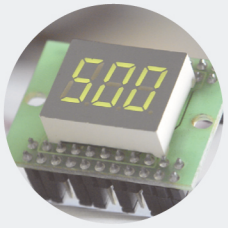
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Ambiente de Simulação para o Sistema de Exploração Robótica Subaquática UNEXMIN

DENYS SYTNYK
novembro de 2018

POLITÉCNICO
DO PORTO

Simulation Environment for the UNEXMIN Underwater Robotic Exploration System

Denys Sytnyk,
Nº 1121150

Master in Electrical and Computer Engineering
Branch of Autonomous Systems

November 15, 2018

Dissertation for partial satisfaction of the requirements of the Master in Electrical and Computer Engineering

Candidate: Denys Sytnyk,
Nº 1121150

Supervisor: Alfredo Manuel Oliveira Martins

Master in Electrical and Computer Engineering
Branch of Autonomous Systems

This page was intentionally left blank.

Aos meus pais, irmã e todos os outros...

Abstract

Underwater mines exploration is a valued, complex, expensive and time-consuming task. The unstable nature of the underwater environment with lack of visibility and the existence of obstructions create the need for complex navigation software which requires numerous missions and hardware/software validations. When testing and verifying control algorithms for such an operation, a simulation environment can be a very helpful tool. This also includes tools for the development of unmanned vehicle software, algorithm benchmarking and system preliminary validation.

The objective in this thesis was to start the development of a simulation platform that can be used when developing and testing control systems for AUV operations. The simulator will include a dynamic model of an AUV in addition to complex world and sensor models such as DVL, IMU, Multibeam, Mechanical Scanning Imaging Sonar (MSIS), cameras, SLS and others. The simulated world includes water graphics, mine meshes, underwater visibility, currents, and hydrodynamics. Control of the robot in simulation is performed by keyboard or joystick over thrusters. The platform must be universal, such that users can implement their own algorithms easily and get immediate simulation results without needing to implement a complete control system. There should also be an easy transition between testing the control system on the simulated AUV and applying it to the real AUV.

Robot Operating System (ROS) and Gazebo were used in the development of the platform. The platform with sensors and navigation was validated with real-world tests comparison.

Keywords:
Underwater Simulation, exploration, simulation, Gazebo, mines, AUV, UNEXMIN, ROS.

This page was intentionally left blank.

Resumo

A exploração de minas subaquáticas é uma tarefa valiosa, complexa, dispendiosa e demorada. A natureza instável do ambiente subaquático, com falta de visibilidade e a existência de obstruções, cria a necessidade de *software* de navegação complexo, qual requer inúmeras missões e validações de *hardware/software*. Ao testar e verificar os algoritmos de controle para tal operação, um ambiente de simulação pode ser uma ferramenta muito útil. Isto também inclui ferramentas para o desenvolvimento de software de veículos não tripulados, *benchmarking* de algoritmos e validação preliminar do sistema.

O objetivo desta tese foi iniciar o desenvolvimento de uma plataforma de simulação que possa ser usada no desenvolvimento e teste de sistemas de controle para operações de AUV. O simulador incluirá um modelo dinâmico de um AUV, além de modelos complexos do mundo e sensores, como DVL, IMU, Multibeam, MSIS, câmeras, SLS e outros. O mundo simulado inclui gráficos de água, malhas de minas, visibilidade subaquática, correntes e hidrodinâmica. O controle do robô é realizado por teclado ou *joystick* sob as dinâmicas de propulsão. O simulador deve ser universal, de modo que os usuários possam implementar seus próprios algoritmos facilmente e obter resultados imediatos de simulação sem a necessidade de implementar um sistema de controle completo. Também deve haver uma transição fácil entre testar o sistema de controle no AUV simulado e aplicá-lo ao AUV real.

ROS e Gazebo foram usados no desenvolvimento da plataforma. A plataforma com sensores e navegação foi validada com comparação de testes reais.

Palavras-Chave:
Simulação Submarina, exploração, simulação, Gazebo, minas, AUV, UNEXMIN, ROS.

This page was intentionally left blank.

Contents

This page was intentionally left blank.

List of Figures

List of Tables

This page was intentionally left blank.

List of Acronyms

AUV Autonomous Underwater Vehicle

API Application Programming Interface

Co3-AUVs Cooperative Cognitive Control for Autonomous Underwater Vehicles

DARPA Defense Advanced Research Projects Agency

DART Dynamic Animation and Robotics Toolkit

DOF Degrees of Freedom

DRC DARPA Robotics Challenge

DVL Doppler Velocity Log

ESC Electronic Speed Controller

FPS Frames Per Second

GNSS Global Navigation Satellite System

GPS Global Positioning System

ICRA International Conference on Robotics and Automation

IMU Inertial Measurement Unit

ISEP Instituto Superior de Engenharia do Porto

LSA Laboratório de Sistemas Autónomos

MORSE Modular OpenRobots Simulation Engine

MOOS Mission Oriented Operating Suite

MSIS Mechanical Scanning Imaging Sonar

NASA National Aeronautics and Space Administration

ODE Open Dynamics Engine

OSG OpenSceneGraph

OSRF Open Source Robotics Foundation

RAUVI Reconfigurable AUV for Intervention Missions

ROCK Robot Construction Kit

ROS Robot Operating System

ROV Remotely Operated Vehicle

SDF Simulation Description Format

SLS Structure Light Scanner

SRC Space Robotics Challenge

UHRI Underwater Human-Robot Interaction

URDF Unified Robot Description Format

USARSim Unified System for Automation and Robot Simulation

UUV Unmanned Underwater Vehicle

UWSim UnderWater Simulator

VBS Variable Ballast System

XML eXtensible Markup Language

YARP Yet Another Robot Platform

Chapter 1

Introduction

The development of underwater robotic systems poses a large set of challenging problems. These include the development of navigation, perception and autonomous control systems.

In order to develop and validate advanced control algorithms there is a need for a robust simulation environment to verify and test algorithms. In this master's thesis, a simulation platform for Autonomous Underwater Vehicle (AUV) systems was developed. The simulation platform was developed based on the middleware ROS and Gazebo.

In real world missions, AUV's may fail in the vehicle control or provide misleading information about its sensors. Any of those situations may result in an unrecoverable vehicle. Also, partial knowledge about the underwater scenario is required to perform the real experiments. To avoid several frequent underwater robotics problems, in most cases, the most promising course of action is the use of a simulation.

Simulators are a relevant tool for the development of autonomous vehicle software and to process various experiments that may occur in real life. It is important for a simulator, the inclusion of a proper dynamics simulation, visual realism, a large variety of sensors and real-time sensor-based control. Simulations help programmers to test various scenarios and situations, that may later, improve the performance of AUV, in real life experiments.

This work was motivated by the develpment of the underwater robotic systems for mine exploration at INESC-TEC.

