The VAROS Synthetic Underwater Data Set: Towards realistic multi-sensor underwater data with ground truth

Peder Georg Olofsson Zwilgmeyer [1] Mauhing Yip[1] Andreas Langeland Teigen[2]
Rudolf Mester[2] Annette Stahl[1]

[1]Department of Engineering Cybernetics [2]Department of Information and Computer Sciences
Norwegian University of Science and Technology, Trondheim, Norway

Abstract

Underwater visual perception requires being able to deal with bad and rapidly varying illumination and with reduced visibility due to water turbidity. The verification of such algorithms is crucial for safe and efficient underwater exploration and intervention operations. Ground truth data play an important role in evaluating vision algorithms. However, obtaining ground truth from real underwater environments is in general very hard, if possible at all.

In a synthetic underwater 3D environment, however, (nearly) all parameters are known and controllable, and ground truth data can be absolutely accurate in terms of geometry. In this paper, we present the VAROS environment, our approach to generating highly realistic underwater video and auxiliary sensor data with precise ground truth, built around the `Blender` modeling and rendering environment. VAROS allows for physically realistic motion of the simulated underwater (UW) vehicle including moving illumination. Pose sequences are created by first defining waypoints for the simulated underwater vehicle which are expanded into a smooth vehicle course sampled at IMU data rate (200 Hz). This expansion uses a vehicle dynamics model and a discrete-time controller algorithm that simulates the sequential following of the waypoints.

The scenes are rendered using the raytracing method, which generates realistic images, integrating direct light, and indirect volumetric scattering. The VAROS dataset version 1 provides images, inertial measurement unit (IMU) and depth gauge data, as well as ground truth poses, depth images and surface normal images.

1. Introduction

In the automotive community dealing with driver assistance and self-driving cars, the KITTI [13] data set initiated a significant boost to systematically investigate and characterize various challenges in robotic environment perception. While KITTI is a family of datasets obtained in the real world, the CARLA [10] simulated autonomous driving environments provided an important completion as it allows, beyond testing perception systems, also to test planning and control algorithms ('AI agents') that steer simulated cars. CARLA is based on a game engine and computer graphics, and can provide provably precise ground truth.

By generating visual and sensor data using simulations and high-end computer graphics methods, it is possible to achieve realistic imagery and have access to completely accurate information about vehicle poses and simulated sensor measurements for every timestep. As the 3D geometry of the scene is known, this can be used to generate correct point clouds and depth maps in all environments that can be simulated. The COnGRATS [4] dataset is one synthetic traffic dataset including such information coupled with realistic path-traced imagery.

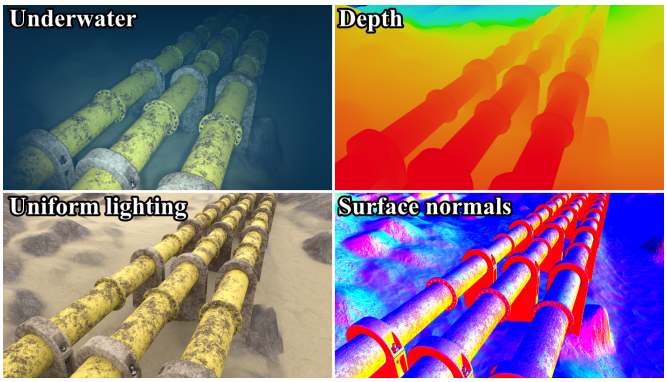

Figure 1. The image types contained in the initial version of the VAROS underwater dataset: underwater images and corresponding images with uniform lighting and no water, as well as pixel-accurate surface normal and depth maps.

In this paper, we present an approach that allows the flexible creation of highly realistic synthetic image sequences, featuring visually degraded underwater scenes with corresponding ground truth data (cf. figure 1). For the genera-

tion of image and auxiliary sensor data, we use the public domain modeling and rendering engine `Blender` [6] version 2.93, extended by a set of own software. In contrary to the game engine `Unity`, `Blender` allows to control more imaging parameters leading to more realistic image sequences.

Combining basic pose sequencing tools in `Blender` with a custom build control-system allows for detailed control of the vehicle course including moving illumination. Image sequences can thus be created both from an in-vehicle perspective as well as from a static position looking onto the underwater scenery, allowing high configuration flexibility applicable to a wide range of applications.

The scenes are rendered using the raytracing method, which is capable of producing images with accurate illumination by integrating direct light, and indirect volumetric scattering. Depth images, surface normal vectors for each pixel, non-uniformly and uniformly illuminated RGB images, sensor measurements from an IMU, and depth sensor were generated as noise-free data, such that they can be augmented with noise to fit a wide range of sensor-models depending on the users application.

2. Related Work

Datasets such as KITTI [13] and LiU [21] provide a large amount of image- and sensor data for traffic scenarios. Likewise, for micro aerial vehicles EuRoC [7] provides images, sensor data and accurate ground truth position measurements from a Leica multistation land surveying tool.

Above water datasets can utilize a global positioning system (GPS) for large-scale traffic scenarios or multistations for land surveying to gain a high level of accuracy, but these possibilities are not present in most underwater scenarios. Alternative methods to obtain a measure of the ground truth pose data varies. AQUALOC [12] uses an offline and highly accurate structure from motion method to obtain an estimate of the camera poses and the 3D environment. A different approach [22] is to place markers in the underwater environment, where the position of each marker relative to the other markers is measured.

