  #If empty, write an array
            else :
                f.seek(-1,2)
                f.truncate()                            #Remove the last character, open the array
                f.write(' , '.encode())                 #Write the separator
                f.write(json.dumps(export_ugps).encode())    #Dump the dictionary
                f.write(']'.encode())                   #Close the array
        #UGPS data
        export_rov = {}
        export_rov['time'] = timestamp
        export_rov['roll'] = roll
        export_rov['pitch'] = pitch
        export_rov['yaw'] = yaw
        export_rov['z'] = ROV_z
        with open('motionminimized.json', 'ab+') as f:
        f.seek(0,2)                                     #Go to the end of file
            if f.tell() == 0 :                         #Check if file is empty
                f.write(json.dumps([export_rov]).encode())  #If empty, write an array
            else :
                f.seek(-1,2)
                f.truncate()                            #Remove the last character, open the array
                f.write(' , '.encode())                 #Write the separator
                f.write(json.dumps(export_rov).encode())    #Dump the dictionary
                f.write(']'.encode())                   #Close the array

        #local point cloud
        pc_img = coordinate_transform(pc_raw, roll, pitch, yaw, ugps_x, ugps_y, ROV_z)
        # for now we assume water surface == sea level and depth measured by pressure sensors

        # if pcvector does not yet exist create it
        if 'pcvector' not in vars():
            pcvector = [pc_img[0], pc_img[1], pc_img[2]]

        # if pcvector already exists append values to it
```

```
        else:
302         pcvector = np.append(pcvector, pc_img, axis=1)

304     # plot point cloud
        print(str(pcvector.shape[1]) + ' points detected')
306     plot_pc(pcvector)

308 if __name__ == '__main__':
    main()
```

Listing C.8: Main Programm for the data allocation accordind to each SONAR timestamp, coordinate transformation and storage in .ply format

D. Reference Object Specification

In this section specification regarding the used reference objects during the measurement trial in the Fischereihafen 1 will be listed. In table D.1 shows the exact dimensions of the poured concrete reference objects, measured with a measuring scale with an accuracy of 0.3 cm.

Table D.1.: Characteristics of the concrete reference objects

Reference object	Height [mm]		Width [mm]	Depth [mm]	Slant angle [°]
Cuboid	155	-	155	155	-
Prism 10°	89	41	80	234	10
Prism 20°	121	41	79	231	20
Rectangle	61	-	205	10	-

Table D.2 shows the exact depth range to witch the reference objects were submerged during the measurement trial.

Table D.2.: Depth range of reference objects

Reference object	Depth	
	min. [mm]	max. [mm]
Cuboid	555	710
Prism 10°	586	820
Prism 20°	599	830
Rectangle	695	900

E. Measurement Results

Before continuing with the measurement results, it has to be mentioned that further representation of the reference object datasets can be found on the storage medium, which is enclosed with the thesis. Also all *.xyz*-files are attached and can be viewed following the instructions in (*Note_How_to_Open_XYZ_Files_Using_NaviModel_Free_Viewer.txt* in the folder "Resulting Point Clouds"). Table E.1 shows the arithmetic average (mean) and the standard deviation of the Euler angles from the four data sets Cuboid, Prism 10°, Prism 20°and Rectangle.