In Europe exist various mines that were forgotten and are now flooded, inaccessible to humans. To simulate their exploration and mapping, the developed simulator was tested, in a scenario with similar conditions, a flooded underground mine (tunnels).

The first part of this work consists in the simulation of the various underwater sensors

in the robotic simulator and in a comparative study between the outputs given by the simulated sensor and by a real sensor under similar conditions. Then follow the inclusion of proper dynamics, hydrodynamics, and environment for robot navigation.

Underwater navigation is a limited and complex activity due to a challenging environment and to a limited type of sensors. Highly unstable environments demand the use of all the information that can be gathered. With the simulation platform it is possible to recreate various environmental scenarios and be more prepared for real missions.

INESC-TEC has been developing autonomous underwater robots for marine applications and more recently is involved in the development of specific robotic solutions for exploration and exploration of inland flooded mines, under the European research projects VAMOS[2] and UNEXMIN[3]. The first aims at the development of efficient and sustainable robotic technologies for mining in flooded open pit mines and the second on the development of autonomous robotic solutions for the mapping and exploration of flooded underground mine caves and galleries.

Those scenarios (and the UNEXMIN one in particular) pose difficulties in system test and development and also have some differences in relation to the typical undersea scenarios. In the above-described projects, there exists the need to perform perception and navigation algorithms in a simulation environment, thus leading the motivation for this thesis.

1.1 Background and motivation

Programming an AUV to perform autonomous tasks can be a very time-consuming process, and it is highly dependent on test missions and validations. Introducing the possibility of realizing code validations without the need for new missions can save a lot of time, and therefore also a lot of resources. A specific, independent and more predictable operation can be accomplished through the use of a simulation platform. The sensor outputs can be accurate enough for initial tests and give similar output as real AUV in more controlled situations. A simulation tool can be very helpful in the process of developing autonomous control systems. The control can be tested on a simulation model, and therefore be more ready for real missions.

In order to develop a control system, a reliable perception and navigation system have to be included. The perception of the environment is an important task and provide the developers the crucial information for software development. The navigation system is one of the most important systems of the robot. The environment and robot information are crucial for navigation. A simulation tool can be very helpful and simulate

the information of the environment as well of the robot.

The environment of simulation can easily be changed, therefore improving the robustness of various systems/sensors in all situations. To model all forces and dynamics accurately is, however, a very difficult task, and some assumptions must always be taken. Thus leading the motivation of this dissertation.

1.2 Objectives

The platform can be used for testing software for AUV operations and missions without performing real-life missions with AUV. The simulated sensor output will be developed with the same messages as real AUV with the use of ROS. Therefore, the simulation platform must be able to simulated AUV motions based on thruster inputs and hydrodynamics. This includes realistic estimations of the hydrodynamic forces acting on the AUV, as well as a realistic estimation of the thrust forces for a given input of each propeller. Such a simulation platform will give an opportunity for developers to test their algorithms and get immediate results from the simulation. The typical user will use the simulation to test algorithms with no need of complex installation/preparation process. The platform will be developed such that the user easily can understand the platform and how to implement new code without comprehensive knowledge of underlying software and frameworks. When the development is done, the software will be tested with the real AUV in the loop.

As referred above, the purpose of this study is to find the best solutions to reduce the problems related with underwater experiments, through a realistic simulation, for that, a simulator capable of replicate the rough conditions underwater, is necessary:

- Low visibility: Floating particles drastically reduce the visibility of underwater experiments;
- Currents: The flows of water beneath the surface exert forces on the vehicle, capable of throwing it out of its trajectory;
- Surface Reflections: Certain sensors provide incorrect data outputs due to reflections on the surface of the water;
- Position Determination: Unlike on aerial, terrestrial or surface applications, exact position determination on underwater applications it's one of the hardest tasks to achieve due to the Global Positioning System (GPS) being inoperable underwater;
- Multiple specific sensors to underwater environments;

1.3 Dissertation Structure

The dissertation structure follows.

In **Chapter 2** the problem formulation for the simulator is present. A brief description of UNEXMIN project and simulation requirements are detailed.

In **Chapter 3** state of the art is presented. This includes simulators comparison and related works.

In **Chapter 4** the simulation environment is described. The ROS middleware, the simulation platform, and the interaction between them is illustrated in this chapter.

In **Chapter 5** the UNEXMIN simulator is described. Simulation platform structure, architecture and all developed components of the simulation are presented in this chapter.

In **Chapter 6** the results are present. This include laboratory tank tests, real vs simulated sensors data, navigation and mapping.

In **Chapter 7** the conclusions and future work, with a brief discussion, are present.

Chapter 2

Problem formulation

2.1 UNEXMIN Project

Thousands of mines left from the previous mining and now inaccessible are present in Europe, with many of those mines flooded and information of their layout is aged or lost. Those mines may still have considerable amounts of essential raw materials. With non-invasive methods, UNEXMIN[3] project will provide autonomous 3D mine mapping for gathering valuable geological, mineralogical and spatial information. The project is being supported by a science and technology merger of deep-sea robotics solutions with user's requirements from the mining industry. The main objective consists in the development of the multi-robot platform, consisting in three robots, that will be capable of operating without remote control.

The project started in 1 of February 2016 with the duration of 45 months.

Goals of UNEXMIN project are:

- Design and build of a multi-platform Robotic Explorer for autonomous 3D mapping of flooded deep mines;
- Demonstration of the operations of the prototypes at numerous abandoned underwater mining sites;
- Development of an open-access platform for technology transfer and further development between stakeholders;
- Development of a research roadmap in support of further technology development;
- Development of commercial services for exploiting the technology.

The UNEXMIN project is very challenging as it consists of a multi-robotic platform that must operate autonomously within flooded mines with a complex environment. The important aspect of UNEXMIN project is the development of non-invasive surveying technologies. This is possible with the use of imaging and other non-invasive sensors.

UNEXMIN can improve significant economic aspects, such as increasing Europe's mineral potential, reducing exploration costs for mining operations and help document unique mining sites.

2.2 Environment

The complex underground layout, topology, and geometry of most underground mines that take part of UNEXMIN project, make it impossible to do any surveying by conventional or remotely controlled equipment. One of these examples is the usage of human divers, which can prove unfruitful and even lethal in harsh deep mine conditions.

Underwater mine environment is rich, complex and composed of numerous materials/obstructions. This includes dense mine walls, leftover obstructions, complex mine layouts, collapsed walls, tight spaces and possibly aggressive water properties (such as pH or temperature). Those extreme environment conditions imply the use of robot onboard sensors and autonomous navigation. Communication with the robot would be only possible near the deployment area.

The multi-robot platform and all its components were tested and built to suit these conditions. Real-life experiments are being performed in four mine test sites to validate all conditions.