The availability of underwater robotic simulation software in the research field is scarce [23, 26]. The availability of underwater robotic simulation software in the research field is limited, where the largest software packages are [23, 26]. In general, these software packages has varying functionalities, such as support for fluid dynamic simulation, different vehicle models and motion controllers. They have in common that simulation of realistic underwater imagery is not the main priority, which results in relatively low-quality underwater images. With the increasing use of computer vision, the demand for realistic underwater imaging is increasing.

Physics based underwater image generation: Some underwater imaging simulators use physics based models to simulate how photons travel through water. However, the analytic method, namely the radiative transfer equation [24], is difficult to solve. Therefore, approximations are used. The two most frequently used approximations are the Fog model [8] for shallow sea water and the Jaffe-McGlamery [17, 27, 29] model for deep-sea water. Generally these methods require knowledge of the inherent optical properties of the water that is simulated, and by extension different optical approximation models for deep and shallow water. The main drawback to this approach is that the inherent optical properties [25] of water, like the absorption [1] and scattering coefficients [15] are wavelength dependent. This information is difficult to obtain, and highly depend on the type of water [18].

Existing image simulators: As discussed, the existing simulation software such as the UUV Simulator [23], which builds upon the Gazebo [20] simulator, and UWSim [26] provides a highly interactive simulation environment, but they do not provide the physically realistic imagery we require. Other methods, as explored by [11], involves using an image mosaic from a real underwater mission and then simulating different visual conditions. This method has two main drawbacks. First, the underwater conditions at the time of image acquisition for the image mosaic will effect the quality of the virtual seafloor. Second, the seafloor is modeled as a flat scene which does not contain any other geometry information that influences the light scattering which comes from the simulated vehicle. Looking back at the automotive community, the COnGRATS [4, 5] dataset generates high quality physically realistic imagery using path-traced 3D computer graphics with dynamic vehicle lights and varying weather conditions.

Due to the lack of proper underwater imagery with highly accurate ground truth, we have created an underwater dataset with physically realistic imagery in a detailed underwater environment. We make use of the information the image generation process generates in order to provide a significant amount of ground truth data which exceeds what is commonly available through underwater datasets created from real-world scenarios.

3. Approach

Our approach to generating highly realistic underwater video and auxiliary sensor data with precise ground truth consists of three main building blocks: First, we need a realistic computer graphics model of a large-scale underwater environment in which a simulated underwater robot can be steered (cf. sec. 3.1). The second building block comprises the generation of image and auxiliary sensor data using `Blender`. This task is significantly more complex for

an underwater scenario compared to regular computer vision scenarios. Particular attention has to be paid to the realistic modeling of the dispersive medium, that is: seawater with a significant amount of backscattering (cf. sec. 3.2). Third, the generation of physically realistic vehicle and camera pose sequences is done by defining waypoints for the simulated vehicle in the given underwater environment and subsequently following these waypoints using a dynamic vehicle model and a controller algorithm (cf. sec. 3.3). We describe these steps for environment, sensor data, and pose sequence generation in more detail in the subsequent sections.

3.1. Underwater Environment

Creating realistic synthetic video and simulated sensor data obviously requires a realistic 3D scene model. For the VAROS simulation, an underwater landscape was created by transferring a photogrammetric model of a real (above water) landscape into the underwater domain and retexturing it with fine grained sand and rock textures. In order to make the environment more complex, we have added 3D models of man-made objects. The resulting environment contains both plain areas with a simple seafloor as well as complex areas populated with many man-made objects.

More details of the underwater scene creation are provided in the next sections.

3.1.1 Underwater 3D Landscape

In order to reduce the artistic labor required and maximize realism, we used a scene model of the Vasquez Rocks by Austin Beaulier [3] which is available under the Creative Commons Attribution license [9]. The model has been created using photogrammetric methods; we used it as an underwater model by modifying some of the geometric attributes and replacing some surface textures (cf. figure 2).

Seafloor model pre-processing: The original photogrammetric mesh model consists of many separate 3D meshes. We have joined them into one object and merged nearby vertices. As the model still contained some holes and non-connected vertices, it was "shrinkwrapped" with a new watertight mesh, which had a higher resolution than the original mesh. This was necessary as we need higher resolution representations for close-up views. We used `Zbrush`[1] to project a new mesh onto the photogrammetry model. First, the new mesh was projected at a low resolution, then the resolution was increased and projected again. This was repeated until the new mesh contained 16.6 million vertices, and then it was decimated[2] to contain 250.000 vertices. This decimated mesh was then imported back into `Blender`.

[1] `Zbrush` is a digital sculpting software from http://pixologic.com/

[2] Decimation triangulates and reduces the amount of vertices present in a mesh while preserving details.

