Sea3D

An Underwater 3D Reconstruction Camera

Scientific thesis for the procurance of the degree B.Sc.
from the TUM School of Computation, Information and Technology
at the Technical University of Munich.

Supervised by	Univ.-Prof. Dr.-Ing. Sandra Hirche Dr.-Ing. Stefan Sosnowski Chair of Information-Oriented Control
Submitted by	Hamish Grant Grasmeierstrasse 19 80805, Munich 015758188224
Submitted on	Munich, 30.09.2024

Abstract

The underwater environment presents significant challenges for computer vision tasks such as depth estimation and 3D reconstruction, primarily due to image degradation and errors caused by refraction at the cameraâs waterproofing interface. In this work, we introduce an underwater stereo-vision system utilizing high-resolution industrial cameras and acceleration hardware specifically designed to address the unique challenges arising underwater with regard to computer vision. This system offers an alternative to off-the-shelf stereo-vision cameras, with the advantage of being optimized for underwater applications. We validate its performance both in the field and in a lab setting as an ROV payload and compare it to a commercially available stereo camera.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Developing a vision system with 3D reconstruction capabilities for the underwater environment poses unique challenges that are not present in typical in-air scenarios. Errors are induced by the refraction of light rays that penetrate water, glass, and air at the crossing of the camera's waterproof interface, causing significant inaccuracies in reconstructed 3D models [1]. In addition, effects such as the attenuation of light, particles causing haze (backscatter) and flickering light referred to as caustics in shallow waters lead to deteriorated image data, which reduces the effectiveness of in-air vision algorithms such as 3D reconstruction in underwater environment [2]. The challenges imposed by the nature of the environment can broadly be grouped into two categories, namely, the calibration of underwater cameras and the rectification or enhancement of underwater image data, which are a direct result of the refraction error and the degradation of underwater image data, respectively. These challenges increase the difficulty of vision tasks such as 3D reconstruction for downstream robotic applications such as positioning and collision avoidance.
Many common out-of-the-box solutions, such as Intel RealSense or Microsoft Kinect, do not address the additional challenges presented underwater. Adaptions, e.g. for the Intel RealSense or ZED 2, have been proposed to allow it to function underwater. Still, a comprehensive vision platform designed for underwater deployments, built on current state-of-the-art hardware, is not available to researchers out-of-the-box. The need for such a platform arises when researchers have specific needs when conducting underwater deployments, e.g., addressing the additional hazards underwater, a system with real-time capabilities or higher resolution. This work presents an alternative platform to common commercially available stereo cameras, built on current state-of-the-art hardware for underwater applications.

1.2 Challenges Posed by the Underwater Environment

Computer vision solutions are typically designed with the terrestrial environment in mind. However, the unique characteristics of underwater environments introduce additional challenges that a system must address to provide high-quality data for vision tasks. These challenges primarily relate to the following aspects [2]:

1. Underwater environments can be complex and difficult to reach. It can, therefore, be hard to deploy and operate the system in the field. Specialized equipment and divers are sometimes needed for data collection tasks, and the requirements for personnel are therefore high.

2. Water bodies differ in optical properties, where insufficient light can lead to dark and blurred images. Light absorption can cause an image's border to blur, resulting in a vignette effect.

3. Cameras are deployed underwater in a waterproof enclosure. The imaging sensor is separated from the target objects by a glass interface. Common interfaces are dome ports and flat ports. A double refraction effect at the interface surface from water to glass and from glass to air cannot be avoided with flat ports. Dome ports can eliminate this refraction effect, but the cameras must be perfectly centred inside the dome port half-sphere for this correction to work. Mechanically, this is a difficult task, meaning an offset from the centre is almost unavoidable. Thus, a special calibration procedure can be employed to correct these errors.

4. Light waves are affected by particles in the water, causing photos to scatter or be completely absorbed. This results in attenuation, ultimately reducing the signal information that reaches the imaging sensor. The red, green and blue discrete waves are attenuated at different rates, and their effects result in a blue-green colour effect in the image data since red light is absorbed at shorter distances in water.

5. In shallow waters up to 10m, sunlight causes a flickering effect in the data, changing the scene and decreasing the effectiveness of common feature extraction and matching algorithms used in many in-air image processing pipelines.

These aspects often cause challenges for conventional systems, which often can't meet the requirements of underwater applications researchers are interested in. Therefore, improvements are needed in order to apply them to underwater tasks. Research has demonstrated that these unique aspects can be grouped into 2 main categories, namely the degradation of underwater images and camera calibration. We will present these specific challenges in the following sections.

1.2.1 Underwater Image Degradation

RGB images are affected at every depth in the underwater environment. Still, shallow waters up to 10m are particularly challenging due to the effect of sunlight being reflected and refracted at the water's surface. This effect is referred to as flickering or caustics. It has been shown to limit the effectiveness of feature-matching algorithms, resulting in inaccurate matching and causing reduced accuracy in downstream tasks. This flickering effect and other common effects present underwater can be seen in Fig. 1.

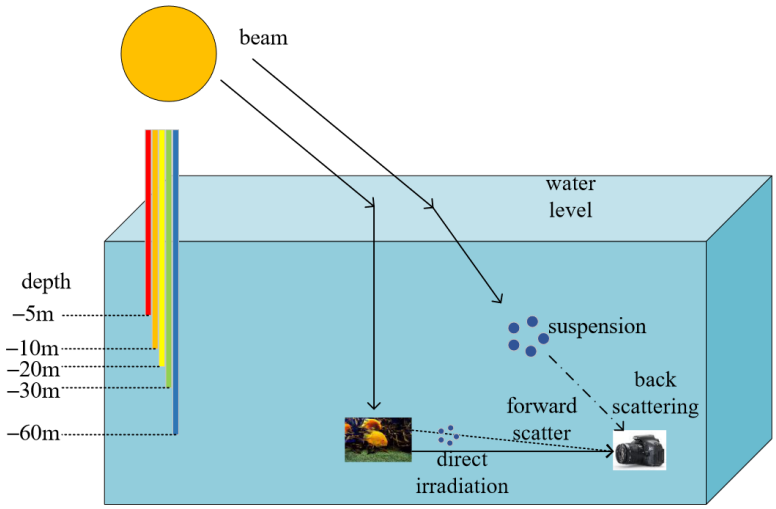


Figure 1.1: Underwater Imaging Model [2].

Light waves are attenuated at different rates when travelling through the water due to particles in the water absorbing and reflecting light, and scattering incoming light in all directions, referred to as backscatter. A model of the underwater imaging process can be observed in Fig. 1.1 where the effects of backscattering, and the different attenuation rates of light are illustrated, with red light attenuating the quickest at depths of around $5m$, whereas green and blue light waves last until around $60m$. This phenomenon results in a green-blue colour effect in underwater image data, reducing the quality and removing valuable colour information that is typically present in in-air applications.

1.2.2 Error due to Refraction

As mentioned in Sec. 1.1, underwater cameras need to be housed in a waterproof enclosure in order to function underwater. This means a glass interface separates the camera from the target object in water. Light rays reflect off of the target object and into the camera, while being refracted twice at the glass interface that protects the camera from the water. Glass interfaces are either a flat port or dome port interface as seen in Fig. 1.2. Flat port interfaces are easier to manufacture and, therefore, cheaper, but the double refraction effect is unavoidable in any case, and this makes dense 3D reconstruction much more complicated. For instance, the projection of a 3D point into the image requires the solution of a 12th-degree polynomial [1]. In addition, flat ports limit the field of view of the overall system to around 100°. In order to avoid these limitations, spherical dome port interfaces are used. They are able to correct for the refraction effect since light rays enter perpendicular to the outer interface of the sphere, but for this correction to work, the camera must be perfectly centred within the dome port. In practice, this is hard to achieve, and therefore, a calibration procedure is needed, which is described in detail in Sec. 2.3.4.

Figure 1.2: A Dome Port Interface [1].

1.3 Related Work

Several optical and acoustical methods have been developed for the goal of 3D reconstruction of the seafloor, such as sonar, laser line scanning, structured light, Structure-from-Motion, stereo vision, underwater photogrammetry, and fusion of optical and acoustical methods [2]. This work will focus on optical methods since they provide a rich representation of the environment from which colour, depth, and pose information can be extracted [2, 3]. An overview of related systems can be seen in Tab. 1.1, featuring varying processor and camera configurations, all spouting a varying amount of onboard processing but all providing a solution to underwater depth estimation and 3D reconstruction.
Stereo-based depth estimation solutions have been shown to have considerable advantages in absolute depth accuracy over monocular-based approaches [4]. Still, some monocular approaches such as the Structure-from-Motion approach in [5], an adaption of [6], have shown to correct the error induced by refraction. Hu et al. propose an underwater object identification and 3D reconstruction system based on binocular vision [7], with a two-part a system comprising the underwater perception module for data collection and the post-processing module on land. Furthermore, a stereo camera system was developed for data acquisition in [8], based on an Intel Core i7-4770TE with 4 physical cores. An in-situ stereo camera calibration procedure was proposed, provided the geometry of target objects was known beforehand. An embedded-GPU-based solution is proposed in [9], with an NVIDIA TX2 and an off-the-shelf ZED 2K stereo camera with a flat port interface. A standard camera calibration is performed, and the error induced by refraction is not discussed, although an IMU is integrated into the system to provide accurate pose estimation. An accurate 3D reconstruction method based on stereo SLAM is proposed in [10], where the error induced by refraction at the camera's waterproof glass interface is corrected for in the stereo matching and 3D reconstruction stages of the SLAM algorithm. She et al. provide adaption of [6] in [5], which accounts for the refractive error in a Structure-from-Motion approach for 3D reconstruction. The CUREE AUV is presented in [11], a fully autonomous open-source underwater robot for ecosystem exploration that features an NVIDIA Jetson Xavier for perception and path planning and is reported to use a Jetson Orin NX in a more recent iteration for GPU-accelerated vision and AI tasks in [12] by using GPU optimized algorithms. It is reported that IMU and DVL data are fused for accurate global pose estimation, allowing for high-resolution offline 3D reconstruction.

Source	Processor	Camera Model	Camera Resolution & FPS	Axuilliary Hardware
[8] 2015	Intel Core i7 477OTE Quad Core	AUT Mako G125L Gig E	1292×964 @ 30 fps	N/A
[7] 2018	AEM Cortex-A17	FMVU-03MTC	640×480 @ 60 fps	N/A
[9] 2019	Jetson TX2 with 256-Core Pascal GPUs 4x ARM Cortex-A57 CPU	ZED 2K Stereo Camera	2208×1242 @ 15 fps 1920×1080 @ 30 fps	Mti 300 IMU
[13] 2021	Raspberry Pi 4	Intel RealSense D455	1280×700 @ 90 fps	N/A
[11] 2023	NVIDIA Jetson Xavier NVIDIA Jetson Orin [12]	N/A	N/A	IMU DVL

Table 1.1: Related Work Overview

1.4 Contributions

In this work, we introduce Sea3D, a stereo camera system specifically designed for underwater depth map estimation and 3D reconstruction. Built around 4K resolution industrial cameras and the NVIDIA Jetson Orin, Sea3D serves as a robust alternative to commercially available stereo camera systems, leveraging cutting-edge hardware accelerators to explore the feasibility of real-time 3D reconstruction. The platform integrates multisensor acquisition and synchronization software, enabling seamless integration into larger robotics systems, and has undergone rigorous testing both in controlled laboratory environments and a real-world underwater field trial. We implemented and evaluated depth map estimation and 3D reconstruction algorithms to showcase the system's capabilities while tackling the unique challenges posed by the underwater environment, where many computer vision algorithms, originally designed for terrestrial applications, encounter additional obstacles.
Additionally, we explore the development of models that integrate data from multiple viewpoints and outline the adaptations made to enable real-time reconstruction capabilities in future iterations, while also discussing the current limitations.

Chapter 2

Technical Approach

2.1 Hardware

A schematic overview of the system is shown in Fig. 2.1. Conceptually, the system is divided into functional blocks: the core hardware components of the vision system are highlighted in green, support hardware in blue, the power supply in purple, and supplementary sensor modules in orange.

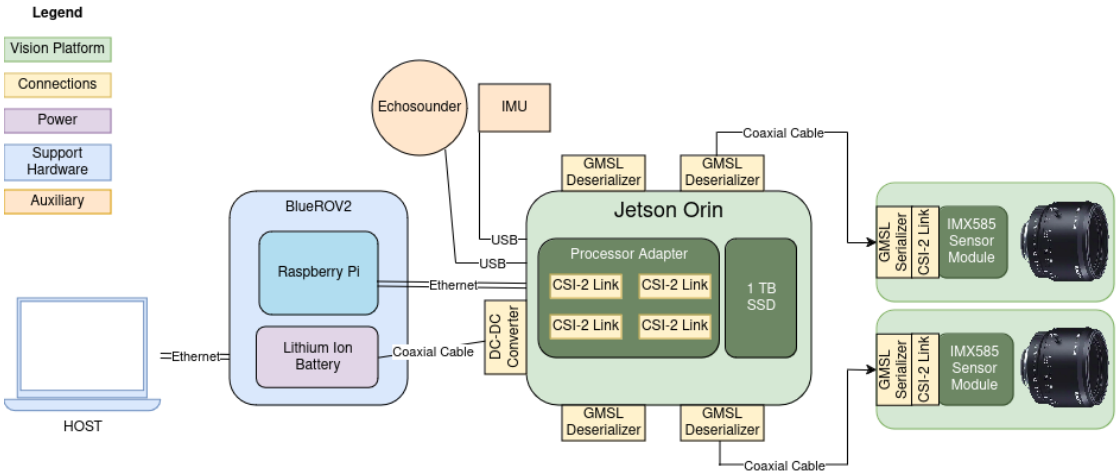


Figure 2.1: System Overview

In the following subsections, the core blocks of the vision platform are described under "Processor Payload" and "Camera Payload." Section 2.1 concludes with a description of the auxiliary sensor modules and their intended applications.

2.1.1 Processor Payload

The system's core consists of the NVIDIA Jetson Orin AGX Dev Kit and two Sony IMX585 camera sensor modules, detailed in Sec. 2.1.2. An overview of the processor's specifications can be found in Tab. 2.1. The hardware accelerators in this system are designed to enhance computationally intensive tasks. The GPU excels at accelerating basic computer vision tasks such as image enhancement, restoration, format conversion, and compression, as well as more complex tasks like image segmentation, object tracking, feature extraction, and classification. Any task that benefits from Single Instruction, Multiple Data (SIMD) architecture can leverage GPU acceleration. This is achieved by running a single instruction kernel (a function) in parallel on each GPU core, with the data cycling through these kernels [14]. In addition to vision tasks, the GPU also supports machine learning inference at the edge. The PVA, VIC, and Encoder-Decoder chips offload specific vision tasks from the GPU, freeing up processing power for other machine-learning tasks running in parallel. The PVA handles kernels like filtering, warping, image pyramids, feature detection, and FFT (Fast Fourier Transform), while the VIC is responsible for tasks such as lens distortion correction, temporal noise reduction, sharpness enhancement, colour space conversion, scaling, blending, and composition.
To provide connection to the cameras, the Orin is equipped with a processor adapter board that provides four PixelMate connection points, each offering four CSI-2 (Camera Serial Interface 2) lanes. Each connection is linked to a GMSL (Gigabit Multimedia Serial Link) deserializer board, which converts the incoming image data for the PixelMate interface. The deserializer boards are powered by a 12V external power supply, which is stepped down to 8V using a switching buck converter to provide Power-over-Coax (PoC). On the camera side, a corresponding serializer board converts the image data from the Sony IMX585 imaging sensor, adapting it for transmission via GMSL to the processor. This setup allows data transmission speeds of up to 12 Gbps [15]. In addition to the camera sensor modules, the Orin receives sensor input from a Movella Mti 300 IMU (Inertial Measurement Unit) and the BlueRobotics Ping sonar, which are connected via USB.
Except for the sonar, the whole sensor payload is housed in a 300mm long and 240mm wide IP68 waterproof enclosure, which provides "protection against being continuously immersed in water for a longer period". It has four waterproof connection points for the camera cables, an 8-pin waterproof connection for communication with the system over ethernet and a 4-pin connection for the power supply. A Lithium-ion battery is housed in an enclosure mounted on the BlueRobotics BlueROV2 underwater drone, and the power is led into the system enclosure via the aforementioned 4-pin connector. The battery powers the Orin and the 4 deserializer boards. The DC-DC voltage converter steps down the battery voltage to the desired voltage of the processor and camera adapter boards. The system enclosure is then mounted to a PVC plate and screwed to the payload shelf of the BlueROV2. An ethernet connection is established via the 8-pin connector between the Orin and

the BlueROV2's companion PC, a Raspberry Pi. The BlueRobotics Fathom ROV Tether is then used to establish a connection with the Host PC, which is situated above water in a field test deployment. These connections establish a local network between the Host, Companion PC and the Orin. We assign all the devices static IP addresses, ensuring they are assigned the same subnet. Thus, communication is established between all devices, giving access to the Orin from the host above water.