Table E.1.: Arithmetic average and standard deviation of roll-, pitch- and yaw-angle

	Roll-angle [°]		Pitch-angle [°]		Yaw-angle [°]	
Reference object	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
Cuboid	0.612052	1.51403	0.985412	2.805061	-13.002563	0.753549
Prism 10°	-0.494998	1.0269	-0.913342	1.309763	-15.046159	1.648642
Prism 20°	0.465079	1.465079	-0.516253	3.464943	-12.146470	0.758398
Rectangle	0.679259	0.679259	0.594466	1.160268	-20.712225	1.086034

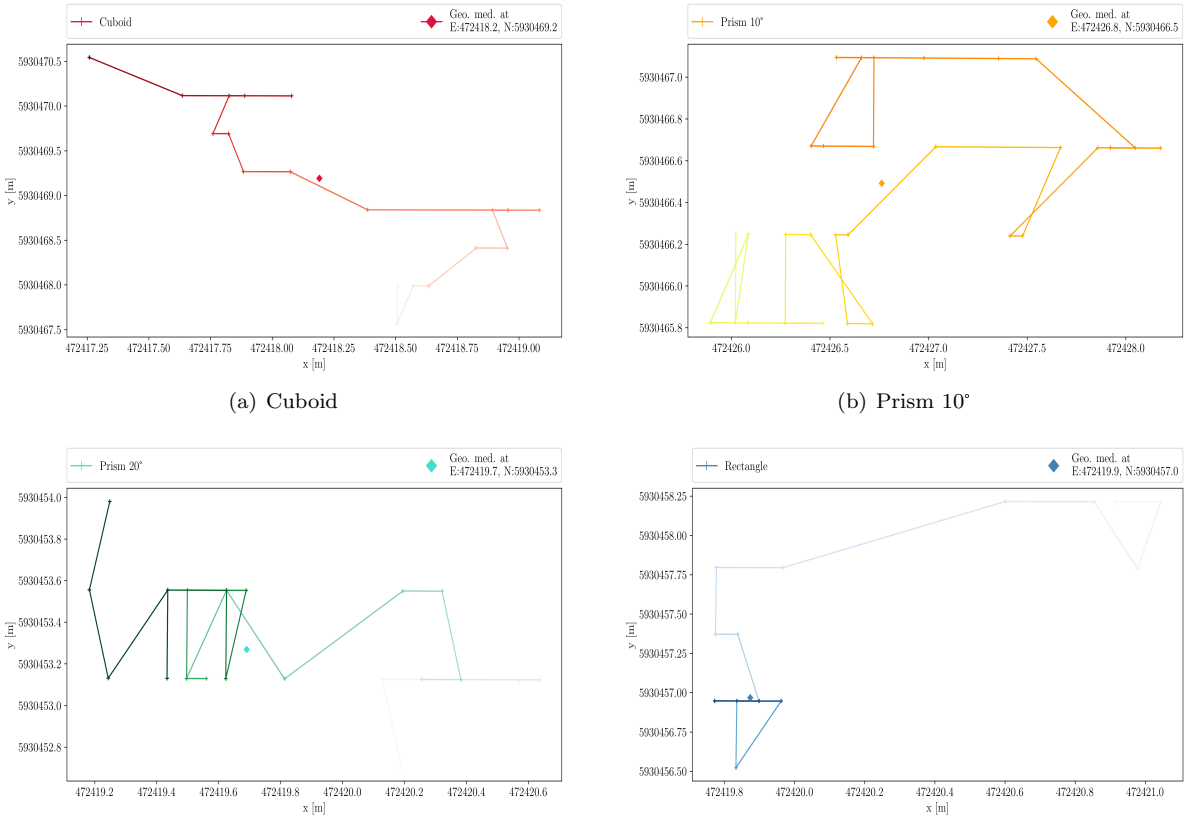
Table E.2 shows the geometric median and the MAD of each dataset.

Table E.2.: MAD regarding the geometric median m of the local position in x-y-direction and global regarding UTM x-y-direction

Reference object	x [m]		y [m]		Amount of outlier
	Geo. med. m	MAD	Geo. med. m	MAD	
Cuboid	3.32	1.12	0.91	1.06	1
Prism 10°	2.55	1.14	1.17	1.10	8
Prism 20°	2.83	1.15	0.96	1.09	2
Rectangle	2.69	1.14	0.83	1.06	2
Mean	2.85	1.14	0.97	1.08	

Reference object	UTM x [m]		UTM y [m]		Amount of outlier
	Geo. med. m	MAD	Geo. med. m	MAD	
Cuboid	472418.19	1.00	5930469.19	2.52	1
Prism 10°	472426.76	1.00	5930466.49	1.60	8
Prism 20°	472419.69	1.00	5930453.27	3.49	2
Rectangle	472419.87	1.14	5930456.97	1.06	2
Mean	472421.13	1.04	5930461.48	2.03	

Figure E.1 shows a close up representation of the acoustic global position of the ROV, during the four different measurement trials, additionally to the overview given in section 5.2 figure 5.12. The colour fading indicates the passing time, light colours representing the beginning of the measurement and darker/intensive colours indicating the end of the measurement. Additionally to the drift along the x-axis, strong position jumping can be observed in the Prism 10°and 20°as well as in the Rectangle dataset. An deviation around a static point cannot be observed.