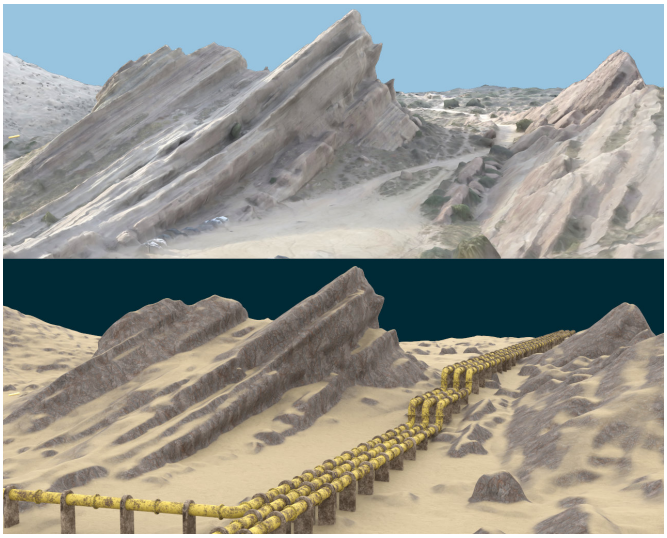

Figure 2. Comparison of the original Vasquez Rocks 3D model by Austin Beaulier [3] (top) and the modified version used in the underwater environment (bottom). These rendering results use uniform ambient illumination and do not show any seawater effects.

Adaptive texturing using surface normal vectors: We applied some modification to the texturing of the mesh model in order to let it correspond more realistically to real underwater scenes. Horizontal flat areas of the model should be covered in sand, while vertical areas should expose rock. Using `Blender`'s material node editor, we created a highly customized physically based rendering (PBR) material for the scene seafloor using the "Principled BSDF" node. The bidirectional scattering distribution function (BSDF) determines the direction where the light bounces upon hitting the material in the rendering process.

By accessing the surface normal vectors of an object, a mask is created which isolates horizontal areas where the surface normal is pointing upwards. A 'procedural blending mask' is used to apply a sand texture for horizontal parts of the model, and a rock texture everywhere else. Tiling a 4096×4096 texture 50 times to cover the model results in a sufficient resolution when viewed from a close distance. Ensuring that the same pattern is not repeated remains a pending challenge. The procedural blending mask removes the need to manually adapt the texture to the environment geometry and reduces the apparent repetition of the tiled textures. Figure 2 shows the original and modified environments.

3.1.2 Additional Man-made Objects

Scene content diversity and the pre-conditions for reality-like mission simulation were achieved by populating the scene with man-made objects ('assets' in computer modeling language) modeled by us. These objects include straight and 90-degree corner pipes, pipe supports, bolts and nuts

(cf. figure 2). When using assets in the scene, it must be ensured that their geometry is similar to that of the real world equivalents. Sharp corners in the models have been beveled or rounded, in order to reflect light from such an edge. This is important as the perfectly sharp 90-degree corners often found on computer-aided design (CAD) models will not reflect light, which decreases the realism of the image and may even affect the performance of computer vision algorithms. Furthermore, we created procedural dirt and rust materials in `Blender` for these assets, allowing for high-resolution textures at a low computational cost. By using the objects location as a pseudo-random seed in material generation, each dirt pattern is unique.

The creation of the water volume along with its physical properties utilized in the generation of images and sensor data are described in the following sections.

3.2. Sensor Data Generation

In this section we describe the generation of image and auxiliary sensor data using `Blender`.

3.2.1 Image Data

In the VAROS dataset, images were generated using the path-tracing renderer Cycles in `Blender` [6], which uses a Monte-Carlo sampling process for each pixel. The higher the number of samples is, the more realistic is the resulting distribution of light in the scene. For our underwater images (which we call image type A), we empirically determined 375 samples per pixel to be sufficient. Subsequently, the "denoiser" node in the "compositor" of `Blender`, which is used for render post processing and image output, is applied. In order to avoid too expensive rendering times we use Open Image Denoise [16] instead of dramatically increasing the number of ray samplings. If like in usual practice, texture and geometry information is used in the denoiser as well, then this information will imprint onto areas obscured by the scattering volume, which introduces visual artifacts in the image. Therefore, we use only the noisy image in that step.

For the full-transparency versions of the underwater image with uniform lighting (image type B), as well as the surface normal vector (image type C) and depth images (image type D), 20 samples per pixel are used to gain a sufficient level of quality with moderate computational effort. Here, the NLM[3] denoiser is used, which works on any CPU or AMD/Nvidia GPU.

Before we describe in the subsequent sections, the generation of these different image types, we elaborate on the image modeling, considering the constraints of the Blender software which do not allow a perfect coincidence with detailed physical models.

[3]NLM: Cycle's native "non-local means" denoiser.

Table 1. Parameters used to create the water scattering volume in `Blender`. Color values are represented in RGB, "A" refers to the alpha channel indicating a measure for transparency.

Parameter	Value	
Volume density	0.125	
Volume color	$(0.014, 0.146, 0.345, 1.00)$	RGBA
Absorption color	$(0.000, 0.394, 1.00, 1.00)$	RGBA
Anisotropy	0.8	

Water volume simulation/creation: Light transmission through a water volume is determined by physical as well as chemical properties such as disolved matter, meaning floating particles of different material, shape and size, which influences the absorption of the dissolved matter in general. The water in the simulation environment is created by using a scene-wide volume which both scatters and absorbs light. In `Blender` this is created using a "principled volume" BSDF node which approximates the behavior of light in a scattering medium to generate physically realistic results. The parameters used are the volume density, color, anisotropy and absorption color. The volume density controls the attenuation of light in the volume. The anisotropy value $g \in [-1, 1]$ determines the amount of forward light scattering in the volume, following the Henyey-Greenstein (HG) phase function[14]. The HG model is known to be only an approximation, but it is the only model which is currently available in the renderer Cycles.