The project defined mines that will be used as test sites are:

- Kaatiala Mine, Finland;
- Idrija Mine, Slovenia;
- Urgeiriça Mine, Portugal;
- Ecton Mine, UK.

Some of those mines nobody has explored for a long period of time.

2.3 Multi-Robotic platform

The multi-robotic platform will be formed by three robots, with a defined set of instruments. Each robot will be capable of operating without any remote control.

The first UNEXMIN robot and the one this thesis is focused on is UX-1a spherical robot, represented in figure 2.4. This robot complies the necessary equipment to obtain geoscientific and spatial data, movement, control, 3D mapping, and perception. The UX-1a robot will gather data that is very difficult to obtain through other means.

The main UX-1a robot characteristics are:

- Maxim operation depth: 500m;
- Diameter: 0.6m;
- Weight: 112kg;
- Power consumption: 150-300W;
- Maximum speed: 1-2km/h;
- Autonomy: up to 5 hours.

Physical robustness, multi-robot cooperation, and self-diagnosis are requirements for UNEXMIN robots that could assure the performance in a complex environment with reliability and safety. The robot needs to be able to fit into small mine openings, resist high depths pressure, avoid being trapped, and avoid equipment damage. The measurement, materials, and sensors are builds according to those conditions. In normal conditions, the robot is neutrally buoyant for less power consumption and improved control capabilities.

To improve the movement inside mine environment, buoyancy control and a pendulum system will be included. Movement or each robot is based on changing the center of mass, buoyancy, pendulum-based angle control and thrusters that will contribute to the stable control of the robotic system in such environments.

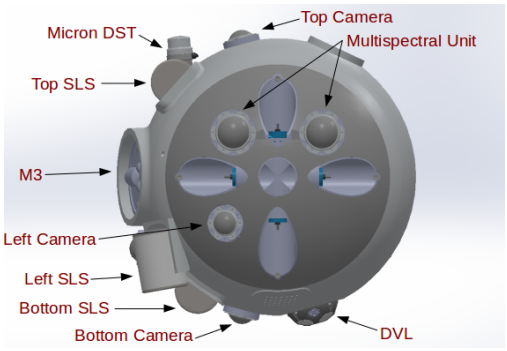


Figure 2.1: UX-1 left side description.

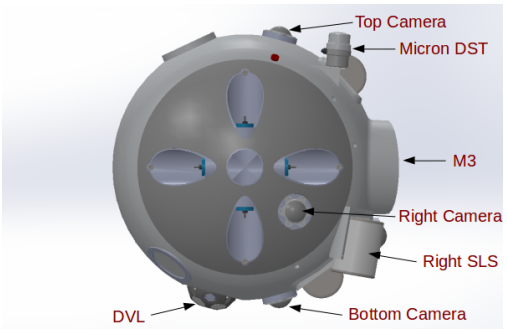


Figure 2.2: UX-1 right side description.

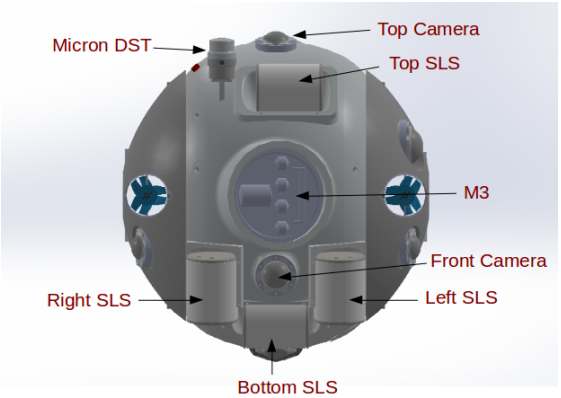


Figure 2.3: UX-1 front description.

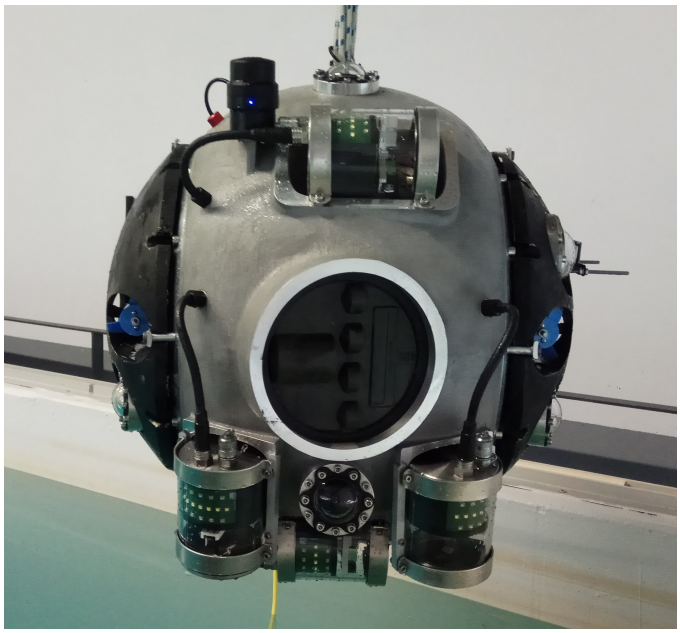


Figure 2.4: UX-1 robot.

2.4 Sensors

UX-1 robot merges multiple sets of sensors for navigation, perception, exploration and scientific samplings. To travel securely and with precision, UX-1 will have positioning, navigation, control and 3D mapping systems. The list of sensors include:

- Sub-bottom profiler;
- Water sampler;
- Conductivity and pH measuring;
- Magnetic field measuring unit;
- 4 LED and Laser projectors (SLS systems);
- Multispectral camera;
- Inertial Measuring Unit (KVH 1775[4]);
- Doppler Velocity Logger (Nortel 1000 DVL[5]);
- 8 Thrusters;

Figure 2.5: SLS sensor system.

Figure 2.6: Cameras system.

- Multibeam sonar (Kongsberg M3[6]);
- Scanning sonar (Tritech Micron DST[7]);
- 5 Cameras.

A brief description of a UX-1 robot is represented in figures 2.1, 2.2, and 2.3. The SLS system developed in INESC-TEC robotics laboratory, consisting in 4 projectors and 5 cameras is depicted in figures 2.5 and 2.6.

2.5 Simulation requirements

The simulation platform is necessary and very beneficial for UNEXMIN project. The complex environments described above can be simulated and modified for every specific and required test. Those can include narrow mine scenarios for maneuver and navigation tests of the vehicle, different shaped mines and objects for sensors information prepossessing and best position design on the robot, 3D mapping of the environment, lighting, underwater visualization, perception, etc.