(a) Cuboid

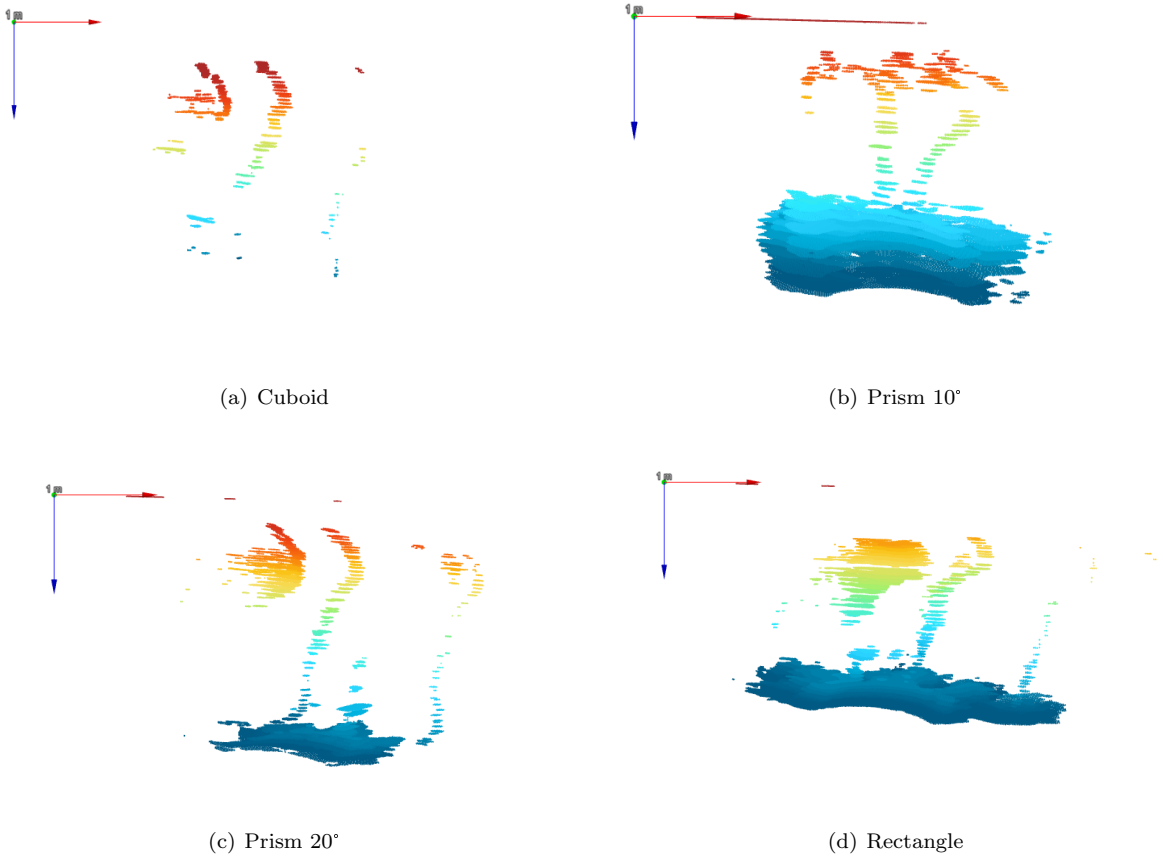
(b) Prism 10°

(c) Prism 20°

(d) Rectangle

Figure E.1.: Global close up representation of acoustic position

Figure E.2 shows the local three-dimensional point cloud representation, created with the NaviModel Free Viewer. In the x-z-visualization in fig. E.2 the backscattered points of the rectangle edge were removed, if possible, in order to allow a clear view of the elongated oval projection. In fig. E.2 (b) and (c) a second reflection of the object can be noticed on its left hand side. The course of the point cloud over the depth in all figures describes a deformed, parabolic curve, with a maximum deviation at 1.8 m to 2.1 m facing in the positive x-direction. Further the buoyancy body of the pontoon can be identified on left hand side of each figure. It is most likely that the detection is also loaded by errors, as the shape is expected to be curved.