Parameter	**Specification**
Processor	NVIDIA Jetson Orin AGX
GPU	2048-core Ampere GPU 64 Tensor Cores
CPU	12-core ARM Cortex-A78AE
Hardware Accelerators	Programmable Vision Accelerator (PVA) Image Encoder-Decoder Video Imaging Compositor (VIC)
Memory (RAM)	64GB LPDDR5 at 204.8GB/s
Storage	1TB NVMe SSD 2104 MB/s read speed 1461 MB/s write speed

Table 2.1: Processor Specifications

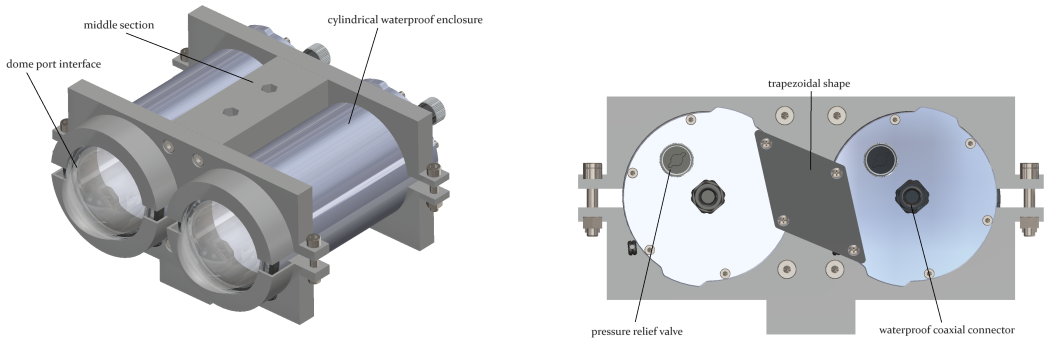
2.1.2 Camera Payload

The Sony IMX585 imaging sensors are the second core component of the vision platform. Unlike common out-of-the-box solutions, such as Intel RealSense or Microsoft Kinect cameras, these industrial sensors offer a high degree of customization. This flexibility provides the advantage of tailoring prototypes to the complex demands of underwater computer vision applications. When combined with the extensive functionality and programmability of the NVIDIA Orin, the result is a highly powerful and customizable embedded vision platform. These sensors are highly efficient SoC solutions using rolling shutter technology, offering 4K resolution and CSI-2 interface. Fixed-zoom lenses are mounted on the imaging sensors. For an overview of important sensor module and lens specifications, refer to Tab. 2.2.

Specification	Value
Optical Format (inch)	1/1.2
Resolution (pixels)	3856×2180
Pixel Depth (Bit)	12
Frame Rate (fps)	90.01
Focal length (mm)	6
Aperture (f-stop)	2.5
Vertical Field of View (°)	86.02
Horizontal Field of View (°)	64.58
Baseline (mm)	107

Table 2.2: Sea3D Camera Specifications

The counterpart to the deserializer SoCs on the processor side, described in Sec. 2.1.1, are the serializer boards connected to the imaging sensors CSI-2 adapter boards. The serializers convert the raw image data for transmission over GMSL. The sensor modules support Master-Master and Master-Salve synchronization. Although Master-Salve synchronization achieves more accurate synchronization within dozens of nanoseconds at 60 fps, the synchronization must be configured manually via a switch on the CSI-2 adapter on the camera side, resulting in tedious handling of the modules while prototyping. The Master-Master synchronization, on the other hand, can be fully configured in software through a simple one-line command of the Linux4Video2 driver while still ensuring a $7.4\mu s$ delay between frames, providing sufficient accuracy for our application [16].
The lenses, sensor modules, CSI-2 adapters and serializer boards are housed in cylindrical enclosures with diameters of 76.2 mm and lengths of 150 mm. As seen in Fig. 3, a cylindrical support structure has been designed and 3D printed to position the 3 circuit boards and the lens within the aluminium enclosure. The water-tight seal of the enclosure is achieved with two locking flanges at either end, with rubber O-rings on the outside diameter of the flanges pressing against the inside of the cylinder. One end is closed with an acrylic glass dome port interface, and the other has an aluminium backplate with two through-holes for a coaxial connector and a pressure relief valve; all points of ingress are sealed with O-rings on the outer surface.
Dome port interfaces, as opposed to flat port interfaces, have the advantage of a higher depth rating and can eliminate the error due to refraction on incoming light rays penetrating the necessary glass interface. For the refraction error to be corrected, the camera's focal point must be positioned perfectly within the centre of the dome port. This is a hard task to achieve mechanically. Our 3D-printed inner support, designed carefully in CAD in relation to the dome port and aluminium enclosure, provides sub-3-millimetre accuracy. Still, even with this precision, an error is induced on incoming light rays refracted at the acrylic interface when hitting the imaging sensor. We take a closer look at this effect in Sec. 2.3.3.



(a) Side View (b) Back View

Figure 2.2: A 3D model of the camera payload.

For the stereo depth calculation discussed in Sec. 2.4.1, the imaging sensors must have the exact same orientation and be approximately positioned in the same plane. To achieve this, we first wedge the two aluminium cylinders in a 3D-printed support structure embedded with nuts in the PLA material at connection points and fastened with bolts. An overview of the outer support, cylindrical enclosures, dome ports, waterproof coaxial connector and pressure relief valve can be seen in Fig. 2.2(a) and Fig. 2.2(b). This design must be robust enough to withstand field deployments and prevent the cameras from moving relative to each other after calibration, discussed in Sec. 3.1.1, since this would degrade the effectiveness of the calibration, leading to less accurate, or potentially unusable data. The outer 3D-printed support structure fixates the camera's horizontal position relative to each other, ensuring the camera axes are parallel. Still, the vertical and rotational positions are left to be addressed. For this, we have designed a trapezoidal shape, seen in Fig. 2.2(b), which is screwed onto the backplates of both aluminium camera enclosures. The exact shape necessary to fixate the camera's rotational position relative to each other, such that the camera imaging sensors inside the aluminium enclosures have the exact same orientation, was first accurately measured in CAD and then 3D printed. This same shape prevents the enclosures from changing their relative vertical positions as well, ensuring the sensor modules are positioned approximately in the same plane.
Finally, we must be able to mount the camera system to the ROV. For this, a fastening block is provided on the bottom of the support structure, which can be replaced independently without changing the overall design of the support. This allows, e.g. a new mounting configuration to be printed without reprinting the whole structure, with through-holes tailored to the intended mounting position on the ROV or other experiment setups.

2.1.3 Auxiliary Hardware

In addition to the core vision sensor package, which includes the camera sensor, SoCs such as the GMSL Serializer and Deserializer, and the processor, additional sensors have been integrated into the payload, enriching the recorded data packages. The echosounder provides sensor data that can serve as an approximate ground truth for stereo depth estimation, while the IMU can be fused with image data to enhance pose estimation in future project iterations. However, integrating the various data streams, such as those from the echosounder and IMU, with the image data presents a significant challenge. This integration is discussed in the following section, Sec. 2.2.

2.2 Software

The Jetson Orin includes its own version of embedded Linux, known as Linux for Tegra (L4T), which provides the operating system along with essential drivers for hardware acceleration and media interfacing. The software environment for data acquisition and recording must fulfil the following requirements:

- Provide a low-level interface for reading image data.
- Allow the user to monitor the incoming data in real-time.
- Synchronized and structured data recording for offline processing and analysis.

Most of these challenges are solved with Python as the underlying software framework, with additional software packages such as OpenCV, Gstreamer and ROS2 providing a lot of the crucial functionality necessary to achieve the above-mentioned goals. An overview of the software pipeline for data acquisition, monitoring and recording can be seen in Fig. **??**.

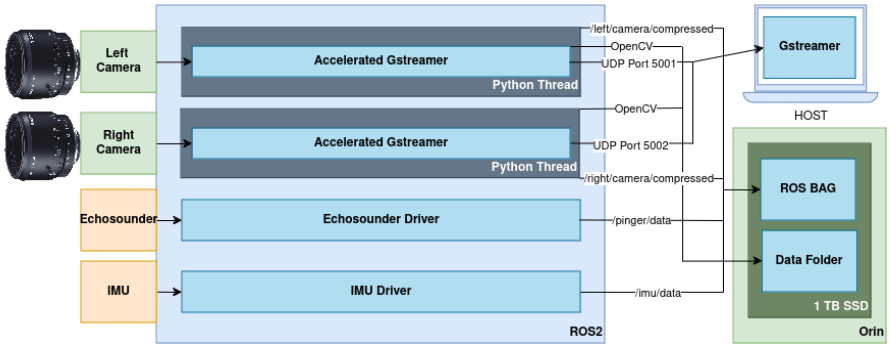


Figure 2.3: Software Pipeline

A significant challenge with rapidly advancing technology, such as the Jeston Orin, is to provide a stable software environment, providing compatibility between all the software packages and drivers and allowing for quick reproducibility after system failures due to, e.g. broken dependencies. The Jetson Orin is based on aarch64 architecture, which differs from the more common x86_64 architecture, for which many software packages do not provide pre-built binaries, forcing the developer to build many software packages from scratch. It is advisable to make use of open-source build scripts to minimize the time spent building and debugging specific software builds, such as this script recommended by NVIDIA for building OpenCV on Jetson in[17]. Additional stability can be ensured, and time saved by making use of a virtual software environment, such as CONDA, to keep track of individual software packages, versions and dependencies and exporting and backing up stable environments frequently in order to rebuild your environment quickly after system failures occurring during development.

2.2.1 Data Acquisition

Before each experiment, it is necessary to configure the Master-Master synchronization of the camera modules previously mentioned in Sec. 2.1.2. Failure to synchronize the imaging sensors may result in unusable data due to not knowing whether a left and right stereo image pair are, in fact, a capture of the same scene. This ambiguity may render a collected dataset unusable or, at the very least, introduce unnecessary errors, resulting in inaccurate data and an unsatisfactory result during post-processing. Thankfully, eliminating this source of error is easily accomplished with the following two command lines:

```
v4l2-ctl -d /dev/video0 -c operation_mode=0 -c synchronizing_function=1
v4l2-ctl -d /dev/video1 -c operation_mode=0 -c synchronizing_function=2
```

After synchronizing the cameras, we use the GStreamer multimedia framework to process the incoming data. GStreamer, a core component of our environment, enables developers to quickly construct low-level media processing pipelines using a simple string format. In our pipeline, we leverage hardware-accelerated video acquisition and image compression. This approach eliminates the need for buffer copies at various stages, significantly improving processing speed. Additionally, efficient hardware compression allows real-time monitoring of the video data on the host machine via the tether cable. With the GStreamer pipeline string stored in the variable *gstreamer_pipeline*, the data is passed to OpenCV in Python using the *cv2.VideoCapture()* method, which initiates an OpenCV capture object:

```
capture = cv2.VideoCapture(gstreamer_pipeline)
```

This allows us to perform additional processing in OpenCV before saving the data to the 1TB SSD or the ROS2 Bag.

2.2.2 Networking and Monitoring

As mentioned in Sec. 2.1.1 and seen in Fig. 2.1, a local area network is established between the Host, the ROV companion and the Orin via ethernet. Once the payloads are watertight, communication with the Orin and, by extension, the camera modules occurs solely through the local network. Using the SSH protocol, the Orin can be controlled remotely via a command line interface, allowing for system configuration, such as synchronizing camera sensors, launching Python scripts for data acquisition, monitoring, and recording, as well as executing ROS launch files, as detailed in Section 2.2.3.
Another accelerated GStreamer pipeline is initiated on the Orin to transmit the incoming video data to the host machine for real-time monitoring. This pipeline compresses the incoming data and streams it over the local network using the UDP

protocol. On the top side machine, a further Gstreamer client pipeline receives the data from the UDP server and outputs it to OpenCV in Python, to be displayed on the host machine's monitor.
The groundwork for transmission of data via ROS2 has been laid for future iterations of the system, with the data being transmitted over ROS-Topics, which can be subscribed to by the top-side machine running ROS2. ROS2 integration proves a significant step in modularising the sensor acquisition, with the intention of providing real-time 3D reconstruction to the top-side machine in future iterations of the project. It not only allows for future integration with more complex software packages but provides an efficient way of recording synchronized data streams coming from different sensor modules.

2.2.3 Data Recording

Our initial approach of recording data as a video file proved inefficient compared to saving each individual time-stamped frame in separate folders for each camera source. The latter method significantly enhances post-processing and analysis, as it utilizes time stamps providing microsecond-level accuracy. The 1TB SSD offers ample storage for 7 hours of deployment, with a 1-hour deployment consuming approximately 140.4 GB of storage. We opted for the PNG file format due to its lossless compression despite its larger storage requirements compared to the more space-efficient JPG format.
The ROS2 packages discussed in Section 2.2.2 allow us to read each data stream concurrently-from the cameras to the IMU and the echosounder-while saving the time-stamped data to a ROS2 bag file. This capability enables us to record a synchronized dataset, which can be used to simulate a field test and develop functional 3D reconstruction capabilities in future iterations of Sea3D before actual field deployment. The images are saved in a compressed format to the SSD, with a 1 hour recording consuming approximately 15 GB of storage space.

2.3 Camera Calibration

2.3.1 Pinhole Camera Model

To understand the image formation process, we consider the pinhole camera model, as seen in Fig. 2, where a camera is modelled as a single point which all rays projected onto the image plane pass through; we refer to this point as the *projection centre*. The image formation model uses *projective transformation* or *projective geometry*, which describe the projection of 3D points in space to 2D points on a plane, in our case, the image plane.
Observe that a *homogeneous coordinate* (X, Y, Z, T) in $\mathbb{P}^3$, with constant X, Y and Z components and varying T, represents a ray in space, that passes through the projection centre, projecting all the points along the ray to a single point, i.e. the T component is irrelevant for the resulting projected point. The same applies to the homogenous coordinate of an arbitrary 2-dimensional point (x_1, x_2, x_3), which maps to a point $(x_1/x_3, x_2/x_3)$ in $\mathbb{R}^2$. The most general transformation from a homogenous coordinate of a 3D point to a homogenous coordinate of a 2D point is described by a 3×4 matrix, which we refer to as the *camera matrix* [18]. The camera matrix represents the image formation process, mapping points in $\mathbb{R}^3$ to points in $\mathbb{R}^2$,

$$\boldsymbol{sp} = \boldsymbol{K}\,[\boldsymbol{R}|\boldsymbol{t}]\,\boldsymbol{p_w} \tag{2.1}$$

where $\boldsymbol{p_w}$ is a 3D point with respect to the world coordinate system, $\boldsymbol{p}$ is a 2D pixel in the image plane, $\boldsymbol{K}$ is the camera intrinsic matrix, $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotation matrix and translation vector that describe the change of coordinates from the world to the camera coordinate systems and $\boldsymbol{s}$ is the projective transformation's arbitrary scaling and not part of the camera model.
The camera intrinsic matrix $\boldsymbol{K}$ projects 3D points given in the camera coordinate system to 2D pixel coordinates:

$$\boldsymbol{p} = \boldsymbol{K}\boldsymbol{p_c} \tag{2.2}$$

The parameters needed to define the camera intrinsic matrix $\boldsymbol{K}$ are the focal lengths f_x and f_y which are expressed in pixel units, and the principal point (c_x, c_y) which is usually close to the centre of the image:

$$\boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

which yields (2.2):

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2.4}$$

We have now defined a transformation from 3D to 2D image space regarding the camera coordinate system, if we would like to understand the correspondence between 2D points on the image plane and 3D points with reference to the world coordinate system, we are going to have to consider the *extrinsic parameters* as well, namely the joint rotation-translation matrix $[\boldsymbol{R}|\boldsymbol{t}]$. This transformation consists of a 3-by-4 projective transformation and a homogenous transformation. The projective transformation maps 3D points represented in camera coordinates to 2D points on the image plane represented in normalized camera coordinates, with $x' = X_c/Z_c$ and $y' = Y_c/Z_c$:

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.5}$$

The homogenous transformation represents a change of basis from the world coordinate system w to the camera coordinates system c. Thus, given the extrinsic parameters with the rotation matrix $\boldsymbol{R}$ and the translation vector $\boldsymbol{t}$, we are able to transform a 3D point given in world coordinates $\boldsymbol{p}_w$ to a 3D point given in the camera coordinate system $\boldsymbol{p}_c$. This transformation is written as follows:

$$\boldsymbol{p}_c = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \boldsymbol{p}_w \tag{2.6}$$

Adding the intrinsic parameters and extrinsic parameters together, we finally write out (2.1):

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.7}$$

which finally yields our desired transformation. These models assume that straight lines in $\mathbb{R}^3$ transform to straight lines in $\mathbb{R}^2$. In practice, this is not the case since wide-angle lenses suffer from radial distortion, which results in the curvature of a straight line in the resulting projection. A distorted projected point $(x_{distorted}, y_{distorted})$, can be denoted as [19],

$$x_{distorted} = x(1 + k_1 r + k_2 r^4 + k_3 r^6) \tag{2.8}$$

$$y_{distorted} = y(1 + k_1 r + k_2 r^4 + k_3 r^6) \tag{2.9}$$

where $r^2 = x^2 + y^2$, and k_1,k_2 and k_3 are the *radial distortion parameters* and (x, y) being the undistorted image point. In practice, a quadratic model of distortion is sufficient, i.e. we set $k_3 = 0$ and tangential distortion is commonly ignored since it can lead to less stable estimates [19].

2.3.2 In-Air Calibration

Given a planar calibration object such as a checkerboard, we are able to use a corner detection algorithm to find checkerboard corners in an image, which we can then use as known points to calibrate our camera. Given a set of checkerboard images for calibration, we assume that the checkerboard is situated at $Z = 0$ and map the calibration points $(X_i, Y_i, 0)$ to image coordinates (x_i, y_i) by computing a separate homography $\tilde{\boldsymbol{H}}$ for each input image, this way we are able to recover the intrinsics in conjunction with the camera pose (i.e. extrinsics):

$$\boldsymbol{p} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\boldsymbol{H}} \boldsymbol{p}_w \tag{2.10}$$

$\boldsymbol{r_0}$ and $\boldsymbol{r_1}$ denote the first two columns of $\boldsymbol{R}$ and $\sim$ indicates equality up to scale. From here, we are able to form linear constraints on the nine entries in the $\boldsymbol{B} = \boldsymbol{K}^{-T}\boldsymbol{K}^{-1}$ matrix, which we refer to as the image of the absolute conic and commonly use in camera calibration. It has been shown that the camera matrix $\boldsymbol{K}$ can be recovered from $\boldsymbol{B}$ using a matrix square root and inversion [19].
We find the radial distortion parameters k_1 and k_2 by back-projecting observed 2D image points to the checkerboard with the intrinsic and extrinsic parameters, and minimizing the resulting re-projection error.