With the simulation help, it is possible to improve the vehicle capabilities and perception of the environment. Starting from the initial design of the robot, it is possible to test various sensors positions and/or thrusters configuration, to give optimal performance, navigation capabilities, and environmental perception. With an already defined robot design, it is possible to replicate an identical model of the robot inside of the simulation platform. The dynamic model of the robot and hydrodynamics of the underwater environment also contribute to a more realistic approach and reliable simulator. This also allows for the development and test of control algorithms. Besides the robot, the

environment can also be replicated with high definition inside the simulation, and with the possibility of distinct material properties in the simulation, the final results of the testing possibilities of the simulator are even more extended.

The simulation platform that combines realistic and detailed robot replica with the detailed environment may be useful for software development. With the possibility of integrating a simulated robot in every possible situation of a complex environment, the developed software may be tested and easily improved without the need for real-life tests, which are very expensive and time-consuming.

UNEXMIN robots are all completely autonomous, which implies the need of complex and reliable software. The software always demands validation and numerous experimental tests. For this purpose, the highly detailed simulation platform may be very helpful and improve greatly the quality of the developed software for the real robots. The hardware in loop simulations increases, even more, the reliability and compatibility of the developed software for real robot. The compatibility between developed software tested in simulation and in real robot is crucial, as it can reduce the time for development and also improve simulated information. In order to perform all the abilities described above, a versatile simulation platform must be developed.

The UNEXMIN project consists of unique environments, sensors, and robot. In order to accomplish every requirement of the UNEXMIN project, the simulation platform must posses next abilities:

- **Open source:** The simulator platform must be open source with capabilities of integration of new modules;
- **ROS:** All UNEXMIN robots have ROS middleware. The simulation platform must have compatibility with ROS and possibility of the software integration between ROS and the simulator;
- **Underwater sensors:** UNEXMIN robot have many unique sensors for all purposes. The simulator must include the simulation of all important sensors and have similar output data;
- **3D mapping:** For a software implementation, the 3D mapping capabilities of the simulator are crucial and must be included;
- **Underwater visualization:** For perception and visual aspect of the simulator, the underwater visualization parameter aggravates or improve the visibility of the environment;

- **Hydrodynamics and robot dynamics:** The dynamic model of the robot and environment are most essential for maneuvers and navigation models that will be implemented in software for real robot.
- **Hardware in Loop:** Hardware in loop simulation is a desired function;
- **Software development and compatibility:** The simulator must be capable of the development of the new software's that can be transferred to the real robot and have the same structure (names, data information, data processing, communication).
- **Real time:** It is desirable that simulation can be run at real time, thus replicate the real operating conditions and allowing for the test of software (without the additional modifications and hardware).
- **Expansion possibilities:** As UNEXMIN project have very unique requirements, it would be helpful to have a simulation platform that allows the creation of new modules for every requirement.

Chapter 3

State of the Art

Nowadays, in numerous simulators [8][9][10][11][12] proper dynamics, visual realism, wide variety of sensors, interfaces, modeling tools and real-time sensor-based control are major factors considered and included. Robotics simulators have been presented in many projects and conferences, some of the most relevant are DARPA Robotics Challenge (DRC), International Conference on Robotics and Automation (ICRA), RoboCup and more recently in Space Robotics Challenge (SRC), a NASA Centennial Challenge. From 2D low fidelity simulators such as Player/Stage to 3D dynamic simulators such Gazebo[12], Modular OpenRobots Simulation Engine (MORSE)[8] or Unified System for Automation and Robot Simulation (USARSim)[13] these systems allow for simulation of mobile robots and the interactions with the environments.

Simulators focused on marine robotics[14][15][16], have been developed either for specific simulation scenario requirements or as more or less generic tools under multiple European research projects such as Cooperative Cognitive Control for Autonomous Underwater Vehicles (Co3-AUVs)[17], Reconfigurable AUV for Intervention Missions (RAUVI) or TRIDENT with UnderWater Simulator (UWSim)[18][19].

This research was focused on open source simulators compatible with ROS, with these requirements, three most popular simulators were selected for further analysis: Gazebo, UWSim, and MORSE. In order to obtain a better comparison between them, different criteria were defined:

- Physical Fidelity: Capability for a correct interaction between the robot and the environment/objects, simple actions such as pushing, picking or grasping objects involve a complex calculation of simulated forces and collisions.

Table 3.1: Simulator Comparison

Simulator	3D rendering engine	Physics engine	Programming language	Middleware support	Operating System	Formats support	Open Source	Adequate Documentation	Required Knowledge	Visual Fidelity
Gazebo	ogre3D	ODE/Bullet/ Simbody/DART	C++	ROS/Player/ Sockets	Linux/MacOS X/Windows	SDF/URDF	Yes	High	Medium	Low
UWSim	OSG	Bullet	C++	ROS	Linux	URDF	Yes	Low	Medium	High
MORSE	Blender	Bullet	Python	ROS/YARP/ Pocolibs/MOOS Sockets	Linux/MacOS X	URDF	Yes	Medium	High	High

- Sensor Modeling: Capability of the software in the simulation of multiple sensors.
- Required Knowledge/Experience: The amount of knowledge/experience required to work with the simulator software in a proficient way.
- Visual Fidelity: Details such as water reflection, refraction and sediments flotation, are taken into account;
- Documentation/Support: Inclusion of proper documentation and tutorials, with proper customer support.

This and other criteria were taken into account on Table3.1.

3.1 UWSim

UWSim is a software tool developed in the scope of the TRIDENT project, for visualization and simulation of underwater robotic missions. Its realistic images are rendered through OpenSceneGraph (OSG), Bullet and osgOcean. OSG is an open source 3D graphics application, while the plugin osgOcean adds visual improvements such as waves, water coloration, reflection/refraction, flotation of sediments, etc. The Bullet physics engine adds the physical fidelity.

Vehicle control can be simulated through ROS nodes. Previously recorded data can be also be reproduced in UWSim. UWSim requires some level of previous experience using ROS. The UWSim wiki page contains articles on installing the software and on the configuration and creation of simulation scenes. The major drawbacks in UWSim are that is only a kinematic simulator and that there is no convenient way of extending the software, any modifications, e. g. adding a new type of sensor must be written in the core source code.

The physics engine is used only to handle contact forces and the implementation of the vehicle dynamics, including the simulation of thruster forces. It is located in one monolithic ROS node, but it could be modified to adhere to a more modular structure.

UWSim possess also an interface to communicate with external software such as Matlab, using ROS nodes.

In this simulator it is also possible to visualize different underwater virtual scenarios that can be configured using standard 3D modeling software (ex: blender, 3D Studio Max, etc). Controllable underwater vehicles, as well as simulated sensors can be added to the scene and accessed externally through network interfaces. This allows to easily integrate the visualization tool with existing control architectures. The scenes in UWSim are XML-formatted documents that describes the general scenario, and simulation parameters. On the other and, robots are described with an Unified Robot Description Format (URDF) file. However, a scene eXtensible Markup Language (XML) file may make a reference to an URDF file for including a robot into the scene. The UWSim scene XML file is divided in blocks, which define the different aspects of the scene. The available blocks are the oceanState block that allows configuring ocean parameters, simParams block, that makes possible to modify the settings of the simulator, the camera block for set the main camera parameters, the vehicle tag that is used to create and configure underwater robots and the sensors available on them, the object block that allows inclusion of other 3D models to interact with the robots and the ROS interfaces block that allows the attachment of ROS interfaces to certain objects, robots or sensors, specifying the communication possibilities.