(a) Cuboid

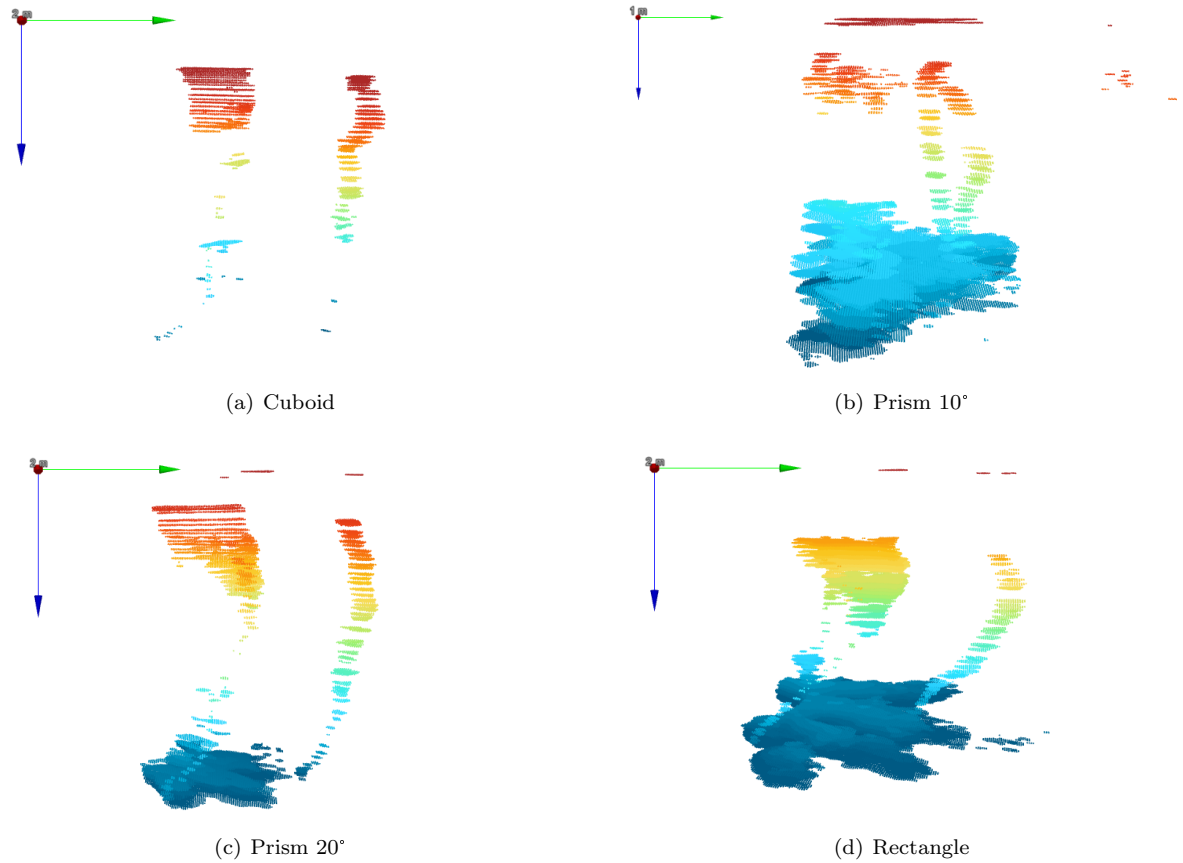
(b) Prism 10°

(c) Prism 20°

(d) Rectangle

Figure E.2.: Local three-dimensional representation of detected reference object in x-z-direction; length of coordinate axes 2 m

In E.3 shows the local spatial point cloud of all four reference object data sets along the y-z-axis. Noticeable here is the lateral offset towards the positive y-axis relative up to the depth of ca. 1.7 m in the case of the Prism 10°, Prism 20°and Rectangle.