Volume color determines the direct color (= the color of the backscattering particles, becoming more apparent the more dense the volume is) of the volume, where the color intensity is dependent on the volume density. The absorption color is used to emulate the effect where the water molecules and suspended particles absorb different wavelengths. The color change caused by absorption increases in intensity with the distance light travels in the scattering medium. Respective example settings for the different parameters can be found in table 1. Figure 3 shows a real reference image and the simulated underwater image.

Vehicle lights: Simulated spotlights are placed 0.1m behind the camera, and 0.15m to each side along the vehicle's x-axis. The total angle of the light cone is $65°$, with a soft edge blend to create a lower light intensity further away from the center of the spotlight. Again, this is only a coarse approximation to the light distribution of a real spot light, due to the limitations of the rendering system.

Camera intrinsics and parameters: The camera intrinsic and lens parameters used for the camera model in the simulation are listed in table 2. In `Blender`, the lens type is set to "perspective", and the aperture is used to generate a depth-of-field effect.

Table 2. Camera and lens parameters used in the simulation camera model, inspired by the parameters from a real camera and lens setup.

Parameter	Value
Render resolution [pixels]	1280×720
Sensor width [mm]	4.416
Sensor height [mm]	2.484
Shutter type	Global shutter
Focal length [mm]	3.4
Aperture	1.7

Exposure correction: Exposure correction (EC) is added in the post-processing step outside of `Blender` due to auto-exposure not being natively supported in `Blender`. By exporting the underwater images as lossless 16-bit OpenEXR files, no information is lost when EC is applied. It is first after EC that the images are exported as PNG files. Figure 3 shows an exposure corrected image.

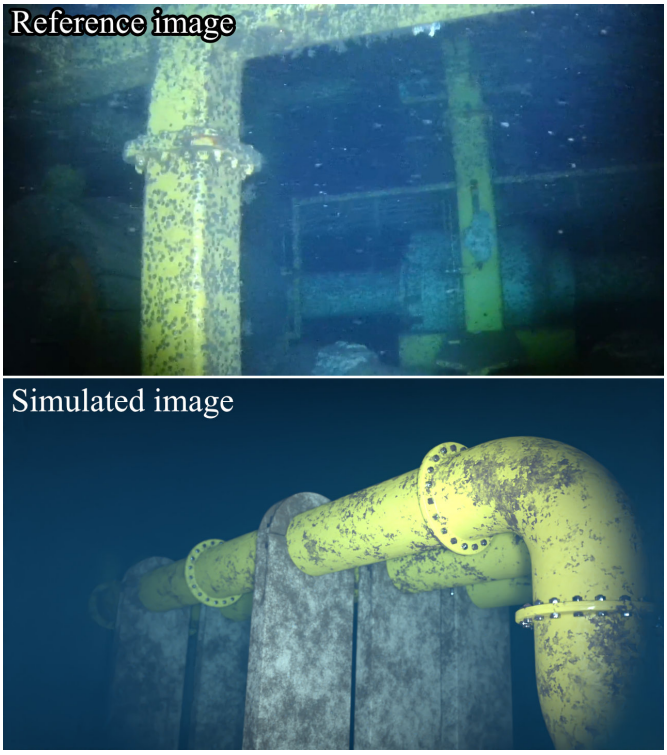

Figure 3. Top: Reference image from a real underwater environment. Bottom: Exposure corrected image from the simulation environment with water properties designed to closely match the visual properties of the reference image.

Image type A - Underwater monocular RGB images: These images contain a depth-of-field effect, caused by the aperture listed in table 2. This results in very close or very distant objects in the scene being out of focus. The only illumination present is provided by the spotlights on the underwater vehicle. Thus the illumination is strongly varying with the motion of the vehicle. Lastly the adaptive EC algorithm is applied to these images.

Image type B - Uniformly illuminated monocular RGB images: These images are rendered with uniform ambient lighting and disabled depth-of-field effect. They serve as reference RGB images with constant lighting, as many computer vision algorithms assume constant lighting. The images are post-processed with a constant EC as the illumination is constant, and they are exported as 8bit PNG files. These images are included as many computer vision algorithms assume static lighting and can be used for determining the effect of the dynamic lighting present in the main underwater image sequence.

Image type C - Surface normal images: Normal images are images where the world coordinate frame surface normal, $\boldsymbol{n}^w$, for the surface visible in each pixel is stored in the RGB channels of the image, using 8bits per channel. To ensure correct export of the normal vectors, the color management in `Blender` is disabled by setting the view transform to "raw" and the sequencer to "linear" as listed in table 3. `Blender` uses the OpenGL [28] normal format, hence $\boldsymbol{n}^w = [n_x^w \; n_y^w \; n_z^w]^\top$, with $\{n_x^w, \; n_y^w, \; n_z^w\} \in [-1, 1]$. The three components of the normal vectors are mapped from $[-1, 1]$ to the three RGB channels in $[0, 255]$. To ensure smooth surface normals, automatic normal smoothing was applied to all edges where the angle between each surface normal of the mesh faces is equal to or lower than 30 degrees.