2.3.3 Refractive Camera Model

The error induced by refraction in Sec. 1.2.2 can be eliminated by positioning the camera's optical centre perfectly at the spherical centre of a dome port interface. In practice, this is a challenging task to achieve, rendering a slight offset unavoidable in most cases. The effect of the camera offset is illustrated in Fig. 2.4. If the camera is offset from the dome port, straight lines appear to jump at the air-water interface in an image captured by a halfway-submerged camera. The refraction at the dome port interface shifts the intersection point of light ray trajectories with the image plain, resulting in inaccuracies in the disparity map from Sec. 2.4.1.

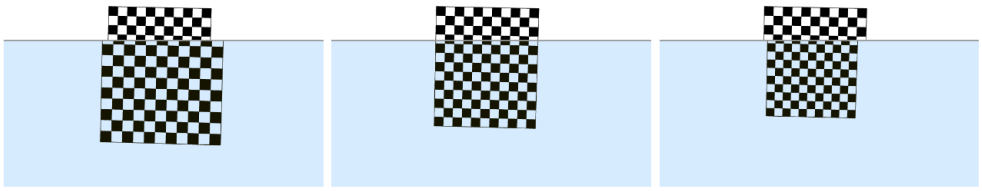


Figure 2.4: Simulated chessboard images with parts above and below the water line. The lens is misaligned forward (left), aligned (centre), misaligned backwards(right) [1]

In order to compute geometrically accurate disparity maps, a model of refraction is necessary to compute the intersection points of refracted rays with the camera's imaging plane.
A light ray refracted at the interface of a medium with an index of refraction n_i to a medium with the index of refraction n_r can be expressed as [1],

$$r = \frac{n_i}{n_r}\boldsymbol{i} + (\frac{n_i}{n_r}\cos(\theta_i) - \sqrt{1 - \sin^2(\theta_r)})\boldsymbol{n} \tag{2.11}$$

where $\boldsymbol{i}$ and $\boldsymbol{n}$ denote the incident and refracted ray respectively, and $\boldsymbol{n}$ is the normal vector at the intersection point.θ_i is the incident angle and can be calculated by $\theta = \arccos(-i \cdot n)$. θ_r is the refraction angle, which can be derived from the incident angle according to Snell's law $n_i \sin(\theta_i) = n_r \sin(\theta_r)$.
With this formula, we can derive the in-water viewing ray projected from a 2D point $\boldsymbol{p}$ on the image plane by first computing the in-air ray with the camera intrinsic matrix $\boldsymbol{K}$ from Sec. 3.1.1,

$$\overrightarrow{\boldsymbol{Ray}_{air}} = \frac{K^{-1}\tilde{p}}{\|K^{-1}\tilde{p}\|_2} \tag{2.12}$$

.
Assuming the dome offset parameters $\delta\boldsymbol{C}_{dome} = (\delta X_{dome}, \delta Y_{dome}, \delta Z_{dome})$ are known, we can find the inner and outer spheres of the acrylic dome by transforming the unit sphere $\boldsymbol{Q} = diag\{1, 1, 1, -1\}$ according to,

$$\boldsymbol{H}(d, \delta\boldsymbol{C}_{dome}) = \begin{bmatrix} d & 0 & 0 & \delta X_{dome} \\ 0 & d & 0 & \delta Y_{dome} \\ 0 & 0 & d & \delta Z_{dome} \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.13}$$

$$\boldsymbol{D}(d, \delta C_{dome}) = (\boldsymbol{H}^{-1})^T \boldsymbol{Q} \boldsymbol{H}^{-1} \tag{2.14}$$

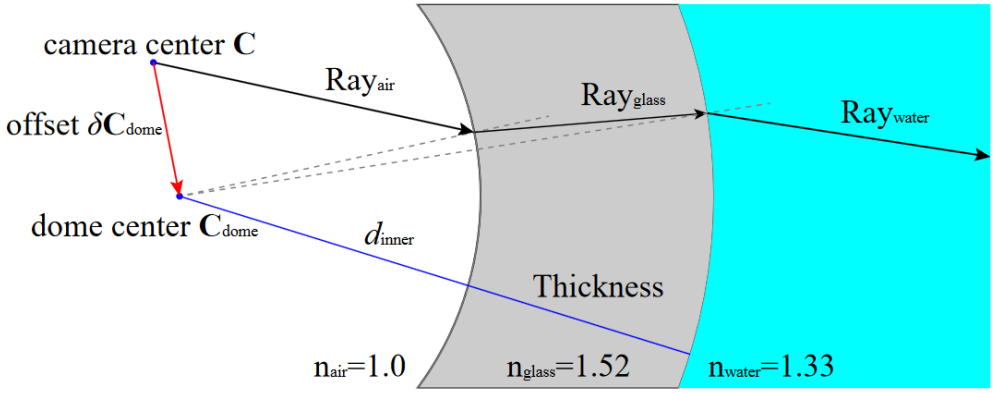


Figure 2.5: Refractive viewing ray in misaligned dome port [1].

By noticing that all 3D points on the unit sphere are satisfying the equation $\boldsymbol{p}_u^\top \boldsymbol{Q} \boldsymbol{p}_u = 0$, we can find the intersection point $\boldsymbol{p}_{a/g}$ of the in-air ray with the inner dome by solving,

$$\boldsymbol{p}_{a/g}^\top \boldsymbol{D}_{air} \boldsymbol{p}_{a/g} = 0 \text{ with } \boldsymbol{p}_{a/g} = \lambda_{air} \overrightarrow{\boldsymbol{Ray}}_{air} \tag{2.15}$$

simplifying to a simple equation in terms of the ray scale-factor λ_{air}, we obtain the normal vector, which allows us to determine the refracted ray $\overrightarrow{\boldsymbol{Ray}}_{glass}$ using (2.3.3). For the glass-water interface, the intersection point $\boldsymbol{p}_{g/w}$ can be calculated in the same way by

$$\boldsymbol{p}_{g/w}^\top \boldsymbol{D}_{glass} \boldsymbol{p}_{g/w} = 0 \text{ with } \boldsymbol{p}_{g/w} = \lambda_{glass} \overrightarrow{\boldsymbol{Ray}}_{glass} \tag{2.16}$$

allowing us to compute the intersection point on the outer interface and, finally, the ray direction in the water. The geometrical relationships of each section of the refracted ray can be seen in Fig. 2.5.
We can now back-project points observed on the image plane into 3D space, such as projecting observed checkerboard corners back to the actual checkerboard. Subsequently, we can calculate the error by comparing the coordinates of the back-projected points to the actual points, which provides the re-projection error for refractive geometry. By shifting the assumed location of our camera centre within the dome port until the re-projection error is minimized, we can find the actual offset location of our camera centre in the dome port, which we will discuss in the following section.

2.3.4 In-Situ Calibration

The goal of in-situ calibration is to determine the camera dome port offset $\boldsymbol{v}_{\text{off}}$ discussed in Sec. 2.3.3. For this, we must find the axis of refraction $\boldsymbol{a}$, defined as the line intersecting the dome port and camera centres, i.e. the line on which the offset camera centre is positioned. We start by first finding the centre of refraction, which is defined as the intersection point of the axis of refraction with the image plane. Given a checkerboard calibration target, a checkerboard corner $\boldsymbol{x}_c$ detected in-air can be related to an observed image point $\boldsymbol{x}_a$ by a perspectivity, a special kind of homography [20],

$$\boldsymbol{x}_a \backsimeq \boldsymbol{H}\boldsymbol{x}_c \tag{2.17}$$

By now submerging the target and camera in water, the refracted 2D image point $\boldsymbol{x}_r$ on the imaging plane and the originally observed point $\boldsymbol{x}_a$, do not align, but $\boldsymbol{x}_r$ must lie on a line $\boldsymbol{q} = [\boldsymbol{r}] \times \boldsymbol{x}_a$ connecting the in-air point $\boldsymbol{x}_a$ with the refraction centre $\boldsymbol{r}$, satisfying $\boldsymbol{x}_r^T\boldsymbol{q} = 0$. Replacing $\boldsymbol{q}$ and $\boldsymbol{x}_a$ yields a constraint that must hold between all points in the refracted image and their corresponding chessboard positions:

$$\boldsymbol{x}_r^\top [\boldsymbol{r}] \times \boldsymbol{H}\boldsymbol{x}_c = 0 \tag{2.18}$$

The relationship is similar to epipolar geometry discussed in Sec. 2.4.1. We define the matrix $\boldsymbol{F} = [r] \times \boldsymbol{H}\boldsymbol{x_c}$ which can be estimated by observing at least 8 different points using the eight-point algorithm discussed in Sec. 2.5.1, for fundamental matrix estimation. After estimating $\boldsymbol{F}$ and refactorizing, the centre of refraction $\boldsymbol{r}$ can be extracted as the left null vector of $\boldsymbol{F}$ [20]. Pose estimation yields the camera rotation matrix $\boldsymbol{R}$ and the axis of refraction can then be found by solving,

$$\boldsymbol{a} = \frac{\boldsymbol{R}^\top \boldsymbol{r}}{\|\boldsymbol{R}^\top \boldsymbol{r}\|} \tag{2.19}$$

Which yields the line on which the offset camera centre is positioned. We now find the direction along which the camera is offset by drawing a line through two refracted chessboard corners $\boldsymbol{x}_{r,1}, \boldsymbol{x}_{r,3}$ and checking whether a third corner $\boldsymbol{x}_{r,2}$ in between them is projected onto the same side of the line as the refraction centre by using the convexity test:

$$\text{convexity}(\boldsymbol{x}_{r,1}, \boldsymbol{x}_{r,2}, \boldsymbol{x}_{r,3}, \boldsymbol{r}) = \text{sign}((\boldsymbol{x}_{r,1} \times \boldsymbol{x}_{r,3})^\top \boldsymbol{x}_{r,2} \cdot (\boldsymbol{x}_{r,1} \times \boldsymbol{x}_{r,3})^\top \boldsymbol{r})) \tag{2.20}$$

We define the positive and negative refraction poles as the intersection points of the axis of refraction with the complete sphere extended from the dome port half sphere, as the intersections that are closer and further away from the camera, respectively. We can then deduce if the camera is in front of or behind the dome port centre, thus

determining the sign of $\boldsymbol{v}_{\text{off}}$. We then optimize the offset $\boldsymbol{v}_{\text{off}}$ by iteratively back-projecting refracted points on the image plane and shifting $\boldsymbol{v}_{\text{off}}$ until the reprojection error is sufficiently small. This is achieved by taking a set of calibration images of a chessboard and minimizing the energy for a detected 3D chessboard corner $\boldsymbol{X} = (X, Y, 0)$ [20], with the measured i^{th} 2D corner in the j^{th} image denoted as x_i^j,

$$E(\Theta) = \sum_{i \in \Omega} \sum_{j \in \Phi} \| \boldsymbol{X}_i^j - \hat{\boldsymbol{X}}_i^j \left(\boldsymbol{x}_i^j, \boldsymbol{v}_{\text{off}}, \boldsymbol{P}^j \right) \|^2 \tag{2.21}$$

where $\hat{\boldsymbol{X}}$ is the 3D point on the chessboard plane back-projected from $\boldsymbol{x}$, and $\boldsymbol{P}^j$ is the camera pose in the j^{th} image, which must be determined along with $\boldsymbol{v}_{\text{off}}$ resulting in 6m + 3 parameters to be estimated $\Theta = (\boldsymbol{v}_{\text{off}}, \boldsymbol{P}_1, \boldsymbol{P}_2, ..., \boldsymbol{P}_m)$, with m being the amount of chessboard images.

2.4 Depth Estimation

2.4.1 Stereo Depth Estimation

Stereo matching is the process of finding the same feature in multiple images and building a 3D model of the scene by converting their 2D positions into 3D depths. Given a pair of stereo images, we efficiently solve the problem of stereo matching with epipolar geometry and the warping of each stereo image onto the same plane, which we call stereo rectification.
We ask ourselves, how can we use the calibration data we have about our cameras, which we received from Sec. 3.1.1 to reduce the number of potential correspondences, and hence speed up matching and increase its reliability? For this we use the concept of epipolar geometry, an overview is provided in Fig. 2.6.

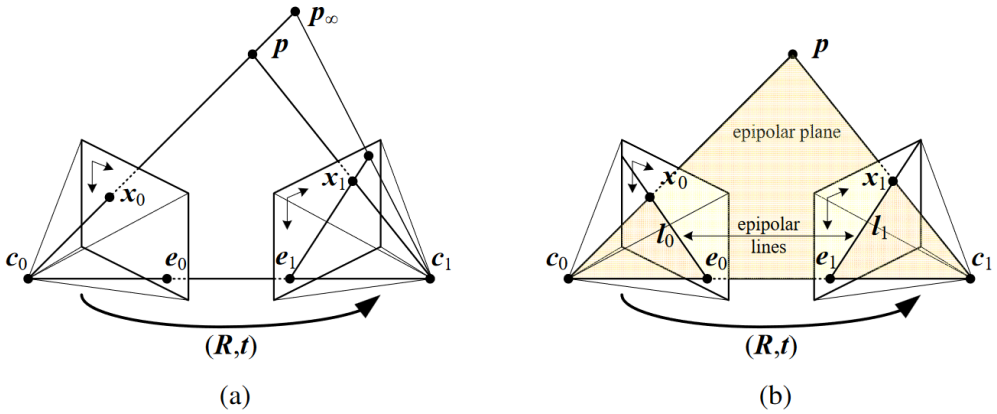


Figure 2.6: Epipolar geometry: (a) epipolar line segment corresponding to one ray; (b) corresponding set of epipolar lines and their epipolar plane. [19]

The interest point $\boldsymbol{p}$ is projected onto the image planes of camera 0 and camera 1 as the pixel coordinates $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$, respectively. $\boldsymbol{x}_0$ projects to an epipolar line in the image plane of camera 1, which is bounded by the viewing ray of the projection of the camera centre $\boldsymbol{c}_1$ to infinity $\boldsymbol{p}_\infty$, and the intersection of the line connecting the camera centres $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ with the image plane of camera 1, which we call the episode $\boldsymbol{e}_1$. This line segment is then projected back to the image plane of camera 0 to receive another epipolar line which is now bounded by the second epipole $\boldsymbol{e}_0$. Extending both line segments into infinity yields a pair of epipolar lines, which represent the intersection of the epipolar plane, the plane defined by the world point $\boldsymbol{p}$ and the camera centres $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$. The epipolar plane can be seen in Fig. 2.6(b).
This information is encoded in the so-called epipolar constraint,

$$\boldsymbol{x}_1^\top \boldsymbol{E} \boldsymbol{x}_0 = 0 \tag{2.22}$$

which tells us that the corresponding 2D point in image 1 $\hat{\boldsymbol{x}}_1$ to the observed point $\hat{\boldsymbol{x}}_0$

in image 0, must lie on a specific line, the epipolar line, in the opposite image, image 1. This reduces our search for point correspondences to a single line, significantly simplifying our search. The matrix $\boldsymbol{E}$ is called the fundamental matrix and is defined as,

$$\boldsymbol{E} = [\boldsymbol{t}] \times \boldsymbol{R} \tag{2.23}$$

Estimating the fundamental matrix will allow us to find the translation vector $\boldsymbol{t}$ and rotation matrix $\boldsymbol{R}$ from two image pairs, which we call the *camera pose* . We need N observations of the same point in both images, $(\hat{\boldsymbol{x}}_{i0}, \hat{\boldsymbol{x}}_{i1})$, with $i \in N$. These points can be determined with a calibration target and a corner detection algorithm, which we then use to construct a system of equations with the 9 entries of $\boldsymbol{E} = (e_{00}, ..., e_{22})$ and the observed points:

$$[\hat{\boldsymbol{x}}_{i1}, \hat{\boldsymbol{x}}_{i0}^{\top}] \circ \boldsymbol{E} = 0 \tag{2.24}$$

We need at least 8 observations to compute an estimate (up to scale) of the fundamental matrix $\boldsymbol{E}$ with Singular Value Decomposition. The direction $\hat{\boldsymbol{t}}$ of the translation vector $\boldsymbol{t}$ can then be determined from $\boldsymbol{E}$, by noticing that $\boldsymbol{E}$ is a singularity, and that $\hat{\boldsymbol{t}}$ lies within the null space of $\boldsymbol{E}$ [19],

$$\hat{\boldsymbol{t}}\boldsymbol{E} = 0. \tag{2.25}$$

Solving for $\hat{\boldsymbol{t}}$ gives us a unit vector with the direction of $\boldsymbol{t}$, even though the sign is not known. The rotation matrix $\boldsymbol{R}$ is left to be determined to estimate depth. The rotational information is encoded in the two remaining non-zero values of the SVD of $\boldsymbol{E}$, yielding two possible solutions of the rotation matrix $\boldsymbol{R}$. These two solutions are paired with the direction and possible signs $\pm\hat{\boldsymbol{t}}$, and the combination is selected that yields the most points seen in front of both cameras. Since we are using a calibration target with known reference points, such as a checkerboard, $\hat{\boldsymbol{t}}$ can be scaled to yield the actual translation vector $\boldsymbol{t}$ since the ratio between the distance of the reference points on the calibration target and the distance of the projected points gives us our scale factor. The computed translation vector $\boldsymbol{t}$ and rotation matrix $\boldsymbol{R}$, can then be used for triangulation since the norm of the translation vector $\boldsymbol{t}$ is equal to the baseline of the stereo camera B.