There are twelve sensors available for vehicles in the current version of UWSim such as camera, range sensor, pressure sensor, Doppler Velocity Log (DVL), Inertial Measurement Unit (IMU), GPS, Multibeam, force sensor and structured light projector.

The supported 3D models formats are all that are supported by OSG, like .osg, .obj, .ive, .stl, .3ds and others. So, it is possible to use a simple 3D modeling program such as Blender that can be used to export a 3D model with one of the formats above.

UWSim requires some level of previous experience using ROS. The UWSim wiki page contains articles on installing the software and on the configuration and creation of simulation scenes.

3.2 Gazebo

Gazebo features multiple physics engines and handles well the dynamics, contact physics and is very versatile through its plugin-based design, recent example of use is shown in [20]. Simulated data can be exported through topics to third party applications.

It uses the Ogre3D rendering engine and supports multiple physics engines, being its default physics engine the Open Dynamics Engine (ODE). The physical fidelity of this

simulator will be dependent on the physics engine chosen for the compilation.

A vast number of sensors are already available in Gazebo, there is also a provided Application Programming Interface (API), for the creation of new sensors as plugins for Gazebo. Gazebo include different types of plugins, each of them controls specific part of simulation. Some examples are world plugin that control a specific world properties such as physics engine, ambient lightning etc. Model plugin that control a specific joint, links and state of a model. And sensor plugin that control sensor information and properties. Each plugin type controls a specific parameters of the simulation. There may be multiple plugins of the same type, controlling distinct components of the simulation. The simulator also presents communication nodes, namely Gazebo topics, used to export simulated data to third party applications.

For an inexperienced user, Gazebo can be fairly easy understood and used, with the help of the large number of tutorials available in Gazebo's web-page and a big community, though it also requires some previous experience with ROS for a deeper understanding.

Gazebo is fully included in every ROS distribution with high interaction and compatibility. It is also possible to use different Gazebo versions with ROS distributions, from the ones originally included.

Gazebo's rendering system is not optimized for underwater scenario, where underwater characteristics are not taken into account. However, Gazebo simulator gives the user the possibility to extend the simulation with plugins. It can be extended for new dynamics, rendering, sensors and world models. Graphical user interface is included to visualize the scenario with extended capabilities.

Robot, sensor or world models are described in their respective Simulation Description Format (SDF) file, an XML format designed for Gazebo. The URDF file format used by ROS is automatically converted to SDF format when used by Gazebo. Visual geometries used by the rendering engine are provided in COLLADA format and the collision geometries in STL format. In Gazebo 8, OBJ format was added as alternative input option for COLLADA.

Gazebo has a regularly updated and well-defined road-map for new releases. The integration with ROS, also developed by Open Source Robotics Foundation (OSRF), is already guaranteed through Gazebo/ROS packages. Contributions from the Gazebo user community allow it to be regularly improved with new features. One of the drawbacks of Gazebo is the lack of tools to realistically represent underwater environments.

3.3 MORSE

MORSE uses the Blender Game Engine and the Bullet physics engine. Two configuration are provided by MORSE for time handling: best effort, that tries to keep a real-time simulation and fixed step that ensures accuracy of simulation.

Like UWSim and Gazebo, MORSE already has a vast number of sensors available for general use.

MORSE has many advantages such as the use of Blender, where a high customization and a high level of graphical detail can be achieved, being designed to allow simulations of multiple robots systems, the ability to create new sensors and camera views and it has multiple tutorials divided by levels of proficiency. However, it has a disadvantage from a research standpoint, as it relies also on Blender, those who are unfamiliar with Blender and its interface may find themselves with a learning curve that could be avoided by choosing a different simulator.

3.4 Underwater Robotics Simulations

Unmanned Underwater Vehicle (UUV) Simulator[21] is an extension of the Gazebo simulator. UUV Simulator adds numerous new implemented plugins and underwater scenarios to the very deficient Gazebo underwater library. It extends the Gazebo simulation platform to underwater environments. The simulation includes new and upgraded sensor models, such as GPS, IMU, magnetometer, pressure sensor, sonar, camera, stereo camera, and DVL. The simulation also include control modules, such as teleoperation and robotic arm control, thruster and fins dynamics, underwater scenarios and environmental loads, hydrodynamic and hydrostatic forces.

Another example is DexROV simulator[22] which uses Gazebo 7.0. The environment produced by the robot is represented by voxels. The environment and vehicle used in the simulation are modeled by real collected data. Include sensor data and network simulation.

ROVsim[23] is an underwater simulator made for Remotely Operated Vehicle (ROV) operations. The simulation includes a wide range of different ROV's and operations. It provides realistic visualization, accurate physics, realistic ocean environment, 5-7 Degrees of Freedom (DOF) manipulators, sonar and other sensors. The drawback of this simulator is that it is a commercial product and more oriented for pilot training, demonstrations, and scholar use.

DeMarco et al.[15] developed a simulator done for Underwater Human-Robot Inter-

action (UHRI) based on MORSE and ROS. This work includes a dynamic model of VideoRay Pro 4 ROV and diver, in addition to forward-looking sonar, and underwater visualization. This simulation also used ROS in combination with Mission Oriented Operating Suite (MOOS) for autonomy architecture that allows for tracking contacts, defining state machines, fusing autonomous behaviors, setting mission parameters, and controlling and monitoring the robot during mission deployment.

Chapter 4

Simulation environment

After a detailed analysis, the Gazebo simulator was selected as the simulation platform for UNEXMIN project.

The most important criteria of analysis were the compatibility with the Linux operating system and ROS middleware. The plugin model facilitates the implementation of new modules necessary for the project and bypasses the lack of underwater details in Gazebo simulator. Underwater sensor models are lacking in simulation, some of them are complex and difficult to simulate in real time. With the help of plugins, it is possible to make new sensor models that would work in real time, due to Gazebo balanced performance. The four different physics engines available in the simulator imply broader dynamics development in the simulator.

Gazebo continues its development with the support of a diverse and active community and new modules are being created for the underwater community as well. The possibility to improve the simulator, make detailed environments, sensor models, with the robot dynamics makes the Gazebo simulator the best candidate for UNEXMIN project.