Image type D - Depth images: Depth images store the distance from the camera to the surface element that corresponds to each pixel. Following the standard set by the KITTI dataset [13], the data is represented as 16-bit grayscale PNG images. The color management in `Blender` is disabled to ensure that only the raw data is exported. The export and color management settings are listed in table 3. Using the KITTI standard, the distance to a given pixel is mapped to a value in $[0, 2^{16} - 1] \Leftrightarrow [0, 65535]$. Let the start and end distance be denoted as d_{end} and d_{start}, then the resolution is expressed as in equation (1).

$$r = \frac{d_{end} - d_{start}}{2^b - 1} \tag{1}$$

3.2.2 Inertial Measurement Unit Data

The IMU data is generated by using the states from the vehicle, namely position and orientation, as well as the temporal first and second order derivatives of these states. The data are transformed from frame $\boldsymbol{w}$ to the reference frame $\boldsymbol{r}$ and vehicle frame $\boldsymbol{v}$ depending on the requirements. Using equation (2) from [2], we calculate the translational acceleration $\boldsymbol{a}_{vr}^s$ and angular velocity $\boldsymbol{\omega}_{vr}^s$. These measurements

represent the relative motion from $\boldsymbol{r}$ to $\boldsymbol{v}$, expressed in the sensor frame $\boldsymbol{s}$.

$$\begin{bmatrix} \boldsymbol{a}^s_{vr} \\ \boldsymbol{\omega}^s_{vr} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{sv}(\boldsymbol{R}_{vr}(\ddot{\boldsymbol{t}}^r_{vr} - \boldsymbol{g}^r) + ([\boldsymbol{\omega}^v_{vr}]_\times [\boldsymbol{\omega}^v_{vr}]_\times + [\dot{\boldsymbol{\omega}}^v_{vr}]_\times)\boldsymbol{t}^v_{sv}) \\ \boldsymbol{R}_{sv}\boldsymbol{\omega}^v_{vr} \end{bmatrix} \tag{2}$$

$\boldsymbol{R}_{sv}$ is the rotation matrix from $\boldsymbol{v}$ to $\boldsymbol{s}$, and $\boldsymbol{R}_{vr}$ is from $\boldsymbol{r}$ to $\boldsymbol{v}$. $\ddot{\boldsymbol{t}}^r_{vr}$ is the acceleration from $\boldsymbol{r}$ to $\boldsymbol{v}$ expressed in frame $\boldsymbol{r}$, and $\boldsymbol{t}^v_{sv}$ is the position offset of the IMU with respect to the vehicle. $\boldsymbol{\omega}^v_{vr}$ is the angular velocity from $\boldsymbol{r}$ to $\boldsymbol{v}$ expressed in $\boldsymbol{v}$, and $\dot{\boldsymbol{\omega}}^v_{vr}$ is the corresponding angular acceleration. Cross products with a leading vector w are expressed in matrix form as the skew-symmetric matrix parametrization is denoted as $[\boldsymbol{\omega}^v_{vr}]_\times$.

Frame $\boldsymbol{v}$ is chosen to coincide with $\boldsymbol{s}$, such that $\boldsymbol{R}_{sv} = I_{3\times3}$ and $\boldsymbol{t}^v_{sv} = \boldsymbol{0}$.

3.2.3 Depth Gauge Data

Most underwater vehicles are equipped with a depth gauge that measures water pressure and computes the depth. We chose to position our underwater environment at a depth of 80m. This depth is represented as a 64bit float and is generated at 200Hz.

3.3. Pose Sequence Generation

The generation of physically realistic pose sequences is the third major building block of the VAROS system. To generate these pose sequences, a user-defined series of waypoints containing the key poses of the vehicle poses is used. A weighted reference based using vehicles current pose, current and next waypoint is then used as a reference to the vehicle control system (VCS).

3.3.1 Way Point Generation

The waypoints are generated in the interactive `Blender` environment; `Blender` allows for placing any object, here our vehicle with an attached camera, at a desired position and orientation in the environment of the keyframe animation system.

3.3.2 Motion Model

The motion of the vehicle is generated and represented as a sequence of poses (location and orientation). In order to be able to generate simulated IMU measurements, the sampling rate must be significantly higher than the video sampling rate. The poses are generated by virtually steering the simulated underwater vehicle to follow the waypoints sequentially. For maximum realism of the resulting motion, a handover scheme has been developed that softly dissolves between waypoints such that waypoint $N+1$ is already considered before being done with waypoint N; this leads to the result that the waypoints are not exactly reached, but the simulated vehicle course only passes by closely, yielding a smooth motion without jerk and without discontinuities in the acceleration. This point is often disregarded in motion simulators (and also in early animated movies). When a waypoint has a strict pose requirement, two waypoints are placed close together which constrains the motion.

For the vehicle, we use a dynamic model that assumes the vehicle to be a spherical body with mass M (cf. figure 4) which is moving as the result of translational thrusts $F_{u,i}$ and rotational moments τ_i, where i refers to the x, y or z-axis.

In our simplified model, the rotation and translation are not coupled, but since the way-points contain both a location as well as a desired orientation, the vehicle is steered in the desired way. Inertial and drag forces, both of them translational and rotational, act on the vehicle and the actual translational and rotational motion results from the interaction of thrust forces and moments with inertia and drag. The resulting motion is realistic despite the very simple model.