To simplify our search for feature correspondences even further, we perform rectification of the stereo image pair. Rectification is the process of warping the stereo pair such that horizontal scan lines in the image, match the epipolar lines of feature points. To achieve this, we rotate our image planes so that they are perpendicular to the line between the camera centres c_0 and c_1, effectively projecting the images into one plane, and then twisting the images such that the *up vector* (the camera y-axis), is perpendicular to the camera center line. This ensures that corresponding epipolar lines are horizontal, and the disparity for points at infinity is 0. We finally re-scale images to account for different focal lengths, magnifying the smaller image to avoid aliasing. The resulting *standard rectified geometry* is employed in a lot of

stereo camera setups since it leads to a very simple inverse relationship between 3D depths Z and disparities d:

$$d = f\frac{B}{Z} \tag{2.26}$$

where the camera baseline is denoted as B and the focal length as f. We can apply this depth calculation to each point correspondence in the stereo image pair, thereby creating a depth map $\boldsymbol{D}$ of the scene. A depth map is generally interpreted as a matrix, with slightly reduced dimension $N \times M$ than the input stereo pair, where each entry of the matrix d_{ij} corresponds to the depth estimated at the pixel coordinate (x, y) within the stereo pairs region of interest (ROI),

$$\boldsymbol{D} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \tag{2.27}$$

2.4.2 Semi-Global Matching

Many stereo-matching algorithms are based on matching pixels by minimizing the dissimilarity cost of potential matches in a stereo image pair, where the dissimilarity cost is often defined as the absolute or squared difference of pixel intensity. For a given pixel $\boldsymbol{p}$ in an image and a potential match $\boldsymbol{q}$ in the corresponding image of the stereo pair, the squared difference of pixel intensity is defined as

$$(I(\boldsymbol{p}) - I(\boldsymbol{q}))^2 \tag{2.28}$$

Given a rectified stereo image pair, as discussed in Sec. 2.4.1, a potential match for a pixel with coordinates (x, y) is searched for in the corresponding image according to $\{(\hat{x}, y) \mid \hat{x} \geq x, \hat{x} \leq x + D\}$, where D is the maximum allowed disparity shift.
Notice that we fixate the y value to a constant since rectification has resulted in epipolar lines overlapping with horizontal lines extended through both images at fixed y coordinates, substantially reducing the subset of potential matches.
Such pixel-wise matching methods are referred to as local matching algorithms, which are efficient to implement, offering real-time performance [19], but often result in many erroneous matches. More robust methods referred to as Global Matching algorithms, take into account larger pixel regions of the image for stereo matching and set global smoothness constraints on the output data. These approaches are often formulated in an energy minimization framework, where the goal is to find a solution $\boldsymbol{d}$ that minimizes the global energy E($\boldsymbol{D}$) which depends on the disparity map $\boldsymbol{D}$ [21],

$$E(\boldsymbol{D}) = \sum_{\boldsymbol{p}} \left((C(\boldsymbol{p}, D_p) + \sum_{\boldsymbol{q} \in N_p} P_1 T\left[|D_p - D_q| = 1\right] + \sum_{\boldsymbol{q} \in N_p} P_2 T\left[|D_p - D_q| > 1\right] \right) \tag{2.29}$$

The first term represents the pixel-matching costs of all the disparities in $\boldsymbol{D}$. The second term adds a constant penalty P_1 to small changes in disparity in the neighbourhood N_p of $\boldsymbol{p}$, that is, disparity changes of 1. This allows for smooth transitions and slanted surfaces to be preserved. In contrast, the last term adds a penalty P_2 to greater changes in intensity to the neighbours in N_p, resulting in preserved discontinuities. This approach minimizes the dissimilarity cost of scanlines in the left and right pair of a stereo image, taking the dissimilarity cost of individual pixel intensity values and neighbouring pixel intensities along the scan line into consideration. Even with this global approach, streaking artefacts are introduced to the output since smoothing does not occur across scan lines.
A solution to this problem is presented by semi-global matching, which aims to minimize the dissimilarity cost and apply the smoothing constraint to multiple scan lines from all directions that reach pixel $\boldsymbol{p}$. This is achieved by aggregating the (smoothed) cost of reaching pixel $\boldsymbol{p}$ from at least 8 or better 16 directions as $S(\boldsymbol{p}, d)$. The disparity value for a pixel $\boldsymbol{p}$ in the disparity map $\boldsymbol{D}$ is then selected according to:

$$d^* = \arg\min_d S(\boldsymbol{p}, d) \tag{2.30}$$

2.4.3 DNN Monocular Depth Estimation

Monocular cameras can also estimate depth information by training a deep neural network. [22] proposes a network architecture that computes depth as statistical depth distributions and outputs a probability depth volume (DPV), denoted as $p(d; u, v)$. This is contrary to numerical depth calculations, where each depth estimate in the depth map is based on the disparity-depth duality principle fundamental to stereo depth, Sec.2.4.1. The authors input a video stream to a 3-layer-deep neural network architecture. The first layer, the D-Net, estimates the DPV for each input frame. The second layer, the K-Net, integrates the DPVs over time, and the third layer is the R-Net, which refines the spatial resolution of the DPVs.
The first layer computes the DPV for each frame $\boldsymbol{I_t}$ by comparing it to its temporally neighbouring frames, such that $N_t = [t - 2\Delta t, t - \Delta t, t, t + \Delta t, t + 2\Delta t]$ with $\Delta t = 5$, resulting in 5 frames per $\boldsymbol{I_t}$. The features of $\boldsymbol{I_t}$ and each temporally neighbouring frame $\boldsymbol{I_k}$ with $k \in N_t$ are extracted, and the latter are warped onto the features of $\boldsymbol{I_t}$ before the sum of the absolute differences are computed,

$$L(d_t | I_t) = \sum_{k \in N_t, k \neq t} \| f(\boldsymbol{I_t}) - \text{warp}(f(\boldsymbol{I_k}); d_t, \delta \boldsymbol{T}_{kt}) \| \tag{2.31}$$

Where $\delta \boldsymbol{T}_{kt}$ is the relative pose transformation from $\boldsymbol{I_k}$ to $\boldsymbol{I_t}$, f() is a feature extractor, and warp$\dot{()}$ defines a function that warps the features of $\boldsymbol{I_k}$ to $\boldsymbol{I_t}$. The result $L(d_t|\boldsymbol{I_t})$ is input into a softmax function, mapping the outputs to probabilities, finally yielding the DPV,

$$p(d_t|\boldsymbol{I_t}) = \text{softmax}(L(d_t|\boldsymbol{I_t})). \tag{2.32}$$

The K-Net then intigrates frames over time by defining a *belief volume* $p(d_t, \boldsymbol{I}_{1:t})$. A Bayesian filter is applied to first predict the next DPV by warping the current DPV at camera coordinate t to the volume at $t+1$, yielding $p(d_{t+1}|\boldsymbol{I}_{1:t})$. Then, computing the DPV of $\boldsymbol{I}_{t+1}$ with the D-Net, and finally updating the belief volume at $t+1$,

$$p(d_{t+1}|\boldsymbol{I}_{1:t+1}) = p(d_{t+1}|\boldsymbol{I}_{1:t}) \cdot p(d_{t+1}|\boldsymbol{I}_{t+1}). \tag{2.33}$$

This integration of previous DPVs into current DPV estimates reduces uncertainty over time. The final layer upsamples the output feature map back to its original size, initially downsampled to 1/4th of its size by the feature extractor in the D-Net.

2.4.4 Depth Map Post-Processing

Post-processing techniques have been shown to enhance an initial disparity map computation in [23]. A cleaner depth map can be achieved by, e.g. grouping larger regions of approximately the same disparity value together or reducing noise induced by, e.g. erroneous matches or sharp changes of intensity, which are often hard for the algorithm to deal with. Below, we present 3 methods that we have explored in this work.

Median Filter

A median filter replaces the current intensity value of a pixel (x, y) with the median value of a specified kernel, effectively smoothing out any drastic intensity fluctuations. Our implemented median filter is defined as [24],

$$\boldsymbol{I}'[x, y] = \text{median}\{\boldsymbol{K}[m, n] \times \boldsymbol{I}[x - (n - \lfloor k_w/2 \rfloor), y - (m - \lfloor k_h/2 \rfloor]\} \tag{2.34}$$

where $\boldsymbol{I}$ and $\boldsymbol{I}'$ are the input and output images, respectively. $\boldsymbol{K}$ is the neighbourhood kernel, where $\boldsymbol{K} \in \{0 \vee 1\}$. k_w and k_h are the kernels width and height, respectively, and *median* is the median filter operation.
The median is selected from the kernel neighbourhoods vertical and horizontal lines, intersecting at the pixel coordinate, sorting all these intensities from low to high, and then selecting the median value as the new intensity value of the pixel at (x, y).

Bilateral Filter

In [23], the use of a bilateral filter for depth-map post-processing was reported to increase the perceived quality of depth maps. A bilateral filter performs local smoothing of image intensity while preserving edges. It is defined as [24],

$$\boldsymbol{I}'(\boldsymbol{p}) = \frac{1}{W_p} \sum_{\boldsymbol{q} \in \Omega} \boldsymbol{I}(\boldsymbol{p}) k_r(\|\boldsymbol{I}(\boldsymbol{q}) - \boldsymbol{I}(\boldsymbol{p})\|) k_s(\|\boldsymbol{p} - \boldsymbol{q}\|) \tag{2.35}$$

and the normalization term is defined as,

$$W_p = \sum_{q \in \Omega} k_r(\|\boldsymbol{I}(\boldsymbol{q}) - \boldsymbol{I}(\boldsymbol{p})\|) k_s(\|\boldsymbol{p} - \boldsymbol{q}\|) \tag{2.36}$$

where I and I' are the input and output images, respectively, and k_r and k_s are the range and space kernels, which are defined as the normalized Gaussian functions,

$$k_r(p) = e^{-\frac{\|p\|^2}{2\sigma_r^2}} \tag{2.37}$$

$$k_r(p) = e^{-\frac{\|p\|^2}{2\sigma_s^2}} \tag{2.38}$$

σ_r influences how the intensity range of the input is smoothed out. Higher values will lead to larger intensity regions being smoothed. σ_s, on the other hand, is the smoothing factor; larger values will result in more aggressive smoothing.

Speckle Filter

Speckle noise results from textureless regions in the image, where the algorithm can't find reliable matches due to the absence of reliable feature detection. The result is sharp changes in intensity that don't agree with the neighbouring intensity values. The noise can be reduced by defining a window size in which the disparity values cannot vary by a defined number of disparity values. The result is a disparity map with more significant regions of approximately the same disparity values, resulting in a smoother 3D reconstruction [25].

2.5 3D Reconstruction

2.5.1 Structure from Motion

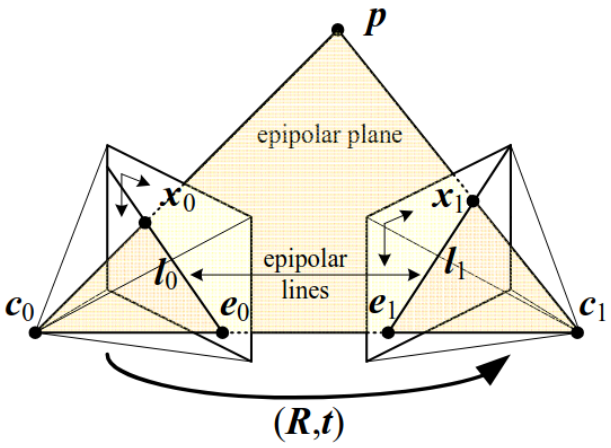


Figure 2.7: SfM Epipolar Geometry [19].

Structure from motion is the process of calculating depth and 3D models from multiple images of the same scene taken from different viewpoints without prior knowledge of the relative positions of the viewpoints. In the case of 2 image frames, the shift from one viewpoint to another is encoded in a rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$.
If we define the observed 3D point in image 0 to be $\boldsymbol{p}_0 = d_0\hat{\boldsymbol{x}}_0$, observed at position $\hat{\boldsymbol{x}}_0$ in image 0 and situated at a distance d_0, then we map $\boldsymbol{p}_0$ onto image 1 with the transformation [19],

$$d_1\hat{\boldsymbol{x}}_1 = \boldsymbol{p}_1 = \boldsymbol{R}\boldsymbol{p}_0 + \boldsymbol{t} = \boldsymbol{R}(d_0\hat{\boldsymbol{x}}_\mathbf{0}) + \boldsymbol{t} \tag{2.39}$$

We eliminate $\boldsymbol{t}$ by applying the cross product $[\boldsymbol{t}]\times$, and then apply the dot product of $\hat{\boldsymbol{x}}_\mathbf{0}$, which yields,

$$d_0\hat{\boldsymbol{x}}_1^T([\boldsymbol{t}] \times \boldsymbol{R})\hat{\boldsymbol{x}}_0 = 0 \tag{2.40}$$

.
Notice that we arrive at the basic epipolar constraint, similar to Sec. 2.4.1,

$$\hat{\boldsymbol{x}}_1^T\boldsymbol{E}\hat{\boldsymbol{x}}_0 = 0. \tag{2.41}$$

Note that in this case, the absolute distance between the two frames can never be recovered, and therefore the absolute depth of points in the scene can not be calculated, as opposed to Sec. 2.4.1. For this, actual measurements of the camera or point positions, also called *ground control points*, would have to be known. In the case of stereo depth estimation, calibrating the stereo camera with a calibration target yields the actual translation vector $\boldsymbol{t}$ and, therefore, the baseline. In the case of SfM, there is inherent scale ambiguity. The result of SfM is a point cloud, discussed in Sec. 2.5.3.
For underwater applications, refraction at the dome port interface causes geometric inaccuracies when computing the disparity between feature matches. A more accurate implementation of Structure-from-Motion for the underwater environment is discussed in Sec. 2.5.2.

2.5.2 Refractive Structure-from-Motion

The basic idea of refractive Structure-from-Motion is the computation of virtual cameras given the dome port offset parameters $\boldsymbol{v}_{\text{off}}$ or $\boldsymbol{C}_d = (c_x, c_y, c_z)^T$, discussed in Sec. 2.3.4, as well as the inner radius and thickness of the dome port. Then, the same approach as in Sec. 2.5.1 is applied for the virtual cameras, resulting in a virtual epipolar constraint, according to which triangulation is performed [5]. In theory, this approach eliminates the error due to refraction, discussed in Sec. 1.2.2.
We start by first finding the in-water viewing ray $\boldsymbol{v}^w$ of a feature observed in an underwater image by backprojecting the corresponding 2D point to 3D space according to Sec. 2.3.3. Once the vector $\boldsymbol{v}^w$ is found, it is extended backwards until it intersects the axis of refraction $\boldsymbol{a}$, discussed in Sec. 2.3.4. A virtual pinhole camera is then placed at this intersection and can be described by a transformation from the real to the virtual coordinate frame ${}^v\boldsymbol{T}_r = ({}^v\boldsymbol{R}_r | \, {}^v\boldsymbol{t}_r)$. The rotation of the virtual camera is defined as the identity matrix ${}^v\boldsymbol{R}_r = \boldsymbol{I}$, and the mean focal length of the real camera is taken as the virtual focal length $f_v = f_{mean}$. In addition, the virtual principal point (c_{vx}, c_{vy}) is defined in a way such that the original observed image point $\boldsymbol{x} = (x, y)^T$ remains the same:

$$c_{vx} = x - f_v \cdot \bar{\boldsymbol{v}}^w_{hnorm}(x), c_{vy} = y - f_v \cdot \bar{\boldsymbol{v}}^w_{hnorm}(y) \tag{2.42}$$

$\bar{\boldsymbol{v}}^w_{hnorm}(x)$ and $\bar{\boldsymbol{v}}^w_{hnorm}(y)$ represent the x- and y-component of the homogeneous-normalized ray in water $\bar{\boldsymbol{v}}^{\boldsymbol{w}}$. The virtual camera center ${}^r\boldsymbol{t}_v$ can be found by intersecting $\bar{\boldsymbol{v}}^w$ with the axis of refraction $\boldsymbol{a}$.
The relative pose between two cameras is found similarly to Sec. 2.5.1, but this time, the observed points are back-projected to 3D space according to the model in Sec. 2.3.3, and the re-projection error is minimized to find the relative pose of the real camera. We then continue by optimizing the relative pose between virtual cameras by defining a refractive virtual epipolar cost and minimizing it regarding the relative pose transformations.
Similar to Sec. 2.5.1 and Sec. 2.4.1, an epipolar plain is formed by the refracted

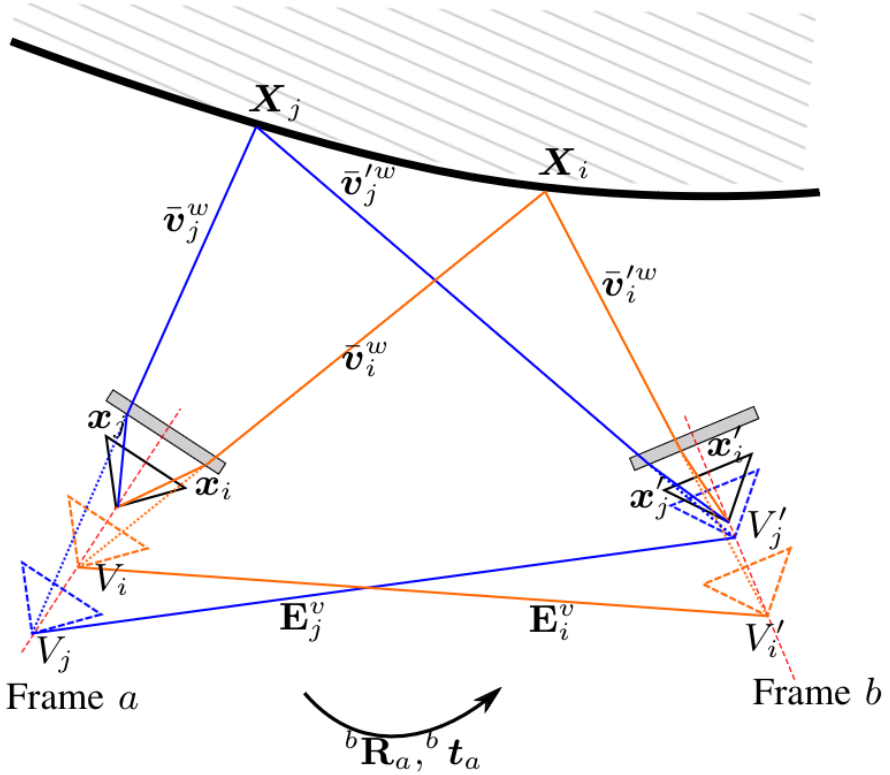


Figure 2.8: A schematic illustration of the virtual epipolar geometry [5].