4.1 ROS

ROS is open-source, BSD licensed, middleware framework used in robotics. It provides a consistent robotic software development environment, multiple inter process communication tools and robotic related software modules. One of the principles is the ability to make software portable and being able to use/adapt with any robot. This way there can be created and shared new functionalities in other robots which improves the evolution progress of robotics overall. Instead of programming everything from scratch, reuse, and/or expand existing code/packages. That was the major point of ROS: ease code

reuse between researchers/developers, so they waste less time reinventing the wheel. It's written mostly in C++/Python.

ROS is fully supported by Linux operation system, Ubuntu distribution. Meanwhile, there are ways to use it with Windows, OSX, and even Android, operating systems that are not currently fully supported.

ROS was developed in 2007 by the Stanford Artificial Intelligence Laboratory (SAIL) in support of the Stanford AI Robot project. As of 2008, development continues primarily at Willow Garage, a robotic company spin-off, with more than twenty institutions collaborating within a federated development model.

ROS have a big and growing community in robotics. Devices used in robotics are also being adapted to ROS. It provides the architecture for navigation, sensing, and robot parts manipulation.

ROS is widely-used and offers easy startup for researchers to quickly equip robots with basic software and focus on experimental work.

It is a middleware that provides:

- Publish/Subscribe anonymous message passing;
- Recording and playback of messages;
- Request/Response remote procedure calls. The ROS middleware provides this capability using services;
- Distributed parameter system. This system allows the easy modification of task settings, and also allows tasks to change the configuration of other tasks.

ROS provides common robot-specific libraries and tools and its workspace is described by the following architecture:

- *Packages:* Packages are the main unit for organizing software in ROS that can be built and shared. A package can contain ROS nodes, a ROS-dependent library, datasets, configuration files, or anything else that is part of the project;
- *Metapackages:* Metapackages essentially contain numerous Packages that are part of the same project. They are handier for packages organization;
- *Package Manifests:* Package Manifests (package.xml files) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages;

- *Message types:* Message descriptions, stored in package message specific folder, define the data structures for messages sent in ROS;

- *Service types:* Service descriptions, stored in package service specific folder, define the request and response data structures for services in ROS.

It is based on graph architecture with a centralized topology, where processing takes place in nodes that may receive, post the sensor, control, state, planning, actuator, and so on. The library is geared towards a Unix-like system. The concepts of the computational graph in ROS are:

- *Nodes:* Nodes are processes that perform a computation. A robot in a ROS system usually comprises many nodes that control/perform every aspect of the robot, including, motors, sensors, navigation, etc.

- *Master:* The ROS Master provides name registration, lookup, and exchange of information between processes to the rest of the Computation Graph elements. It stores topics and services registration information for ROS nodes.

- *Parameter Server:* The Parameter Server is part of the Master and allows data to be stored by key in a central location.

- *Messages:* Messages simply a data structure, comprising typed fields that are used as a way to communicate between Nodes. Messages can also include arbitrarily nested structures and arrays.

- *Topics:* Topics identify the content of the message. They are transport buses with publish/subscribe semantics. A node sends out a message by publishing it on a given topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. With this methodology nodes communicate anonymously, unaware of each other existence.

- *Services:* Services allow request/reply communication between nodes, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.

- *Bags:* Bags are a format for saving and playing back ROS message data (topics).

The most common protocol used in a ROS is called TCPROS, which uses standard TCP/IP sockets.

ROS tools make easy to comprehend, monitor and debug issues as they occur. These tools support introspecting, debugging, plotting, and visualizing the state of the system. All tools are accessible with the command line, with similar or extended functionalities, graphical interfaces are also included for better interpretation, like rviz and rqt.

Rqt is a Qt-based framework with already included rqt plugins like: rqt_grapth, rqt_plot, rqt_topic, rqt_publisher, rqt_reconfigure, rqt_console and rqt_bag. Rqt plugins can also be extended and created with new functionalities.

Rviz package provides three-dimensional visualization of sensor data and any described robot, including laser scans, point clouds, camera images, markers, pose and rendering of the robot. All information is related to a frame of choice.

Tf is a ROS package that keeps track of 3D coordinate frames over time. Tf operates in a distributed system and maintains the relationship between coordinates frames in a tree structure. It also allows performing several operations regarding these coordinates.

4.2 Gazebo

Gazebo simulator uses a distributed architecture with separated libraries for physics simulation, rendering, user interface, communication, and sensor generation. Simulation is separated by two executable programs:

- **gzserver:** Simulates physics, rendering, and sensors. It parses a world description file and then simulates the world using a physics and sensor engine;

- **gzclient:** Provides a GUI for visualization and interaction/modification of the simulation.

A communication library is responsible for communication between processes and it uses the open source Google Protobuf for the message serialization and boost::ASIO for the transport mechanism. Similar to ROS, it supports publish/subscribe communication paradigm.

Topic management and name lookup are performed by Gazebo Master. A single master can handle multiple physics simulations, sensor generators, and GUIs.

The physics library provides fundamental simulation components, including rigid bodies, collision shapes, and joints articulation constraints. Four open-source physics

engines are integrated: Open Dynamics Engine (ODE); Bullet; Simbody; Dynamic Animation and Robotics Toolkit (DART), with the possibility to load robot models with any of these engines.

The rendering library uses OGRE which provides a simple interface for rendering 3D scenes to both the GUI and sensor libraries.

Sensor generation library implements all types of sensors, with world state updates and produces output specified by sensors.

The GUI library uses Qt to create graphical widgets for simulation.

All libraries in Gazebo simulator support plugins. Those plugins have access to libraries and can add new features without using the communication system.

4.3 Gazebo-ROS

Integration between Gazebo and ROS is performed with the *gazebo_ros_pkgs* meta package. This includes:

- **gazebo_ros:** Wraps gzserver and gzclient by using two Gazebo plugins that provide the necessary ROS interface for messages, services and dynamic reconfigure;
- **gazebo_msgs:** Messages and Service data structures for interfacing with Gazebo from ROS;
- **gazebo_plugins:** Robot-independent Gazebo plugins, sensors and motory (joints, force, template).

Figure 4.1 represent Gazebo-ROS architecture. The *gazebo_ros_api_plugin* plugin, located within the *gazebo_ros* package, initializes a ROS node called "gazebo". It integrates a scheduler to provide the ROS interfaces, such as services and topics that enables the user to manipulate the properties of the simulation environment over ROS, as well as spawn and introspect on the state of models in an environment.

A second plugin, *gazebo_ros_paths_plugin*, also available in the *gazebo_ros* package, allows Gazebo to find ROS resources.

4.4 Plugins

A plugin in Gazebo is a C++ code that is compiled as a shared library and inserted directly into the simulation. The plugin has direct access to all the functionalities of Gazebo through the standard C++ classes.