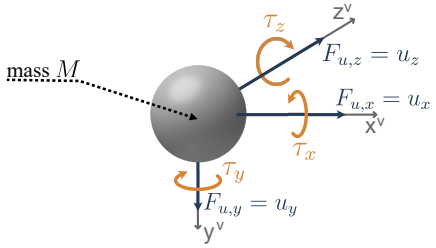

Figure 4. Vehicle model used in the simulations, where $\tau_x/\tau_y/\tau_z$ and $u_x/u_y/u_z$ are the torques and input forces around each of the $x-$, $y-$ and z-axes.

3.3.3 Vehicle Control System

The vehicle is controlled by a linear quadratic regulator (LQR) [19]. This allows for easy tuning of the regulator parameters which control the inputs for translation and rotation to get the desired motion characteristics. Using an LQR allows us to control the cost of the error in each state or input directly through weighting matrices, which gives us more control in fine tuning the vehicles behavior.

In a real system, the LQR would control the voltage to a motor. A voltage change influences the thrust by changing the revolutions per minute (RPM) of the motor with a very small time delay. This effect is modeled through the use of a fast second-order system placed in between the LQR output and the acceleration input for the vehicle model. In a real UW drive system, the thrusters are themselves dynamical systems with inertial moments; for that reason, a second-order low-pass was added to the dynamic model.

3.4. Simulation Limitations

3.4.1 Rendering

The raytracing renderer Cycles has several limitations. The camera and light models do not account for refraction effects between air in the internal casing, glass and the water. Furthermore, Cycles uses only the HG phase function (see section 3.2.1) to calculate light scattering in the water volume, with no separate functions for forward and backward scattering, limiting the customizability of the turbidity in the water volume. In addition, small-angle forward scattering resulting in image blur is not modeled; this could be approximated by applying image blur in post processing.

3.4.2 Vehicle model

The used vehicle model is relatively simple and assumes a spherical vehicle body with independent translation and rotation. However, since the way-points consist of locations and vehicle orientations, this limitation is of minor importance. So far, the control loop operates on the assumption that there are no random disturbances on the vehicle motion, thus external forces such (e.g. currents) are currently not considered. Both these limitations can be addressed in further versions.

4. The VAROS Dataset: Structure and Usage

The VAROS dataset utilizes the described underwater simulation environment complete with both simple and complex 3D models with physically realistic textures, the possibility to place cameras and lights where needed, and most importantly, physically realistic path-traced images.

In the following we describe the main structure elements of the VAROS dataset: data structure, data formats, and naming conventions.

4.1. Data Format

Pose data for the vehicle pose for every sensor sample (camera, IMU, depth) is provided in the world frame $\boldsymbol{w}$. Each sensor type comes with a file with ground truth pose information in the corresponding sensor folder. If the stream file format, discussed later in this section, is used, the ground truth information will also be integrated.

RGB images: Lossless RGB images are exported from the simulation as 16-bit OpenEXR files. After post-processing, the images are exported as lossless 8bit standard RGB PNG files for the benchmark sequence, corresponding to the standards used by KITTI [13], and LiU [21] datasets. These images can easily converted to monochromatic 8bit PNG images that are used by datasets such as the underwater dataset AQUALOC [12] and the aerial dataset EuRoC [7]. The color management settings for the export of RGB images are shown in table 3, and information for the RGB images included in the dataset is shown in table 4.

Table 3. The different filetypes exported from `Blender` prior to post-processing.

Image type	Filetype	Channel depth	View transform	Sequencer
RGB	OpenEXR	16bit	Not used	Not used
Normal	PNG	8bit	Raw	Linear
Depth	PNG	16bit	Raw	Linear

Table 4. The images contained in VAROS dataset version 1 and their corresponding filetypes, channel bit depth and number of channels per image.

Image type	Filetype	Channel depth	Num. channels
RGB, underwater	PNG	8bit	3, RGB
RGB, uniform lighting	PNG	8bit	3, RGB
Normal	PNG	8bit	3, RGB
Depth	PNG	16bit	1, Grayscale

Surface normal vector images: Surface normal vectors are stored in PNG files with 8bit channel depth (already discussed in section 3.2.1).

Depth images: The depth images are given as 16bit PNG files. Due to the limited visibility in the underwater scene we set $d_{end} = 25\text{m}$, resulting in a resolution of 0.381mm from equation (1).

Pose data: The pose data consists of the vehicle states obtained by driving the vehicle dynamic model with the control output of the VCS. The data can be transformed into the reference frame $\boldsymbol{r}$ if needed. The poses of sensors or cameras attached to the vehicle can be derived from these measurements using the transformation matrices for each sensor. The position along the x-axis of the vehicle frame $\boldsymbol{v}$ in the world frame $\boldsymbol{w}$ is given as $s^w_{x,vw}$. Likewise, $\theta^w_{x,vw}$ is the x-component of the vehicles orientation in the world frame $\boldsymbol{w}$ expressed in $\boldsymbol{w}$. This data is stored in a comma separated value (CSV) file, where each row contains 7 columns:

$$\text{timestamp}[ns],\ s^w_{x,vw},\ s^w_{y,vw},\ s^w_{z,vw},\ \theta^w_{x,vw},\ \theta^w_{y,vw},\ \theta^w_{z,vw}.$$

IMU measurements: IMU measurements are computed at the overall systems clock rate of 200Hz. This includes acceleration and angular velocity of the vehicle frame $\boldsymbol{v}$ relative to the reference frame $\boldsymbol{r}$ expressed in the sensor frame $\boldsymbol{s}$, $\boldsymbol{a}^s_{vr}$ and $\boldsymbol{\omega}^s_{vr}$ respectively. These vectors consists of the following components:

$$\boldsymbol{a}^s_{vr} = [a^s_{x,vr},\ a^s_{y,vr},\ a^s_{z,vr}]^\top,$$

$$\boldsymbol{\omega}^s_{vr} = [\omega^s_{x,vr},\ \omega^s_{y,vr},\ \omega^s_{z,vr}]^\top.$$

Every IMU measurement is then exported as one line in the CSV file such that each row reads:

$$t,\ a^s_{x,vr},\ a^s_{y,vr},\ a^s_{z,vr},\ \omega^s_{x,vr},\ \omega^s_{y,vr},\ \omega^s_{z,vr}$$

where t denotes the system clock timestamp in nanoseconds for a given measurement.