rays $\bar{\boldsymbol{v}}^w$ and $\bar{\boldsymbol{v}}'^w$, corresponding to a given feature point in the scene, and the vector connecting the two virtual camera centres V_j and V_j', as seen in Fig. 2.8. The relative pose of the real cameras are known due to estimating the fundamental matrix $\boldsymbol{E}$ after minimizing the reprojection error of the feature points. If we can now find the rotation and translation information for transforming a real camera to a virtual camera at the intersection of the refracted rays with the axis of refraction and project feature points onto the virtual camera plane, then the same procedure as in Sec. 2.5.1 can be used to estimate depth with higher geometrical accuracy.
If we express the relative pose transformation from real camera a to real camera b as ${}^b\boldsymbol{T}_a = \left({}^b\boldsymbol{R}_a | {}^b\boldsymbol{t}_a\right)$ and a feature point $\boldsymbol{x}_i$ in image a is matched to the feature point $\boldsymbol{x}_i'$ in image b, then the transformation from the real camera to the virtual one at feature points $\boldsymbol{x}_i$ and $\boldsymbol{x}_i'$ are ${}^{v_i}\boldsymbol{T}_r$ and ${}^{v_i'}\boldsymbol{T}_r$, respectively. Next, the transformation from the virtual camera to its corresponding virtual camera in frame b is concatenated as:

$$ {}^{v_i'}\boldsymbol{T}_{v_i} = \left({}^{v_i'}\boldsymbol{R}_{v_i} | {}^{v_i'}\boldsymbol{t}_{v_i}\right) = {}^{v_i'}\boldsymbol{T}_r \, {}^b\boldsymbol{T}_a ({}^{v_i}\boldsymbol{T}_r)^{-1} \tag{2.43} $$

Then, for each feature point pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_i'$, we have an epipolar constraint:

$$ \hat{\boldsymbol{x}}_i'^T \boldsymbol{E}_i^v \hat{\boldsymbol{x}}_i = 0 \text{ where } \boldsymbol{E}_i^v = \left[{}^{v_i'}\boldsymbol{t}_{v_i}\right] \times {}^{v_i'}\boldsymbol{R}_{v_i} \tag{2.44} $$

This epipolar constraint expresses the same idea as in Sec. 2.4.1 and Sec. 2.5.1, only now the epipolar lines are determined, on which both matched feature points, detected in-water, lie as if the dome port interface had not influenced their viewing ray. This epipolar constraint is used in Sec. 2.4.1 and 2.5.1 to compute the disparity between matched points in 2 corresponding frames. In theory, the output of refractive Structure-from-Motion is a point cloud that more accurately represents the spatial relationships of the points. Note that the absolute position of the points in relation to the camera cannot be determined due to the lack of ground control points.

2.5.3 Point Clouds

A point cloud is one of the most simple 3-dimensional representations of a scene. Given the camera intrinsic parameters yielded by Sec. 3.1.1 and the depth map discussed in Sec. 2.4.1, each point of the depth map can be back-projected out into 3D space.
Given a depth map D with size n $\times$ m, we first initialize a 2-dimensional mesh grid with the exact dimensions as the input depth map, illustrated by Fig. 2.9.

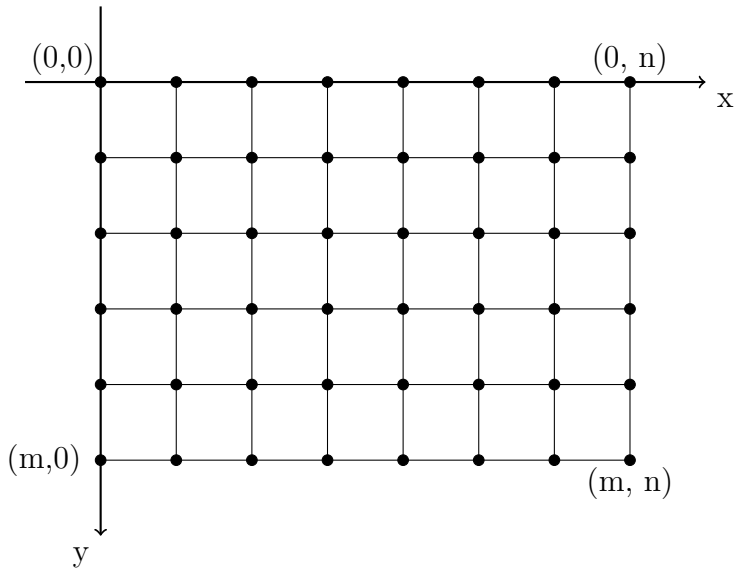


Figure 2.9: A 2-dimensional mesh grid.

Note that we define the origin of the mesh grid in the upper left corner and interpret increasing y-values as moving down the mesh grid from top to bottom, which is common in computer vision since an image is interpreted as a matrix.
For each entry of the mesh grid (u, v), we compute a corresponding 3-dimensional coordinate using the intrinsic camera parameters and the depth map. The x and y coordinates are projected into 3D space according to:

$$x = \frac{(u - c_x)}{f_x} * D(u, v) \tag{2.45}$$

$$y = \frac{(u - c_y)}{f_y} * D(u, v) \tag{2.46}$$

and the z-coordinate is assigned the depth value at pixel (u, v) in the depth map,

$$z = D(u, v) \tag{2.47}$$

If we now lay one of the rectified images, used to compute the depth map, over the mesh grid, we can assign each 3-dimensional point an RGB value from that image, resulting in a coloured 3-dimensional representation of the photographed scene. The result is referred to as a coloured point cloud. This representation is helpful in validating the core 3D reconstruction functionality, and it can be further processed to compute surface reconstructions such as a mesh or volumetric reconstructions such as voxels. A more common method for volumetric reconstruction based on voxel grids is discussed in the following section, Sec. 2.5.4.

2.5.4 Voxel Grids

The advantage of voxel grids, as opposed to point cloud representations of a 3D scene, is that each point in 3D space is approximated as a voxel, the 3-dimensional equivalent of a pixel, and assigned a depth value. Depth information about each point in 3D space (x, y, z) can then be easily looked up via their index. Occupancy grids improve on this idea by classifying each voxel in space as either *occupied* or *free*, meaning that a particular voxel is either outside of a 3D object if it is *free* and situated within the object if it is *occupied*. The result is a volumetric representation of the scene, which can give us a better idea of an object's geometric properties. Voxel grids are also often used in real-time 3D reconstruction applications since the underlying data structure can be continuously updated as the scene changes, e.g. freeing up occupied voxels or assigning a depth value to a newly occupied voxel when objects in the scene move or the position of the camera is shifted. It is important to note that in this approach, the voxel grid is static, while depth information is updated as the scene or camera changes.
Given a depth map and a corresponding rectified RGB image, the idea is to project each voxel in view within the scene to a depth value on the depth map with the following function,

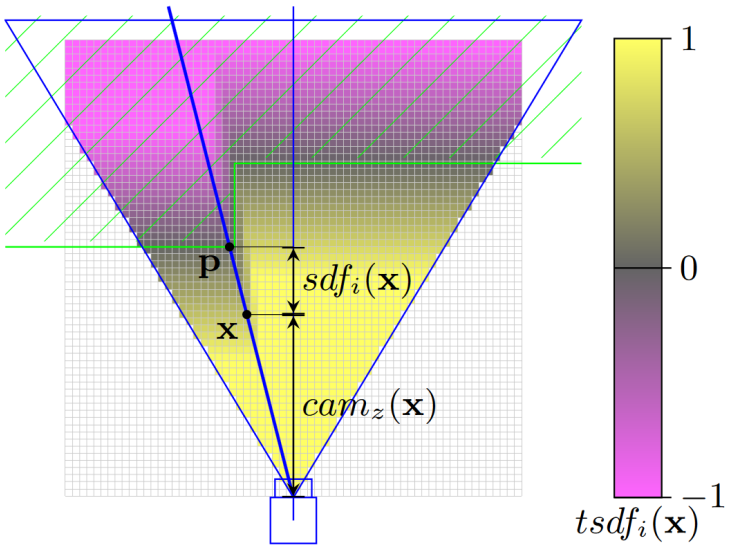
$$d = \mathcal{D}\left[\pi_{camera}\left(\boldsymbol{T}_{CL}\boldsymbol{p}_{\boldsymbol{L}}\right)\right] \tag{2.48}$$

whre $\boldsymbol{T}_{\boldsymbol{CL}}$ denotes the cameras pose and $\boldsymbol{p}_{\boldsymbol{L}}$ denotes the voxel center position. For a given 3-dimensional point in space $\boldsymbol{p}_C$ the camera projection function, mapping a 3D to a 2D point, is defined as,

$$\pi_{camera}(\boldsymbol{p}_C) = \frac{1}{p_{C,z}} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix} \begin{bmatrix} p_{C,x} \\ p_{C,y} \\ 1 \end{bmatrix} \tag{2.49}$$

where f_x, f_y, c_x and c_y are the pinhole camera intrinsic parameters we recieved from Sec. 3.1.1. This means for an observed 2D point of the voxel centre in the grid coordinate frame $\boldsymbol{P}_L$ we project this point to the camera coordinate frame $\boldsymbol{P}_C$ and then finally sample the depth to the nearest 3D object in space $\boldsymbol{p}$ with the function $\mathcal{D}[\boldsymbol{u}]$. We, therefore, assign that voxel a distance value to the nearest 3D object point $\boldsymbol{p}$, which intersects the viewing ray through the voxel centre $\boldsymbol{p}_L$ and the observed 2D point on the image plane $\boldsymbol{P}_C$.
We achieve this via the Truncated Signed Distance Function (TSDF), which is a simplified version of the Signed Distance Function (SDF), which assigns all points in 3D space a distance relative to a 3D object. For the i-th observation of a scene, the SDF is defined as [26]:

$$sdf_i(\boldsymbol{P}_L) = \mathcal{D}\left[\pi_{camera}\left(\boldsymbol{T}_{CL}\boldsymbol{p_L}\right)\right] - cam_z(\boldsymbol{P}_L) \tag{2.50}$$



(a)

Figure 2.10: The 3D object in space is denoted in green. The truncated distance values are denoted in colour from pink to yellow. In this case an exemplary truncation distance t is set to 1 [26].

where $\pi_{camera}\left(\boldsymbol{T}_{CL}\boldsymbol{p_L}\right)$ is the projection of the voxel center on to the depth image, yielding the 2D depth map coordinate $\boldsymbol{u}$ from which the depth to the intersection with the 3D object along the viewing ray extending from $\boldsymbol{u}$ through $\boldsymbol{p}_L$ is extracted

with $\mathcal{D}$. The voxel centre distance from the camera $cam_z(\boldsymbol{p}_L)$ is then subtracted to finally yield the signed distance of the voxel $\boldsymbol{x}$ to the intersection point $\boldsymbol{p}$ on the 3D object in space. An overview of the geometry can be seen in Fig. 2.10.
For the truncated signed distance function, the sdf_i, are cut off after a certain distance threshold, since not all distances are required for our reconstruction, reducing the memory footprint. For the TSDF the truncation parameter must be set as t. The truncated version of the sdf_i for the i-th observation of a scene is defined as $tsdf_i$ [26]:

$$tsdf_i(\boldsymbol{P}_L) = \max\left(-1, \min\left(1, \frac{sdf_i(\boldsymbol{P}_L)}{t}\right)\right) \tag{2.51}$$

The coloured regions in Fig. 2.10 indicate the values within the truncation threshold. After calculating the $tsdf_i$ value, the TSDF function computes a weight for a given tsdf value with a weight function $w = f_w(tsdf_i)$, which are used to combine TSDF distance values with a weighted average.
The TSDF only assigns distance values to voxels within a small truncation distance from the object's surface. The Euclidean Signed Distance Function improves on this by assigning each voxel in the grid a distance value to the objects, even if the voxels are far away from potential obstacles.
Volumetric occupancy grids can efficiently be implemented for GPU acceleration. The grid is divided into larger chunks of voxels, e.g. an 8x8x8 block, and each chunk is stored within GPU memory; each block is then continuously updated in parallel by the GPU, freeing up voxels that are no longer occupied, assigning depth values to newly observed voxels and updating distances as the camera or objects move through the scene. An adaption of [27] featuring GPU acceleration is implemented in [26], reporting real-time performance.

2.6 Implementation

2.6.1 In-Air Experimental Setup

Calibration

For in-air calibration, we use a 7 x 4 grid checkerboard calibration pattern with a 53 mm square size, which is sufficiently large to calibrate our camera at our target distance of 2-3 meters. At least 20 stereo pairs of the calibration target are recorded with the Sea3D, without dome ports, and with dome ports, while making sure the calibration dataset covers the whole camera frame and is visible in both left and right frames at all times. If the target is partially non-visible within one of the frames, the stereo pair cannot be calibrated. The frames are then read by the *stereoCameraCalibrator* app in Matlab, which allows us to remove calibration images with high reprojection errors, bringing down the overall mean reprojection error of the calibration parameters. The tool yields the intrinsic camera parameters and lens distortion coefficients, discussed in Sec. 3.1.1, which can easily be exported to OpenCV-compatible parameters with a single line of code. In OpenCV, the images are remapped to their rectified formats and are passed to the disparity estimation. We also calibrated our system with a larger calibration pattern with dimensions 841 mm $\times$ 1189 mm, a 9 $\times$ 10 checkerboard grid and a 95 mm square size, with the aim of achieving a lower reprojection error at further distances.

Example Scene

An example scene was set up in Fig. 2.11 to compare the 3D reconstruction performance of Sea3D and the commercially available OAK-D stereo camera from Luxonis, as seen in Fig. 9. An overview of the OAK-D's specifications can be seen in Tab. 2.3.

(a) In-air RGB scene from Sea3D. (b) In-air RGB scene from the OAK-D.

Figure 2.11: In-air stereo example scene.

We take a stereo image pair with the Oak-D and a stereo pair of the exact same scene with Sea3D and compare the depth map and point cloud results in Sec. 3.2. The chosen objects in the scene have clear boundaries and have been placed at a

distance from each other such that the variance in the resulting depth estimation will be identifiable for the object.
After the in-air calibration is performed, we use the intrinsic parameters for in-situ calibration according to Sec. 2.3.4 by using the CALIBMAR tool [28].

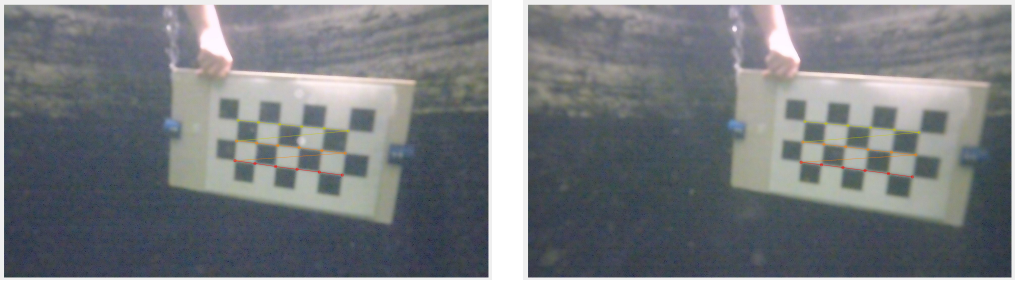
Specification	**Value**
Sensor	OV9282
Optical Format(inch)	1/4
Resolution (pixels)	1280×800
Max Frame Rate (fps)	120
Aperture (f-stop)	2
Vertical Field of View (°)	55
Horizontal Field of View (°)	80
Baseline (mm)	75

Table 2.3: OAK-D Specifications.

2.6.2 In-Situ Experimental Setup

Calibration

Two calibration approaches are explored for in-situ calibration. The first approach is to apply the same calibration and rectification procedure as in the in-air case discussed in Sec. 3.1.1 and Sec. 2.4.1, but submerging the entire system along with the same 7 x 4 grid checkerboard target as in Sec. 2.6.1 underwater. Again, at least 20 images are recorded with Sea3D, ensuring enough variation in the angle and position of the calibration target, such that the dataset covers the entire camera frame. An image of a calibration pair can be seen in Fig. 2.12.



(a) Left image. (b) Right image.

Figure 2.12: Calibration with a 7×4 checkerboard grid.

The second in-situ calibration approach aims to determine the camera dome port offset to account for the error induced by refraction, as discussed in Sec. 2.3.4. For this approach, first, the dome ports on Sea3D are removed from the camera setup, and the in-air intrinsic parameters of the camera are determined by following the

procedure in Sec. 2.6.1. Then, the dome ports are mounted back on the camera setup, and the system is submerged underwater and aimed at the same calibration target as in Sec. 2.6.1. Again, more than 20 images are recommended for accurate calibration, and the target must be recorded at varying angles and orientations.
We have found the CALIBMAR tool [28] to be more robust in calibrating the camera with underwater data for the stereo calibration step than Matlab's *stereoCameraCalibrator* tool, since the checkerboard is more often detected with CALIBMAR. Again, the calibration result is the camera's intrinsic and extrinsic parameters, which can be used for rectification, but this time, the parameters are determined with the dome port interfaces mounted on Sea3D.
CALIBMAR provides in-built functionality for calibrating the dome port offset of the camera, yielding the offset of the cameras from the dome port centre $\boldsymbol{v}_{\text{off}}$ discussed in Sec. 2.3.3 and in Sec. 2.3.4. Along with the camera intrinsic parameters, CALIBMAR assumes that the indices of refraction for air, water and glass, as well as the inner radius and the thickness of the dome port, are known [29], as seen in Tab. 2.4 and Tab. 2.5. If we know these parameters, all we need to do is take approximately 20 images of the same checkerboard target to calibrate the dome port offset $\boldsymbol{v}_{\text{off}}$.