(a) Cuboid

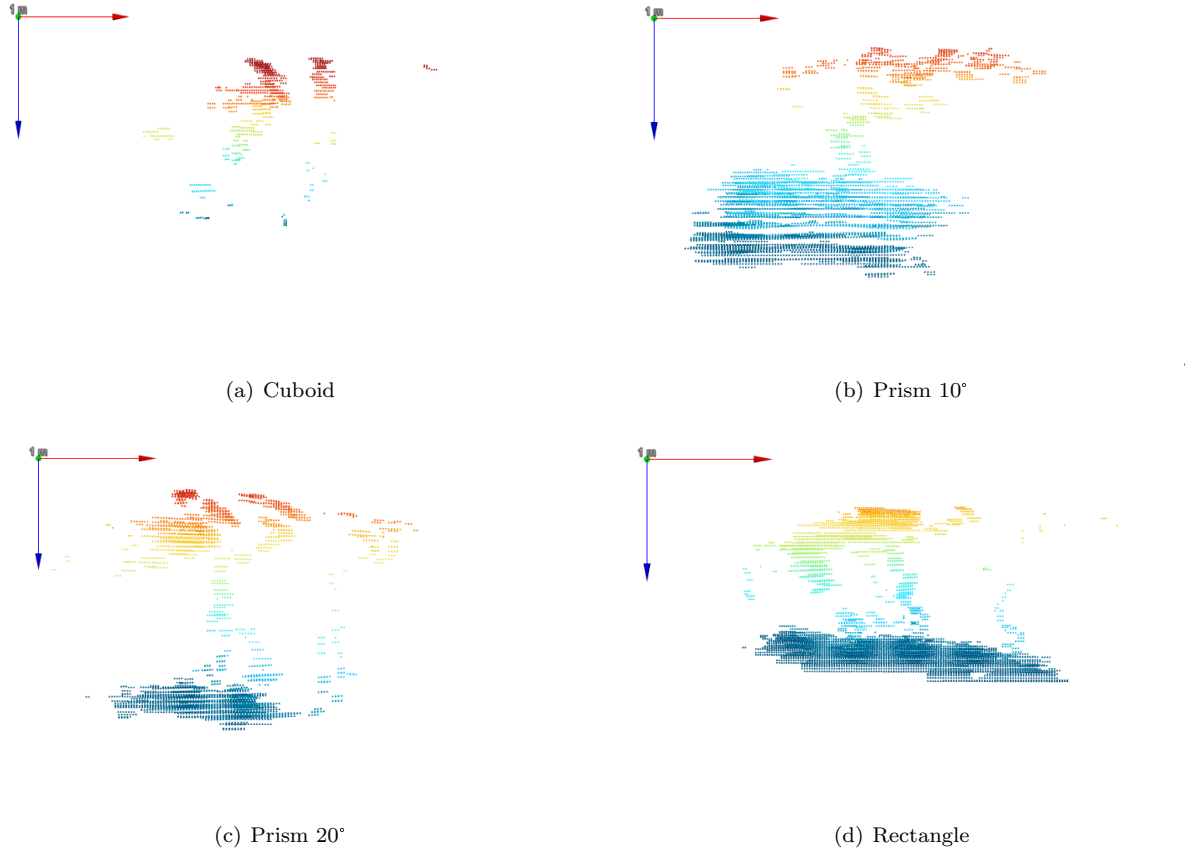
(b) Prism 10°

(c) Prism 20°

(d) Rectangle

Figure E.3.: Local three-dimensional representation of detected reference object in the y-z-direction; length of coordinate axes (a), (c) and (d) 2 m, (b) 1 m

Figure E.4 shows the global three-dimensional point cloud representation in x-z-direction, created also with the NaviModel Free Viewer. Similarities with the local representation can be made. For example the buoyancy body seems to show a similar shape in the global and local front view.