Depth gauge data: Depth gauge measurements are given at the same sample frequency as the IMU at 200Hz. The depth of the vehicle frame $\boldsymbol{v}$ in the scene expressed in the fixed world-frame $\boldsymbol{w}$, $\tilde{d}_v^w$, and it is exported as a CSV file of the following form:

$$\text{timestamp}[ns],\ \tilde{d}_v^w.$$

4.1.1 Image Naming Convention

For enumeration in the dataset file names, ASCII encoding using $[0 \rightarrow 9,\ a \rightarrow z,\ A \rightarrow Z]$ is used, allowing for 62 different values for a single character slot. The image naming convention is as follows:

seq[SeqNumber]_veh[VehicleNumber]_cam[CameraType] [CameraNumber]_[ImageType]-########.[Ext]

where each ID placeholder, in brackets [], is explained in table 5. The folders containing the images follows the same ASCII enumeration convention, where one character is assigned per folder with the encoding in table 6. This convention and the overall folder structure is shown in fig. 5.

Table 5. The encoding scheme used for generating the VAROS dataset filenames.

Number ID	Description	ASCII chars
SeqNumber	Unique sequence ID	2
VehicleNumber	Vehicle ID for multi-vehicle support	1
CameraType	Mono, stereo left, stereo right and more	1
CameraNumber	Enumeration of a specific CameraType	1
ImageType	Datatype stored in the image	1
########	Image number	8 digits
Ext	Image file extension	-

Table 6. Image folders encoded with an ASCII character and what the respective folder contains.

Folder ASCII code	Contents
A	RGB image with water
B	RGB image without water
C	World space normal image
D	Depth map with distance to each pixel

4.1.2 Folder Structure

The data structure of the dataset is made with inspiration primarily from the LiU [21] and EuRoC [7] datasets. KITTI [13] and AQUALOC [12] as standard were used additionally for the format of the dataset data to ensure maximum compatibility with already existing datasets. This folder structure is shown in figure 5.

Planned future file organization: The Linköping University (LiU) stream format [21] was proposed in 2013 as a framework for storing larger datasets in an efficient manner

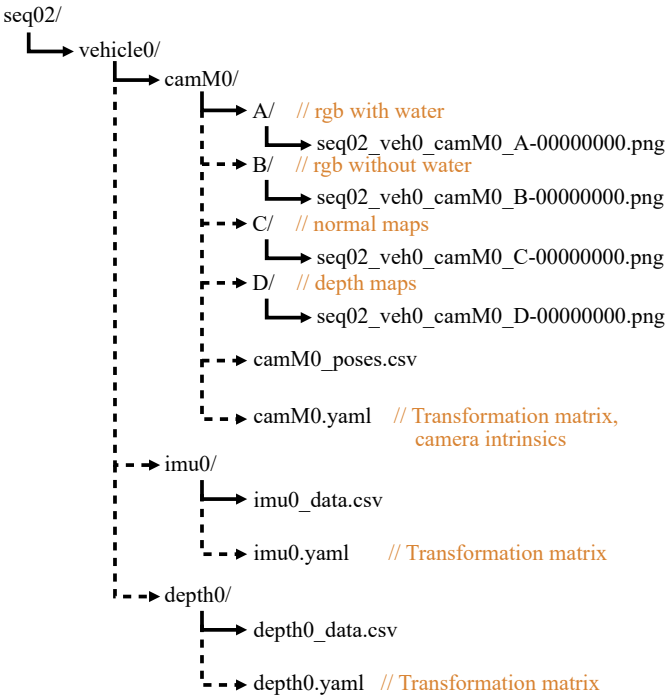

Figure 5. Folder structure of the included data in the VAROS dataset version 1.

in sequential stream files. We are currently preparing a transition to a variant of this stream format which, given a corresponding stream file reader software, will simplify the reading of the multi-sensor data in the correct temporal order. The envisaged stream format is a simplification of the LiU stream file format, consisting of a binary stream file with the structure in table 7. The 'payloads' of this stream file consists of time-stamped pointers to externally stored images, and internally stored time-stamped sensor data from the IMU, depth sensor and ground truth pose data.

Table 7. A single message in the LiU stream file format [21].

Messagestart	ID	Length of Payload	Payload
4x uint8 '$BST'	uint32	uint32	Dependent on message

5. Conclusion

We have created VAROS, a framework for generating synthetic underwater datasets with physically realistic images, auxiliary sensor data, and ground truth pose data. Hence we provide a complete set of synthetic data which enables systematic testing of underwater computer vision methods. At the moment of submitting this paper (August 2021), the dataset consists of one sequence, VAROS dataset version 1 with 4713 images at 10fps. Further sequences are in preparation. The data is available at the following web location https://www.ntnu.edu/arosvisiongroup/varos.