Medium	Air	Water	Acrylic
Index of Refraction	1.000292	1.3330	1.492

Table 2.4: Indices of Refraction for Air, Water and Acrylic Glass

Dome Port Specification	Dome Port Inner Radius	Dome Port Thickness
Value (mm)	31.25	3.1

Table 2.5: Dome Port Inner Radius and Dome Port Thickness

Lab Tests

Both the processor payload and camera payload are mounted on the BlueRobotics BlueROV2 underwater drone, as shown in Fig. 2.13. Before each test, all compartments of Sea3D and the ROV are vacuum-sealed and pressure-tested. The pressure inside the enclosures is reduced to approximately 15 inHg, and the setup is monitored to ensure the pressure does not rise by more than 0.5 inHg over a 10-minute period. Once verified, the ROV is submerged in a cylindrical water tank with a diameter of approximately 1.23 m and a height of 1.25 m. Due to poor lighting conditions in the tank, an onboard LED was used to enhance visibility.
An ArucoMarker Cube and plastic bottle are placed on the bottom of the tank as a reference for evaluating the 3D reconstruction performance. These objects are photographed from varying angles.

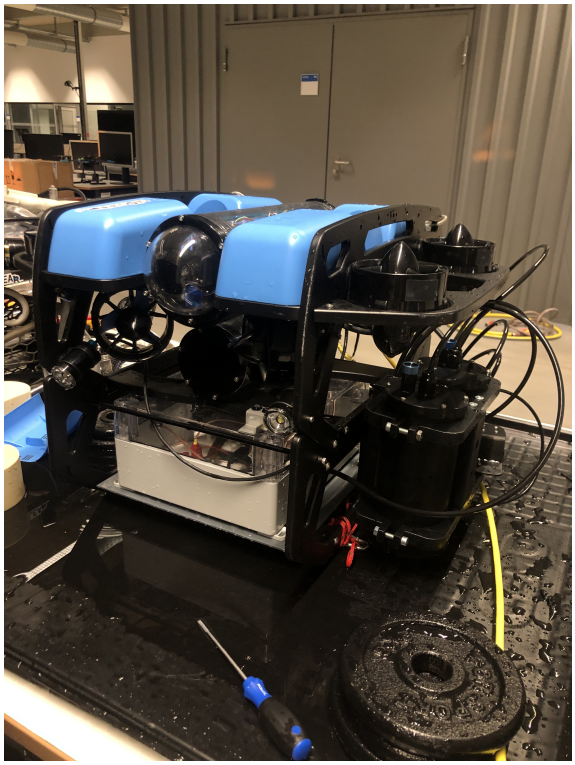


Figure 2.13: Experimental Setup.

Field Test

A field test was conducted on 1st August 2024 at the Hollerner See, about 23km north of Munich. The exact same setup as in the lab was transported to the lake and the ROV was steered around the lake, capturing about 1h of video footage of underwater vegetation and the reference objects mentioned previously. The data was evaluated offline and is presented in Sec. 3.

2.6.3 Data Processing

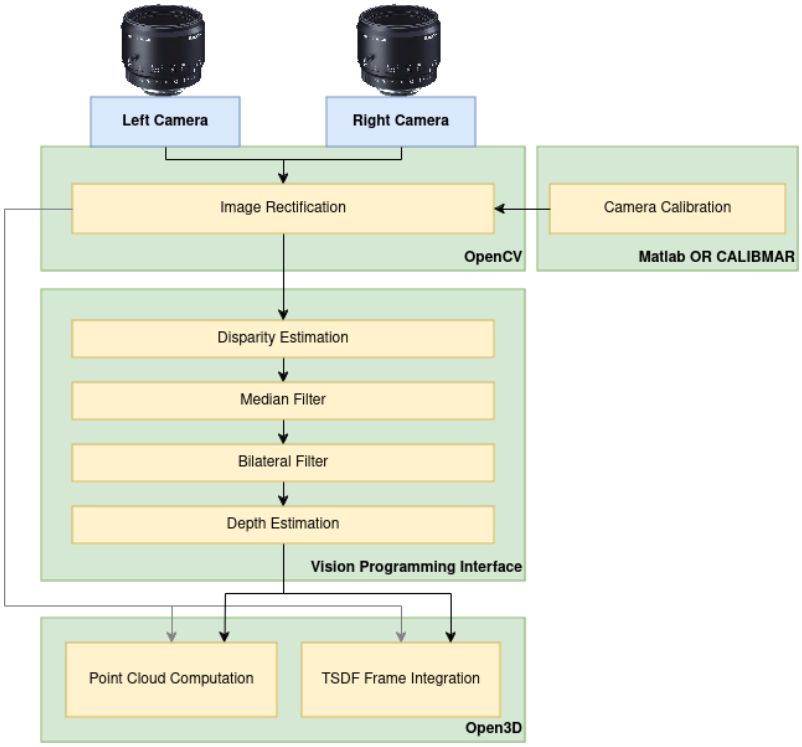


Figure 2.14: Post Processing Overview.

During data processing, the steps outlined in Sec. 2.3, Sec. 2.4, and Sec. 2.5 are combined to produce the final outputs: point clouds and integrated TSDF models. An overview of all the stages involved is provided in Fig. 2.14. The pipeline integrates camera calibration using Matlab or CALIBMAR, stereo rectification, disparity estimation, filtering, and depth estimation through OpenCV and VPI, followed by point cloud generation and integration using Open3D, an open-source framework for 3D data processing. The code for the data processing pipeline can be found at [30].

Chapter 3

Experiments

3.1 Calibration

3.1.1 In-Air Camera Calibration

Our in-air calibration for 2-3 metres yields a re-projection error of 0.29 pixels with the 7×4 checkerboard grid calibration target. A slightly better overall mean error of 0.25 pixels can be achieved with a larger calibration target, covering approximately 20% of the image frame, as seen in Fig. 5.

Figure 3.1: Rectified In-air Stereo Pair with a Selection of Epipolar Lines.

We can visually assess the accuracy of our calibration by overlaying horizontal lines across both images, displayed side by side. By identifying a distinct feature point along a line in the left image, we can follow the corresponding line in the right image to verify if it intersects the same feature point. If the feature point remains aligned across both images without significant deviation, we can conclude that the rectification has been successful.
Looking at the rectified pair of our example scene in Fig. 3.1, we can observe, e.g. that the tip of the drill is well aligned along the same line in both images. We can do this for points closer to the edges of the frames, such as the upper left corner of the left-most window or the bottom of the blue cylindrical container, verifying that the rectification is accurate across the frame.

The corresponding intrinsic and extrinsic camera parameters used to compute the rectified images can be seen in Tab. 3.1 and Tab. 3.2, respectively.

Intrinsic Parameters	f_x	f_y	c_x	c_y	Overall RMS
Left Camera	1082.4	1074.4	949.5	535.7	0.29
Right Camera	1076.8	1069.1	951.2	545.7	0.29

Table 3.1: In-air Intrinsic Camera Parameters.

Rotation Matrix (R)			**Translation Vector (t)**
0.9989	0.0476	0.0041	107.9218
−0.0476	0.9988	0.0076	−0.7328
−0.0038	−0.0078	1.0000	0.9580

Table 3.2: In-air Extrinsic Camera Parameters.

3.1.2 In-Water Camera Calibration

Repeating the same calibration procedure underwater, as described in Sec. 3.1.1, results in an overall mean re-projection error of 0.6 pixels, and the rectified image pair shown in Fig. 3.2. As before, features such as the bottle cap remain aligned along the same epipolar line in both images. However, there is a slight deviation when validating the calibration near the edges of the frame. For instance, when tracking a bright patch of sand on the tank floor across the epipolar line to the other image, the feature shifts slightly out of alignment. The intrinsic camera parameters from this calibration procedure can be seen in Tab. 3.3 and in Tab. 3.4.

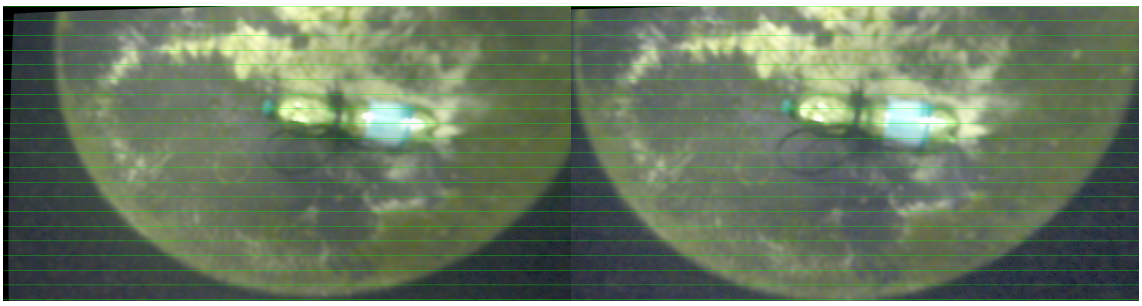


Figure 3.2: Rectified in-situ stereo pair with a selection of epipolar lines.

Intrinsic Parameters	f_x	f_y	c_x	c_y	Overall RMS
Left Camera	1097.86	1089.78	921.038	520.44	0.60654
Right Camera	1118.34	1109.4	925.302	525.286	0.558961

Table 3.3: In-situ Camera Intrinsic Parameters.

Rotation Matrix (R)			**Translation Vector (t)**
0.999984	0.00482615	0.00281708	−109.27
−0.00481898	0.999985	−0.00254567	2.69592
−0.00282933	0.00253206	0.999993	−9.35285

Table 3.4: In-situ Extrinsic Parameters.

3.1.3 Dome Port Offset Calibration

The in-air calibration parameters, calibrated in the field, are found in Tab. 3.5. The dome port offset calibration yields the camera dome port offset parameters seen in Tab. 3.7.

Intrinsic Parameters	$\boldsymbol{f}_x$	$\boldsymbol{f}_y$	$\boldsymbol{c}_x$	$\boldsymbol{c}_y$
Left Camera	1140.73	1140.15	961.94	548.07
Right Camera	1125.75	1125.34	956.04	540.35

Table 3.5: In-air Parameters for Dome Port Calibration.

Rotation Matrix (R)			**Translation Vector (t)**
0.999311	0.036385	0.007214	107.839
−0.036496	0.999207	0.015868	0.927
−0.006631	−0.016120	0.999848	4.331

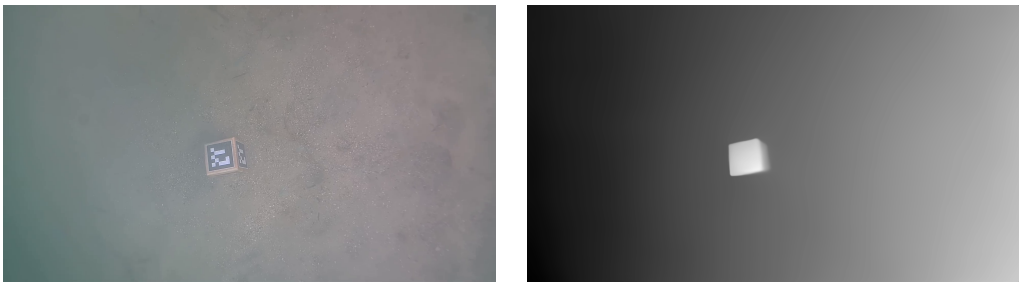
Table 3.6: Dome Port Calibration Extrinsic Parameters

Offset Direction	δX_{dome}	δY_{dome}	δZ_{dome}
Offset Magnitude	-0.536468	0.582285	2.09815

Table 3.7: Camera Dome Port Offset Prameters

3.2 Depth Estimation

3.2.1 Monocular Depth



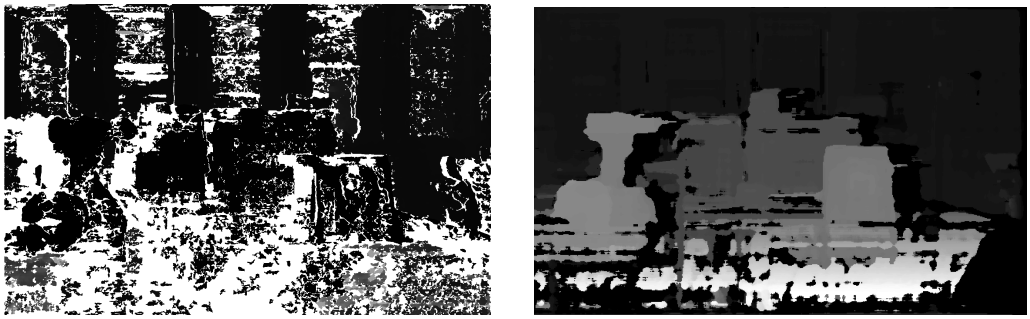
(a) RGB Field Test Image. (b) Monocular Depth Map.

Figure 3.3: DNN Monocular Depth Map Estimation Result.

The monocular DNN depth estimation in Fig. 3.3(b) yields a visually pleasing result, showing smooth, clearly defined edges of the ArucoCube. The downward slope of the seafloor is also nicely represented as a smooth transition of depth from the right to the left edge of the frame. The overall scene is nicely reflected by the depth map, not showing any significant issues resulting from the effects present underwater, such as the reduced visibility conditions due to haze or the blurring effect at the frame edges due to the absorption of light, discussed in Sec. 1.2, and seen in Fig. 3.3(a). We must remember that the depth map does not present geometrically accurate information,i.e. the actual distance of the ArucoCube from the camera, but instead infers a notion of depth from the change in visual information across multiple time-varying frames. While the result in Fig. 3.3 is visually appealing, its effectiveness is overshadowed by the absence of concrete geometrical information. The aesthetic quality cannot compensate for the lack of precise depth and structure, which limits its utility for applications requiring accurate spatial understanding.

3.2.2 In-Air Stereo Depth

When comparing the Sea3D SGM depth map in Fig. 3.4(a) with the depth map from the OAK-D camera in Fig. 3.4(b), we see that the OAK-D depth map shows more clearly segmented regions with equal disparity, whereas the Sea3D depth map captures finer details on the surface of objects, e.g. when comparing the depth estimates in the region of the mouse, seen on the right side of the table in Fig. 2.11, we notice that Sea3D captures the contours of the mouse within the plastic packaging, whereas the OAK-D camera groups the whole mouse and plastic packaging into an almost uniform depth region. The same goes for the scene's background, with the wall and windows being grouped into a uniform region by the OAK-D and Sea3D, estimating varying disparities in smaller regions.

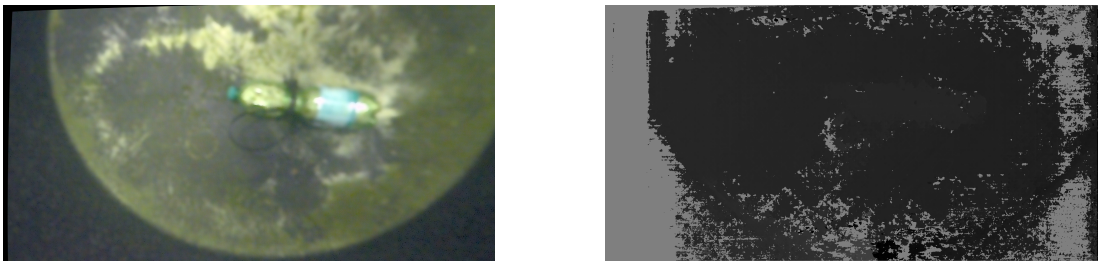


(a) In-air Stereo Depth from Sea3D. (b) In-air Stereo Depth from OAK-D.

Figure 3.4: Monocular Depth Map Estimation Result.

Another clearly noticeable feature is the handling of textureless regions by the OAK-D and Sea3D. The OAK-D appears to perform much better with regards to the processing of the table surface in the foreground of Fig. 2.11, whereas this region introduces a particularly large amount of noise in the Sea3D depth map. In general, the Sea3D depth map displays a significantly larger amount of noise and uncertain depth regions than the OAK-D, even after increasing the maximum disparity value considered by the disparity estimation algorithm for the disparity map from 0 to 1, effectively removing a tremendous amount of noise in the depth maps, by removing the depth estimated to lie at infinity. The OAK-D depth map more clearly retains the overall structure of the scene compared to Sea3D; the edges objects are cleaner and more accurate, and even thin objects such as the drill bit are grouped into the larger depth region of the electric screwdriver.

3.2.3 In-Situ Stereo Depth



(a) Rectified Left Image of the In-situ Scene. (b) In-situ Disparity Map from Stereo.

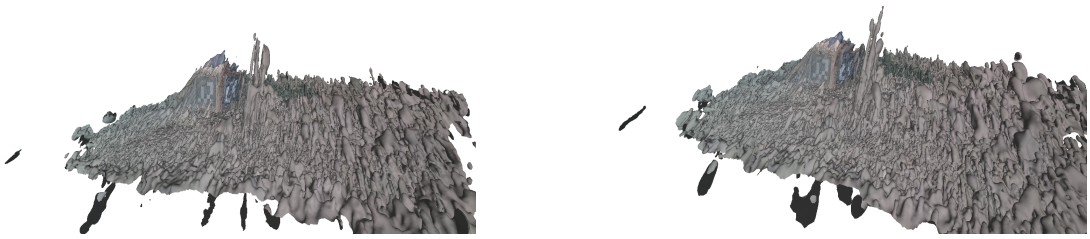
Figure 3.5: Monocular Depth Map Estimation Result.