(a) Cuboid

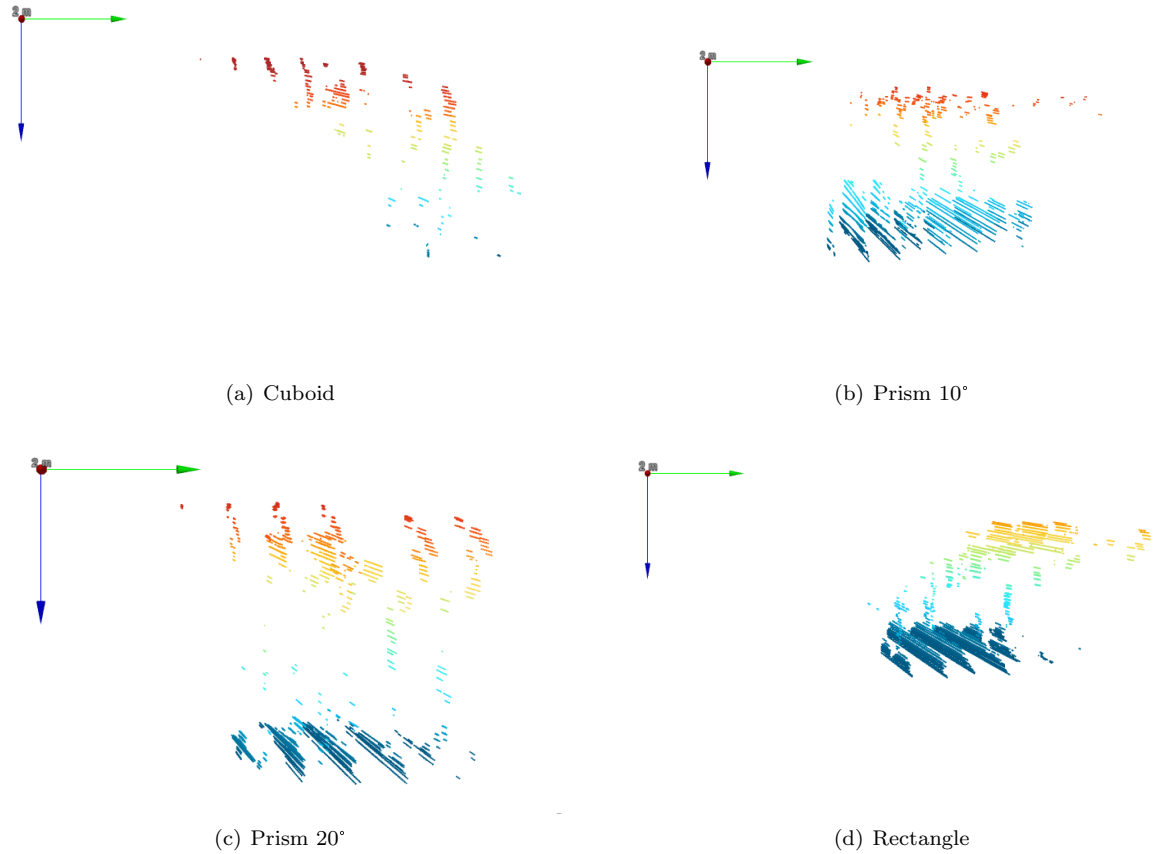
(b) Prism 10°

(c) Prism 20°

(d) Rectangle

Figure E.4.: Global three-dimensional representation of detected reference object - Front view; length of coordinate axes 1 m

Figure E.5 shows the global point cloud in a spatial representation along the y-z-axis, by means of the NaviModel Free Viewer. It can clearly the stratification of point along the x-axis, which leads to a strong distortion of the spatial view.



(a) Cuboid

(b) Prism 10°

(c) Prism 20°

(d) Rectangle

Figure E.5.: Global three-dimensional representation of detected reference object - Side view; length of coordinate axes 2 m, except for (b) 1 m

Table E.3 shows the total number of points in each point cloud representation for each subdivision, as well as the total time of recording.

Table E.3.: Determined number of points in the point cloud of each reference object dataset, subdivided into total number of points, amount of points in the specified depth segment and points of the to be assumed reference object

Reference object	Time [s]	Points					
		Global			Local		
		tot.	depth sec.	depth sec. obj. only	tot.	depth sec.	depth sec. obj. only
Cuboid	12.826	56076	13938	4890	56076	13938	4890
Prism 10°	19.731	666253	17688	-	666252	17688	-
Prism 20°	13.987	380241	28074	4731	380241	28074	4731
Rectangle	12.928	1327977	46953	1842	1327977	46953	1842

Table E.4 shows the dimension of the to be expected reference objects, measured from the depth restricted representation for each data set. It must be noted at this point, that these measurements were performed manually. Therefore slight deviations may occur when repeating the measurement.

Table E.4.: Dimension of depth restricted representation, in the specified depth of the reference objects, of local and global point clouds

Reference object	Local					Global			
	x [m]		y [m]		Planes	x [m]		y	Planes
	min.	max.	min	max.		min.	max.	[Pixel]	
Cuboid	0.18	0.22	0.12	0.17	7	0.19	0.22	1	7
Prism 10°	0.09	0.15	0.15	0.20	5	0.16	0.25	1	5
Prism 20°	0.16	0.20	0.15	0.20	6	0.19	0.21	1	6
Rectangle	0.14	0.21	0.11	0.14	4	0.19	0.28	1	4