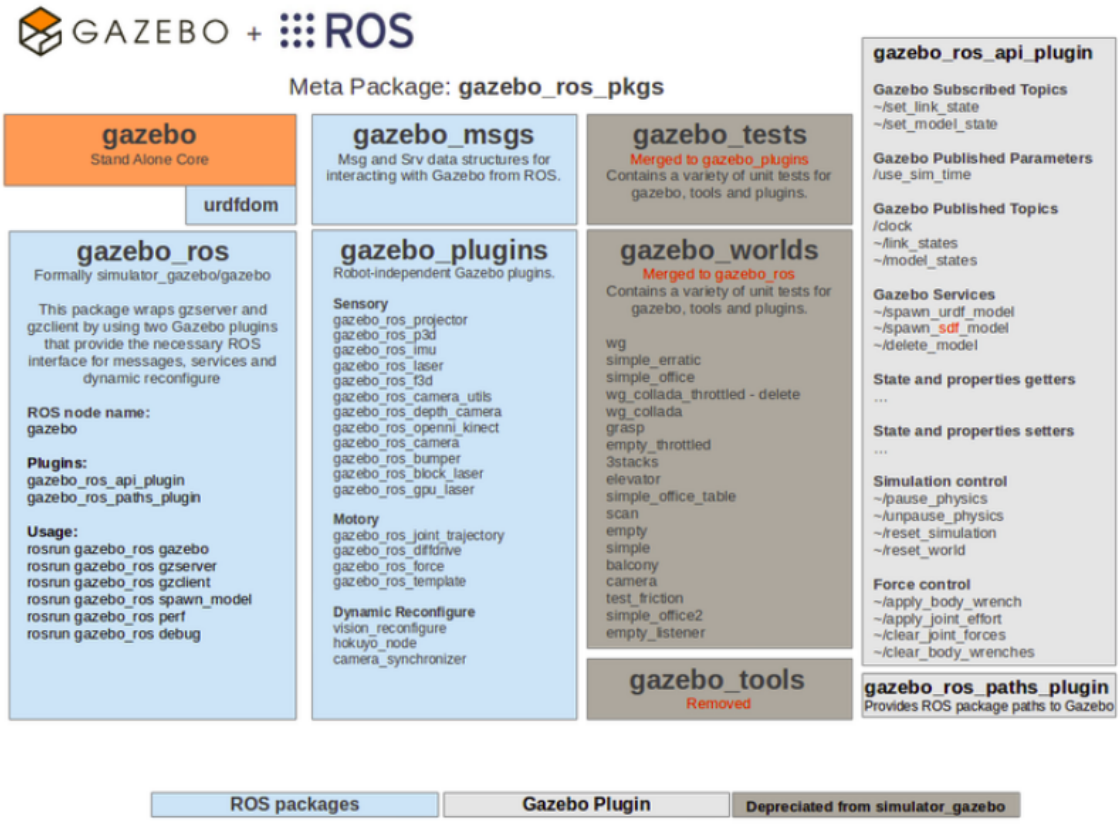


Figure 4.1: Gazebo-ROS architecture[1].

All plugins must be inside the gazebo namespace. Each plugin inherits from a plugin class type, such as a sensor, model, world, GUI, and others. A mandatory function is **Load** which receives SDF file with all robot descriptions.

In the end, the plugin is registered within the simulator using a macro.

On startup, Gazebo parses the SDF file, locates the plugin, and loads the code. Plugins are attached to an SDF file, it can be either model, world, sensor, etc. Each plugin is attached to different parts of environment description depending on the type of plugin.

Plugins can improve and add new functionalities to the simulation and also publish/-subscribe to ROS topics and services.

4.5 Environment description

4.5.1 World Files

World file contains all the description about elements in simulation, including models, lights, sensors, physics. The file format of a world file is same as the model file, using SDF format, with *.world* extension. The Gazebo server (gzserver) reads this file to generate and populate a world in the simulation.

4.5.2 Model Files

A model file uses the same SDF format as the world files. A model file can be included in the world file, or spawn directly into the simulation with spawn_robot node in ROS. With the second method is possible the use of Xacro files that allow management and implementation of parameters allowing the user to adjust every aspect of robot, sensor or simulation. Spawn_robot node translates Xacro file to SDF file for Gazebo simulator interpretation. The spawn_robot method uses a python script to make a service call request to the *gazebo_ros* ROS node to add a custom URDF into a simulated environment. The script is located within *gazebo_ros* package.

4.5.3 Environment Variables

There are many ways to start Gazebo, open world models and spawn robot models into the simulated environment. Gazebo uses different environment variables (paths) to locate files, including model, resource, master, plugin. These variables can be modified in a shell script, to extend the path it searches for models which facilitate the creation of new packages in ROS and makes them more cross-platform ready. URDF

files, meshes, materials, media, are stored in ROS packages with resource paths relative to ROS workspace.

4.5.4 Xacro

Xacro (XML Macros) is an XML format file that can construct shorter and more readable XML files by using macros. It's most useful when working with large XML files and add new functionalities, including property's, property blocks, math expressions, condition blocks, rospack commands, macros, default parameters and the inclusion of other xacro files.

4.5.5 URDF

The Universal Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot. Besides the inclusion and use of all distinctive parameters and macros, inside URDF files are also defined the property's like links, joints, visual, collision, mass, inertial, pose, sensors, plugins, geometry, and all the parameters that are required for each plugin.

The conjunction of URDF files describes all the information about the robot that will spawn inside the simulation world. That include its material properties, dynamics, sensors, plugins, navigation algorithms, everything that is related directly to the robot.

The Gazebo simulator uses SDF file for robot description, which is similar to URDF files but with extra information that is unique for the simulator. Inside URDF files is used the parameter $< gazebo >$ to describe every information about the robot that is unique to SDF file. Before the robot is spawned into the simulation world, URDF file is translated automatically into a SDF file by Gazebo and is launched into Gazebo world.

Chapter 5

UNEXMIN Simulator

This chapter will describe the details of the developed simulation platform, such as the environment, models, sensors, controllers, etc. The platform also includes a multi-robot option with individual sensors and messages for each robot, numerous environments, sensors configurations, hydrodynamics, thruster forces, and ocean visualization. Figure 5.1 represent the relation between the simulator and real robot (Hardware in loop simulation),

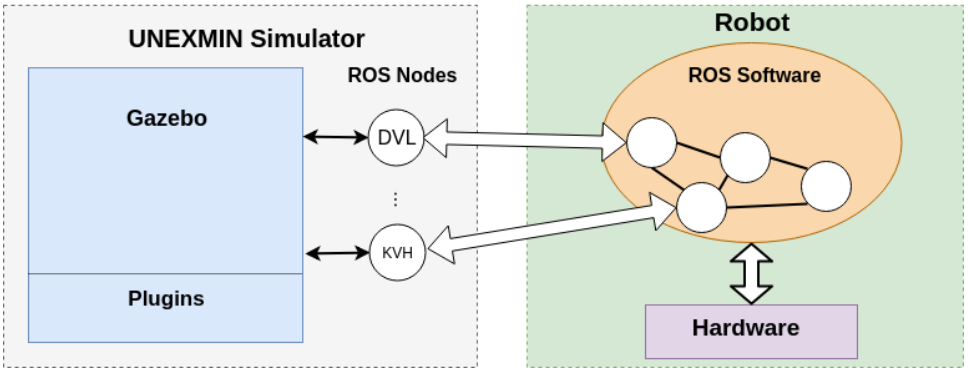


Figure 5.1: General architecture between robot and simulator.