Taking a look at the disparity map for the underwater scene in Fig. 3.5(b) and comparing it to the RGB image in Fig. 3.5(a), we see that the disparity around the region of the bottle clearly indicates an elevated region of the image. Still, the edges

of this region are rough, and noise is introduced, particularly on the bottom edge of the bottle, where the algorithm seems to struggle to estimate the depth of the black cable binder.
Most noticeable is the introduction of a significant amount of noise around the edges of the frame, while the middle of the frame generally seems to deliver more accurate depth estimates. Around the middle of the frame, the flat surface of the bottom of the tank appears to be accurately estimated, with this larger region being clearly separable from the bottle region.

3.3 3D Reconstruction

3.3.1 Monocular Structure from Motion



(a) SfM without Dome Port Offset Parameters. (b) SfM with Dome Port Parameters.

Figure 3.6: SfM 3D Reconstruction Result.

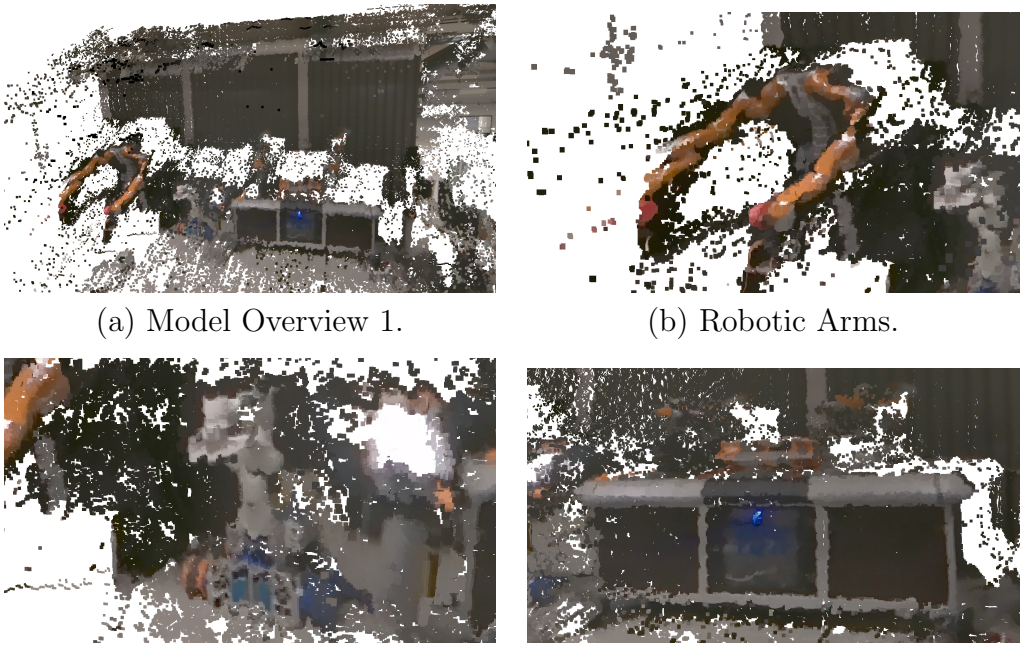
We turn our attention to Structure-from-Motion. The result of the integration of 300 monocular frames from the field test, with an example of the input data and corresponding depth estimated by COLMAP presented in Fig. 7, is presented in Fig. 3.6. On the left, Fig. 3.6(a) we can see the result from SfM without applying the dome port offset parameters discussed in Sec. 2.3.4, and on the right, Fig. 3.6(b), they have been taken into account during the reconstruction process. Only a slight change is observable from left to right, with the ArucoCube clearly visible in both models. However, the right side of the ArucoCube has been reconstructed more accurately, whereas the left side of the ArucoCube seems to attach to an incoherent mass. The mesh on the right side of Fig. 3.6(b) seems to exhibit larger blob-like features on the lake floor. Overall, the regions of the model that were clearly in view by the camera are reconstructed better than occluded or partially occluded regions, such as the left side of the ArucoCube and the region almost behind the camera.
A more accurate reconstruction can be achieved by integrating more than double the number of frames into the model, with applied dome port offset parameters, as seen in Fig. 8. The result is a 3D model where the borders of the ArucoCube are clearly defined when in view. We can see that the occluded regions of the cube have gaps in reconstruction. The surrounding sandy area is represented by a region with a rugged texture, with many artefacts occurring as hundreds of little blobs appear to move upwards from the lake floor. The errors occurring at the Aruco Markers on the cube, clearly distort the sharp edges and corners of the markers, which little blob-like artefacts, again, appearing to move upwards from the cube's surface. The cube also seems to have been stretched to the side, appearing to have morphed into a prism.

3.3.2 In-Air Reconstruction from Stereo Depth

(a) OAK-D Point Cloud. (b) Sea3D Point Cloud.

Figure 3.7: In-air Point Cloud Example 1.

When comparing the point clouds generated by the OAK-D and Sea3D in Fig. 3.7(a) and Fig. 3.7(b), the observations from Fig. 3.4 are reflected in the results. The OAK-D point cloud appears much cleaner, with distinct segmentation of objects at various depths. In contrast, while the Sea3D point cloud also allows for object differentiation (e.g., the drill is closer to the camera than the grey box with the red stripe), it suffers from significantly higher noise levels. This is particularly evident around the table, a textureless surface that lacks distinct features. As mentioned in Sec. 3.2, the depth map for this area exhibited substantial noise, and this is mirrored in the point cloud, with erroneous points extending far behind the main scene, almost as if they were positioned beyond the wall and windows. Regarding the background, the Sea3D performs better, showing similarities to the OAK-D point cloud, though it still presents more gaps in the wall region. Overall, the Sea3D point cloud has more gaps, increased noise (especially in textureless areas), and struggles to preserve object boundaries clearly.
The scene in Fig.11 is used to create the model in Fig. 3.8, where we have positioned the camera approximately 5-6 metres from the target objects. In this case, Sea3D groups larger objects more accurately at approximately the same depth together. These objects, such as the robotic arms or the table, are considerably larger than those in Fig. 3.7 by a factor of 10. The robotic arms in Fig. 3.8(b) are clearly distinguishable, and the arms are pointing away from the body of the robot. Still, there is a lot of noise, with many points floating randomly in the room.



(a) Model Overview 1. (b) Robotic Arms.

(c) White Robot. (d) Table.

Figure 3.8: 3D Model from the Faraway Scene.

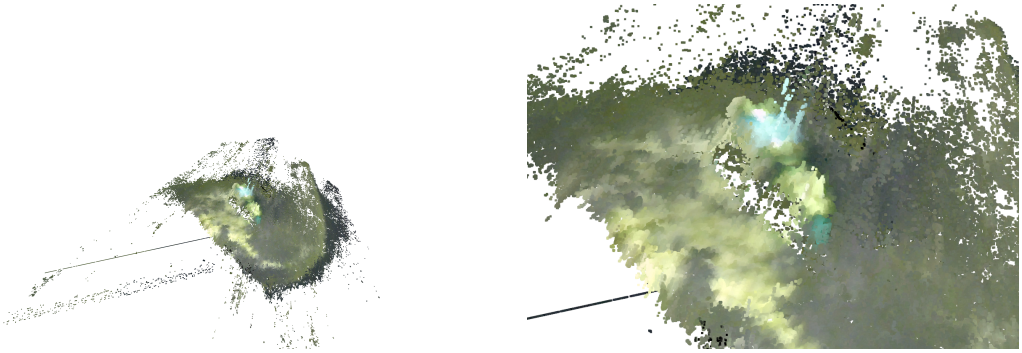
Next, examining the model in Fig. 3.9, created using the TSDF algorithm described in Sec. 2.5.4, we observe the integration of multiple depth maps and RGB images into a voxel grid. The images used for this model mainly were captured from the left-hand corner near the robotic arms, as shown in Fig. 10. The wall in the scene is particularly well-reconstructed, with no obvious gaps. The left robotic arm, positioned closer to the camera, is reconstructed with much greater accuracy than the right arm. Other occluded regions, such as the objects behind the right side of the robotic arm, are also reconstructed inaccurately with considerable amounts of noise, as seen in Fig. 3.9.
Occluded regions are particularly susceptible to noise during TSDF computation. Increasing the data input from these areas can improve accuracy, as demonstrated with the left robotic arm. However, the experiments revealed that the algorithm's camera pose estimation becomes unstable after approximately 600 frames. This instability causes the same scene to be reconstructed at varying positions relative to the voxel grid, leading to a completely inaccurate model.

(a) In-air TSDF Model View 1. (b) In-Air TSDF Model View 2.

Figure 3.9: In-air TSDF Result.

3.3.3 In-Situ Reconstruction from Stereo Depth

The green plastic bottle at the bottom of the tank is clearly visible in the in-situ point cloud shown in Fig. 3.10. Both the green bottle cap and the indentation about a third of the way down the bottle are easily distinguishable. While the edges appear somewhat rough, they do not extend significantly beyond the bottle, making its perimeter clearly identifiable.
However, noise near the edge of the frame, as observed in the corresponding depth map in Fig. 3.5, manifests in the point cloud as a cluster of points that appear to float in-air toward the camera. Additionally, the occluded region of the bottle is not reconstructed, leaving a visible gap on the tank floor beneath it. Additional noise appears to extend behind the bottom of the tank, floating in space.

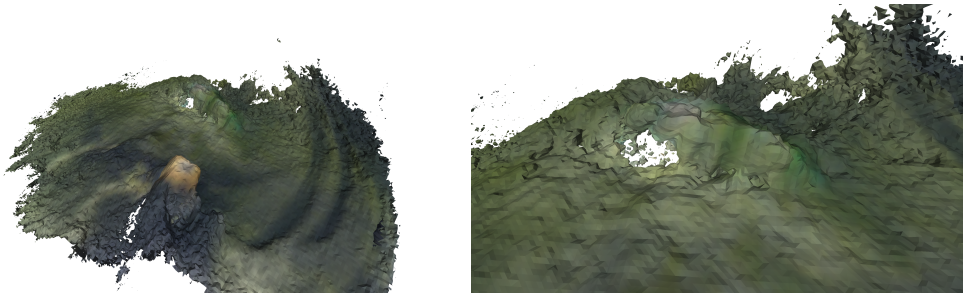


(a) In-situ Point Cloud View 1. (b)In-situ Point Cloud View 2.

Figure 3.10: In-situ point cloud.

Integrating multiple depth maps and corresponding RGB images into a voxel grid, as shown in Fig. 3.11, significantly reduces the noise present in Fig. 3.10. The bottom surface of the tank becomes much smoother, with a subtle wave-like texture forming toward the right side of Fig. 3.11(a). While both the Aruco Cube and the

plastic bottle remain identifiable, especially when compared to an example RGB image used for integration, the algorithm appears to erode the edges, diminishing the perimeters of the objects. Notably, the transparent bottle beside the ArucoCube, visible in Fig. 3.12, is no longer distinguishable in the integrated point cloud.



(a) In-water TSDF Model View 1. (b)In-water TSDF Model View 2.

Figure 3.11: In-situ TSDF model.

This TSDF model is the result of approximately 600 frame integrations, after which the camera pose estimation becomes unstable, resulting in inaccurate scene reconstruction. This limitation currently prevents Sea3D from reliably reconstructing larger scenes.
Interestingly, when compared to the in-air TSDF model in Fig. 3.9, the underwater model exhibits less noise and a more coherent overall structure despite expectations to the contrary.

Figure 3.12: In-situ TSDF Data.

Chapter 4

Discussion

This work demonstrates the feasibility of developing an underwater 3D reconstruction camera specifically designed for targeted research applications, emphasizing its advantages and limitations. It shows that a functional prototype can be built using commercially available components and open-source software, allowing researchers to integrate custom hardware tailored to their unique requirements, an option often lacking in off-the-shelf solutions. We have successfully demonstrated that a prototype can be built from off-the-shelf hardware, enhanced with 3D-printed components, that can conduct field deployments, enabling researchers to collect valuable underwater data with this design. Additionally, we implement 3D reconstruction algorithms with varying strengths and weaknesses that demonstrate the system's capabilities at this stage and suggest potential directions for future enhancements to improve the current iteration.

Monocular-based solutions excel in terms of ease of implementation and visual quality. Techniques such as Structure from Motion (SfM) and Deep Neural Network (DNN) depth estimation are readily available, with minimal adjustments needed to produce satisfactory 3D representations of an underwater scene. Notably, DNN depth estimation maintains high quality and is unaffected by image degradation or refraction errors present in our dataset, yielding results comparable to typical in-air scenes. However, it is important to note that our field test data was collected under particularly favourable visual conditions; thus, assessing DNN monocular depth performance in poor lighting and increased visual haze is essential to fully understand its limitations in underwater environments.

Similarly, SfM offers visually appealing results with significantly less effort than stereo solutions. The shift from two cameras to one simplifies the setup, making monocular solutions particularly advantageous for applications where speed is paramount and a general sense of depth suffices. However, this convenience comes at the cost of geometrical accuracy. As previously emphasized, monocular solutions face scale ambiguity, which severely limits the utility of the resulting 3D data for accurate scene interpretation. The depth and model data generated by DNN monocular depth estimation and SfM lack metric significance; the relationships among fea-

ture points are purely relative, and no definitive conclusions can be drawn regarding the actual size of the represented objects.
This is in contrast to stereo-based solutions; the increased complexity of adding two cameras to the deployment, with all the entailing hurdles to collect relevant data, such as stereo calibration and rectification, camera synchronization, increased storage footprint and so on, is rewarded with actual geometric measurements of the environment. We have shown in Sec. 3.3 that Sea3D is approaching the performance of a commercial depth camera. Despite its clear limitations, Sea3D allows the user to quickly receive a 3-dimensional representation of in-air scenes and in the underwater environment. The integration of up to 600 frames demonstrates the capability of Sea3D to integrate depth information from multiple viewpoints into voxel grids, delivering more sophisticated models than point clouds.
For successful 3D reconstruction, we cannot overstate the importance of accurate camera calibration, ensuring it is performed for the whole frame resolution and at a large amount of varying distances and angles. This is one of the easiest ways for the user to have a massive impact on the accuracy of the depth maps and, thus, the resulting point clouds and voxel grids. Calibration should be performed carefully, especially underwater, where light absorption at the edges can render calibration less accurate. In Fig. 3.2, we can see the inaccuracies of the calibration, which are most likely part of the cause for the noise seen in Fig. 3.5. A more accurate calibration that applies to the whole frame of the camera could be achieved by performing in-situ calibration with a larger calibration target, as in Fig. 5.
We have demonstrated that Sea3D performs better for objects at medium distances from the camera than close-range objects, as it groups larger regions of similar depth more effectively at that range. This suggests that computing depth at lower resolutions has an advantage, as it consolidates broader regions rather than focusing on fine-grained estimates for smaller areas. Comparing the OAK-D's resolution, shown in Tab. 2.3, with Sea3D's depth map resolution of 1920×1080, we find that the OAK-D's depth maps contain approximately 44% fewer pixels. This lower resolution provides an advantage for capturing a more generalized view of the scene, while Sea3D excels in capturing finer details but at the risk of introducing more noise to the depth map. Post-processing techniques play a crucial role in enhancing depth map quality. Improving the post-processing techniques of Sea3D, such as employing segmentation techniques to group larger pixel regions into coherent objects and surfaces, should significantly reduce noise and improve depth estimation accuracy. For underwater image data, a more accurate result may be achieved by first enhancing the image data to account for the image degradation discussed in Sec. 1.2.1.
We have demonstrated that integrating multiple RGB-depth pairs into a voxel grid using a TSDF algorithm can significantly reduce noise and fill in gaps by updating voxels as images are captured from different viewpoints. This results in more detailed models and a better understanding of their volumetric properties. Currently, the integration is limited to approximately 600 frames, after which the algorithm's pose estimation drifts. However, significant improvements can be made by incor-

porating an inertial measurement unit (IMU), which can enhance pose accuracy through visual-inertial odometry techniques. These techniques combine pose estimation from feature matching with inertial data for a more precise result, as shown in [31]. To address this, we have integrated an IMU into the processor payload and provided a framework for synchronized IMU and stereo vision data acquisition in ROS2, as detailed in Sec. 2.1.3 and Sec. 2.2.3. Implementing a VIO algorithm in future iterations allows Sea3D to reconstruct much larger areas since the pose can be accurately estimated for longer. The capabilities of Sea3D could be extended to real-time performance, given an accurate pose estimation from these sensor fusion techniques. Accurate pose estimation is the greatest hindrance to achieving real-time reconstruction since the ample processing power of the hardware accelerators on the Orin provides the necessary computational power. The IMU could also enable Sea3D to compute metric depth information for monocular depth estimation approaches by providing a reference for the distance travelled between frames, as demonstrated in [32].
Nevertheless, the importance of capturing objects from varying viewpoints for accurate reconstruction has been demonstrated. This introduces challenges for underwater applications where the camera is, e.g. mounted on an ROV since objects on the seafloor will commonly only be captured from a top-down angle, with the sides and especially the bottom of the object partially or completely occluded. Surprisingly, the large amount of noise in in-air voxel grids at occluded regions and borders of the image is considerably less pronounced in the underwater tank model, Fig. 3.11. The overall model appears smoother, and the general structure of the surroundings is more accurately preserved. This is most likely due to the robotic arms in Fig. 10 being very close to the camera, resulting in fine-grained noise, as discussed earlier. As seen in Fig. 11, we can see that Sea3D reconstructs target objects that are situated at a medium distance away from the camera, at about 5-6, more coherently than close-up objects, where noise is often introduced. This is most likely because the baseline has been configured, so accurate depth can be computed at a minimum depth of $0.6m$. Adjusting the baseline for the intended application is a crucial step in providing accurate data. It can be adjusted based on the application by adjusting the 3D printed support in CAD and reprinting the structure.