References

[1] Derya Akkaynak, Tali Treibitz, Tom Shlesinger, Yossi Loya, Raz Tamir, and David Iluz. What is the space of attenuation coefficients in underwater computer vision? In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4931–4940, 2017.

[2] Timothy D Barfoot. State estimation for robotics. Cambridge University Press, Toronto, 2021.

[3] Austin Beaulier. Vasquez rocks (photogrammetry). Available at https://skfb.ly/6Rqon, 2021.

[4] Daniel Biedermann, Matthias Ochs, and Rudolf Mester. Congrats: Realistic simulation of traffic sequences for autonomous driving. In 2015 International Conference on Image and Vision Computing New Zealand (IVCNZ), pages 1–6, 2015.

[5] Daniel Biedermann, Matthias Ochs, and Rudolf Mester. Evaluating visual adas components on the congrats dataset. In 2016 IEEE Intelligent Vehicles Symposium (IV), pages 986–991, 2016.

[6] Blender Online Community. Blender - a 3D modelling and rendering package. Blender Foundation, Blender Institute, Amsterdam, 2021.

[7] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. The International Journal of Robotics Research, 35(10):1157–1163, 2016.

[8] Fabio Cozman and Eric Krotkov. Depth from scattering. In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 801–806. IEEE, 1997.

[9] Creative Commons. Attribution 4.0 international. https://creativecommons.org/licenses/by/4.0/legalcode, 2021.

[10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, Proceedings of the 1st Annual Conference on Robot Learning, volume 78 of Proceedings of Machine Learning Research, pages 1–16. PMLR, 13–15 Nov 2017.

[11] Amanda C Duarte, Guilherme B Zaffari, Rômulo Thiago S da Rosa, Lucas M Longaray, Paulo Drews, and Silvia SC Botelho. Towards comparison of underwater slam methods: An open dataset collection. In OCEANS 2016 MTS/IEEE Monterey, pages 1–5. IEEE, 2016.

[12] Maxime Ferrera, Vincent Creuze, Julien Moras, and Pauline Trouvé-Peloux. Aqualoc: An underwater dataset for visual–inertial–pressure localization. The International Journal of Robotics Research, 38(14):1549–1559, 2019.

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. International Journal of Robotics Research (IJRR), 2013.

[14] Mohit Gupta, Srinivasa G. Narasimhan, and Yoav Y. Schechner. On controlling light transport in poor visibility environments. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4319–4326, 2008.

[15] Vladimir I Haltrin. One-parameter two-term henyey-greenstein phase function for light scattering in seawater. Applied Optics, 41(6):1022–1028, 2002.

[16] Intel. Open image denoise. Available at https://www.openimagedenoise.org, 2018.

[17] Jules S Jaffe. Computer modeling and the design of optimal underwater imaging systems. IEEE Journal of Oceanic Engineering, 15(2):101–111, 1990.

[18] N Jerlov. Irradiance optical classification. In Optical Oceanography, pages 118–120. Elsevier, 1968.

[19] Mikhail V Khlebnikov, Pavel S Shcherbakov, and Vladimir N Chestnov. Linear-quadratic regulator. i. a new solution. Automation and Remote Control, 76(12):2143–2155, 2015.

[20] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2149–2154, Sendai, Japan, Sep 2004.

[21] P. Koschorrek, T. Piccini, P. Öberg, M. Felsberg, L. Nielsen, and R. Mester. A multi-sensor traffic scene dataset with omnidirectional video. In 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 727–734, 2013.

[22] Angelos Mallios, Pere Ridao, David Ribas, Marc Carreras, and Richard Camilli. Toward autonomous exploration in confined underwater environments. Journal of Field Robotics, 33(7):994–1012, 2016.

[23] Musa Morena Marcusso Manhães, Sebastian A Scherer, Martin Voss, Luiz Ricardo Douat, and Thomas Rauschenbach. Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation. In OCEANS 2016 MTS/IEEE Monterey, pages 1–8. IEEE, 2016.

[24] Curtis D Mobley and Charles D Mobley. Light and water: radiative transfer in natural waters. Academic press, 1994.

[25] Theodore J Petzold. Volume scattering functions for selected ocean waters. Technical report, Scripps Institution of Oceanography La Jolla Ca Visibility Lab, 1972.

[26] Mario Prats, Javier Perez, J Javier Fernandez, and Pedro J Sanz. An open source tool for simulation and supervision of underwater intervention missions. In 2012 IEEE/RSJ international conference on Intelligent Robots and Systems, pages 2577–2582. IEEE, 2012.

[27] A. Sedlazeck and R. Koch. Simulating deep sea underwater images using physical models for light attenuation, scattering, and refraction. VMV 2011 - Vision, Modeling and Visualization, pages 49–56, 2011.

[28] Dave Shreiner, Graham Sellers, John Kessenich, and Bill Licea-Kane. OpenGL programming guide: The Official guide to learning OpenGL, version 4.3. Addison-Wesley, 2013.

[29] Yifan Song, David Nakath, Mengkun She, Furkan Elibol, and Kevin Köser. Deep sea robotic imaging simulator. In ICPR Workshops (2), pages 375–389, 2020.