Chapter 5

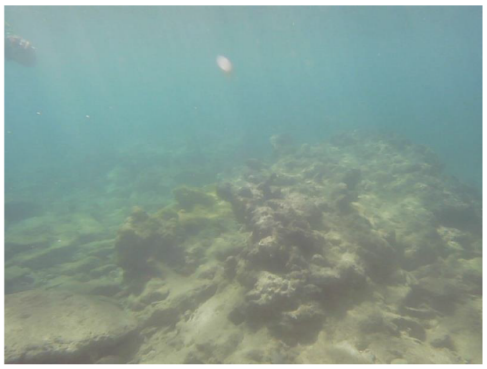
Conclusion

In this work, we present a prototype for a custom underwater stereo camera for 3D reconstruction based on high-resolution industrial cameras and acceleration hardware and validate its functionality in a lab environment and a real-world field deployment as an ROV payload.
Furthermore, we implement two monocular depth estimation algorithms and validate their performance in an underwater setting while presenting the limitations of monocular methods to estimate depth accurately, providing only a notion of depth. Accounting for the error induced by refraction in a Structure-from-Motion approach doesn't significantly enhance the resulting model, with the main benefit of monocular approaches being the ease of implementation.
We compare the monocular approaches to the performance of Sea3D's stereo depth estimation, which is demonstrated to approach the performance of a commercially available stereo camera, showing limitations in reconstruction accuracy in textureless regions and objects close to the camera lens. We have successfully demonstrated that the system can integrate up to approximately 600 frames into a voxel grid model with a TSDF algorithm, which leads to a considerable reduction in noise compared to the point cloud of a singular scene. After 600 frames, the algorithm fails to accurately estimate the camera's pose, demonstrating the necessity of robust pose estimation over longer periods of time in order to reconstruct larger areas, and with regards to achieving real-time performance.
Promising future directions are the integration of Visual-Interial Odeometry for accurate pose estimation, by fusing IMU data and pose estimation from feature extraction, the effect of enhanced underwater image data on the resulting model, and geometrically accurate depth estimation from monocular approaches with the addition of an IMU.

(a) Caustics 1

(b) Caustics 2

(c) Green-Blueish Color Effect



(d) Haze Effect

Figure 1: Common Underwater Images with Haze, Color and Caustic Effects[7].

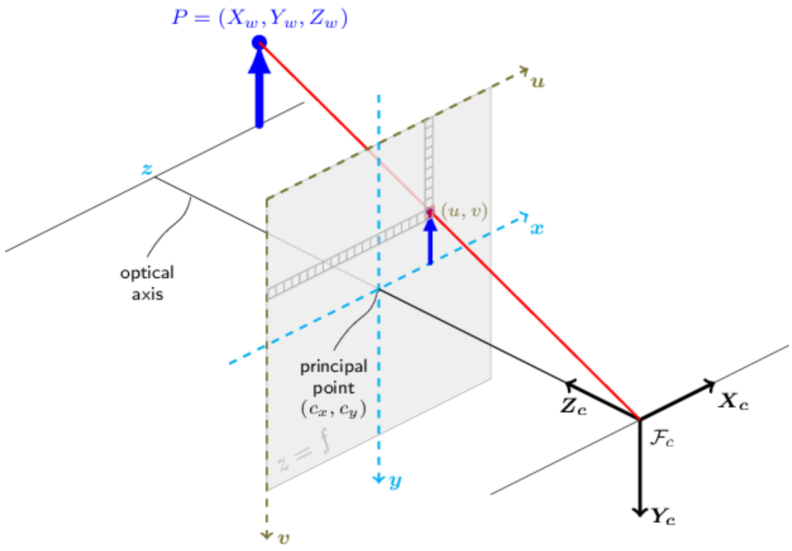


Figure 2: The pinhole camera model with the point of interest $\boldsymbol{P} = (X_w, Y_w, Z_w)$, principal point (c_x, c_y) and the projection of $\boldsymbol{P}$ on the image plane (u, v), [33]

Figure 3: Printed Cylindrical Camera Support

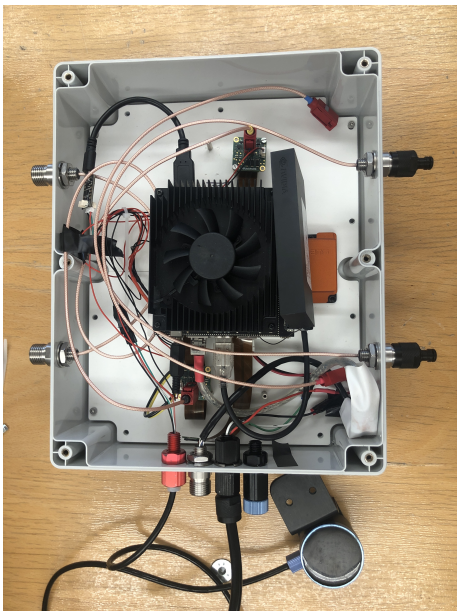
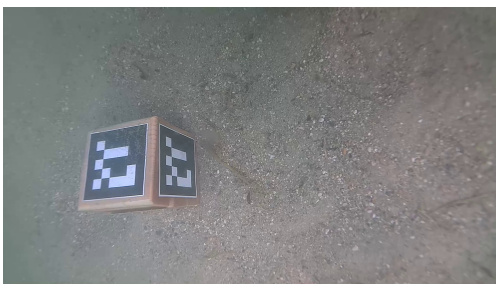


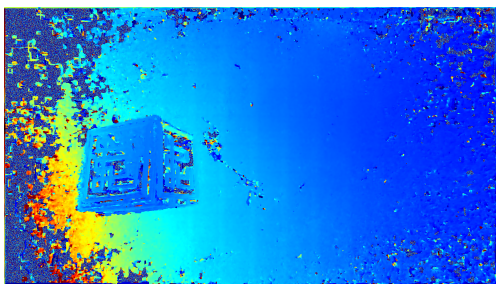
Figure 4: Processor Payload

Figure 5: An A0 format, 10×7 checkerboard grid calibration target.

Figure 6: OAK-D PoE [34]

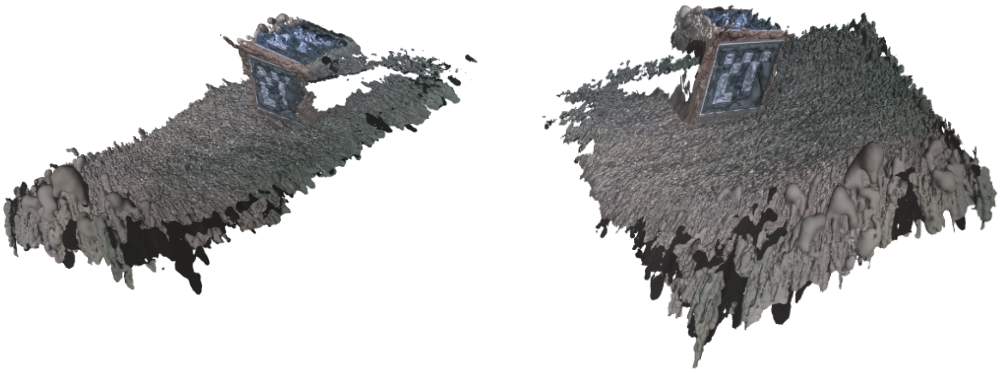


(a) Example input for SfM.



(b) Depth from SfM.

Figure 7: Example input data for SfM.



(a) SfM Mesh View 1 (b) SfM Mesh View 2

Figure 8: Monocular depth map estimation result.

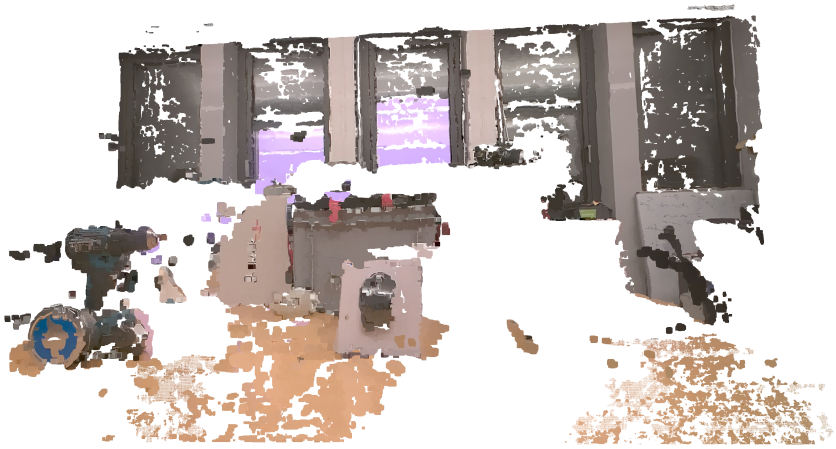
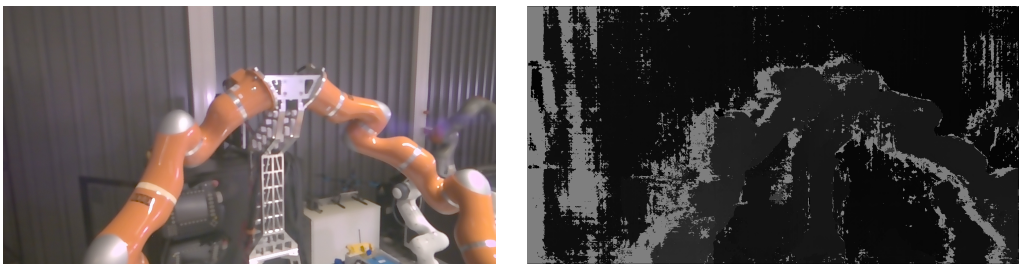


Figure 9: Close up point cloud.



(a) In-air Scene 2 RGB. (b) In-air Scene 2 Disparity.

Figure 10: In-air Stereo Example Scene 2.

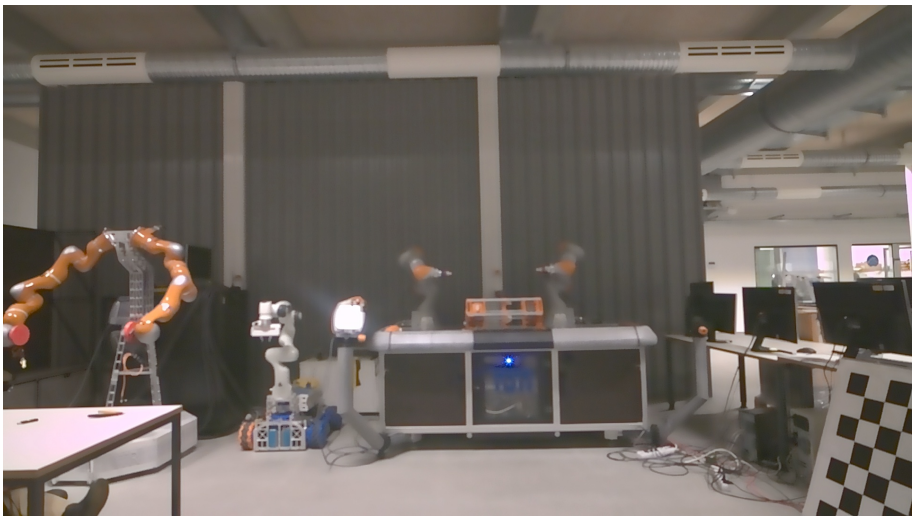


Figure 11: In-air Stereo Example Scene 3.

List of Figures

List of Tables

Acronyms

ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processor
ESDF	Euclidean Signed Distance Function
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
PVA	Programmable Vision Accelerator
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
SDF	Signed Distance Function
SfM	Structure-from-Motion
SGM	Semi-Global Matching
TSDF	Truncated Signed Distance Function
VIC	Visual Imaging Compositor
VIO	Visual-Intertial Odometry
VPI	Vision Programming Interface

Bibliography

[1] M. She, Y. Song, J. Mohrmann, and K. Köser, "Adjustment and Calibration of Dome Port Camera Systems for Underwater Vision," 2019. [Online]. Available: https://www.geomar.de/fileadmin/personal/fb2/mg/kkoeser/domecalibration_preprint.pdf

[2] K. Hu, T. Wang, C. Shen, C. Weng, F. Zhou, M. Xia, and L. Weng, "Overview of Underwater 3D Reconstruction Technology Based on Optical Images," *Journal of Marine Science and Engineering*, vol. 11, no. 5, p. 949, Apr. 2023.

[3] M. Massot-Campos and G. Oliver-Codina, "Optical Sensors and Methods for Underwater 3D Reconstruction," *Sensors*, vol. 15, no. 12, pp. 31 525–31 557, Dec. 2015.

[4] N. Smolyanskiy, A. Kamenev, and S. Birchfield, "On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach," 2020.

[5] M. She, F. Seegr Käber, D. Nakath, and K. K Köser, "Refractive COLMAP: Refractive Structure-from-Motion Revisited," 2024.

[6] J. L. Schonberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016.

[7] G. Huo, Z. Wu, J. Li, and S. Li, "Underwater Target Detection and 3D Reconstruction System Based on Binocular Vision," *Sensors*, vol. 18, no. 10, p. 3570, Oct. 2018.

[8] F. Oleari, F. Kallasi, D. L. Rizzini, J. Aleotti, and S. Caselli, "An Underwater Stereo Vision System: From Design to Deployment and Dataset Acquisition," in *OCEANS 2015 - Genova*. IEEE, May 2015.

[9] C. Wang, Q. Zhang, S. Lin, W. Li, X. Wang, Y. Bai, and Q. Tian, "Research and Experiment of an Underwater Stereo Vision System," in *OCEANS 2019 - Marseille*. IEEE, Jun. 2019.

[10] J. Servos, M. Smart, and S. L. Waslander, "Underwater Stereo SLAM with Refraction Correction," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013.

[11] Y. Girdhar, N. McGuire, L. Cai, S. Jamieson, S. McCammon, B. Claus, J. E. S. Soucie, J. E. Todd, and T. A. Mooney, "CUREE: A Curious Underwater Robot for Ecosystem Exploration," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023.

[12] S. Martin. (2024) Turning the Tide on Coral Reef Decline: CUREE Robot Dives Deep With Deep Learning. Accessed: Jul. 2024. [Online]. Available: https://blogs.nvidia.com/blog/coral-reef-decline-curee-robot-jetson-isaac-omniverse/

[13] P. Tueller, P. R. Paxson, D. Truong, D. Drews, J. Doan, J. Guerrero, T. Cosino, Z. Sun, R. Maddukuri, V. Suresh *et al.*, "Fishsense: Underwater RGBD Imaging for Fish Measurement and Classification," *Ocean Visions 2021 Summit*, 2021. [Online]. Available: https://kastner.ucsd.edu/wp-content/uploads/2021/08/admin/oceans21fishsense.pdf

[14] J. Ghorpade, "GPGPU Processing in CUDA Architecture," *Advanced Computing: An International Journal*, vol. 3, no. 1, pp. 105–120, Jan. 2012.

[15] FRAMOS. (2024, Sep.) Sensor Module Ecosystem User Guide. FRAMOS GmbH. [Online]. Available: https://www.framos.com/framos-fsm-startup/downloads/User-Manual/FSM_Ecosystem_UserManual.pdf

[16] ——. Multi-Sensor Synchronization:SONY Rolling Shutter Sensors. FRAMOS GmbH. Accessed: Sep. 2024. [Online]. Available: https://www.framos.com/framos-fsm-startup/downloads/AppNotes/FRAMOS-AppNote_MultiSensor-Synchronization.pdf

[17] H. Grant. (2024, Sep.) Build OpenCV from Source in CONDA on Jetson. Accessed: Sep 2024. [Online]. Available: https://github.com/HJGrant/opencv_conda_build_on_orin/tree/main

[18] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2003.

[19] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer International Publishing, 2022.

[20] M. She, D. Nakath, Y. Song, and K. Köser, "Refractive Geometry for Underwater Domes," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 183, pp. 525–540, Jan. 2022.

[21] H. Hirschmuller, "Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPRâ05)*. IEEE.

[22] C. Liu, J. Gu, K. Kim, S. Narasimhan, and J. Kautz, "Neural RGB-D Sensing: Depth and Uncertainty from a Video Camera," 2019.

[23] M. Nezveda, "Evaluation of Depth Map Post-Processing Techniques for Novel View Generation," 2014.

[24] NVIDIA. Visual Programming Interface Documentation. Accessed: Sep. 2024. [Online]. Available: https://docs.nvidia.com/vpi/index.html

[25] OpenCV. OpenCV Documentation. Accessed: Sep. 2024. [Online]. Available: https://docs.opencv.org/3.4/index.html

[26] D. Werner, A. Al-Hamadi, and P. Werner, *Truncated Signed Distance Function: Experiments on Voxel Size*. Springer International Publishing, 2014, pp. 357–364.

[27] H. Oleynikova, Z. Taylor, M. Fehr, J. Nieto, and R. Siegwart, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," Sep. 2016.

[28] F. Seegräber, M. She, F. Woelk, and K. Köser, "A Calibration Tool for Refractive Underwater Vision," May 2024.

[29] BlueRobotics. Watertight Enclosure End Caps, Domes, and Flange Caps. BlueRobotics. Accessed: Sep. 2024. [Online]. Available: https://bluerobotics.com/store/watertight-enclosures/locking-series/wte-end-cap-vp/

[30] H. Grant, "Sea3D Data Processing," 2024, Accessed: Sep. 2024. [Online]. Available: https://github.com/HJGrant/Sea3D_Data_Processing

[31] V. Usenko, J. Engel, J. Stuckler, and D. Cremers, "Direct Visual-Inertial Odometry with Stereo Cameras," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016.

[32] J. Mustaniemi, J. Kannala, S. Särkkä, J. Matas, and J. Heikkilä, "Inertial-Based Scale Estimation for Structure from Motion on Mobile Devices," Nov. 2016.

[33] OpenCV. Pinhole Camera Model. OpenCV. Accessed: June 2024. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[34] Luxonis, "OAK-D PoE," 2024, Accessed: Sep. 2024. [Online]. Available: https://docs.luxonis.com/hardware/products/OAK-D%20PoE

License