The simulation platform aggregates various modules. Figure 5.2 represents the software architecture of the simulation models and interaction between Gazebo and ROS. Developed plugins are displayed in the middle of the figure as they works between ROS and Gazebo. On Gazebo side, plugins receive/send information to simulator in order to perform desired tasks. And in ROS part, information is send/received from plugins and published into desired topics. ROS nodes are also created in that part for messages interpretation. In order to perform hardware in loop simulation, the messages and formats

of the simulated modules are strictly the same as in real robot. Only the information inside the messages is simulated. With the close interaction between Gazebo and ROS it is possible to apply the simulation into the real robot information system and vice verse, as will be explained in this chapter.

5.1 Simulation platform structure

Simulations, vehicles, sensors, and control have been done multiple times before. ROS and Gazebo have already basic packages for simulation of different aspects of robot and environment. This implementation consists of the making of new UNEXMIN simulator that will accomplish the project requirements. The simulation package consists of the creation of new plugins and models for underwater environments and sensors. Although, some parts of the simulation package include other software, such as UUV Simulator package and other ROS packages that have already developed useful plugins and modules. Those packages were modified and adapted for this work. The implementation consists of a ROS metapackage with three main packages, *unexmin_description*, *unexmin_keyboard_op* and *unexmin_gazebo_plugins*. This metapackage requires Gazebo version 8 or later and ROS installed. It is recommended the usage of ROS Kinetic or later. The package was tested in ROS Kinetic with Gazebo 8.6, which was chosen in order to take advantage of all its latest features. The folder structure follows the next representation:

```
unexmin_gazebo_plugins
├── include
├── msg
└── src
unexmin_keyboard_op
└── src
unexmin_descriptions
├── launch
├── media
├── meshes
├── urdf
│   ├── UX_0
│   ├── UX_1
│   └── UX_2
└── worlds
```

unexmin_gazebo_plugins package includes all plugins, ROS nodes and messages for simulation. *unexmin_descriptions* package includes a description of all simulation as-

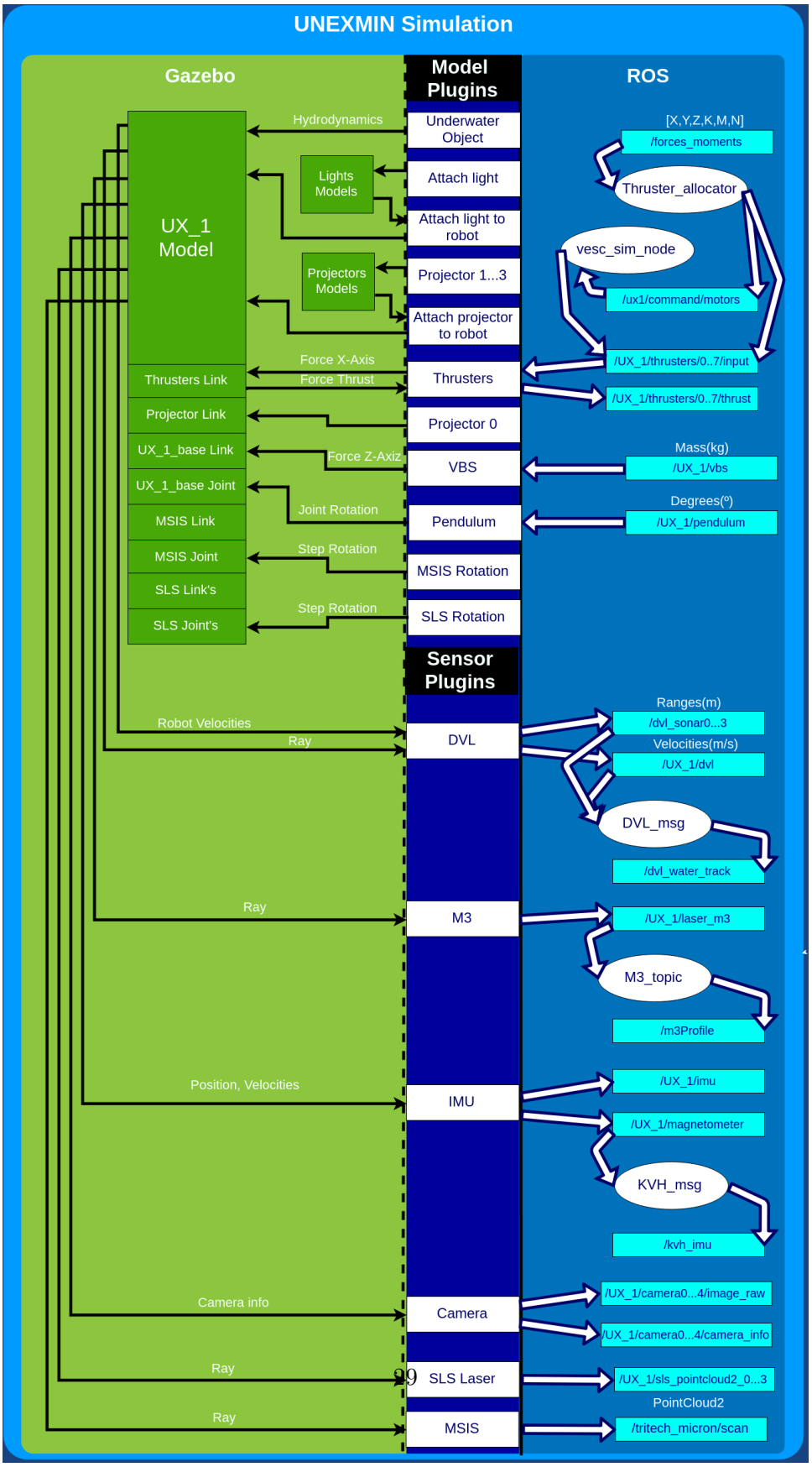


Figure 5.2: Software Architecture.

pects, such as launch file, textures, images, meshes, URDF files for each robot and worlds. Inside the URDF folder, the URDF files are separated for each robot description aspect and are combined with a macro file. *unexmin_keyboard_op* package include a python robot control tool. Which enables the keyboard control over thrusters, pendulum and ballast system. This control tool is only necessary for test purposes and direct command on simulator.

The robot description is made in a URDF file. URDF files are commonly used by ROS, these are converted to SDF files when used by Gazebo. Xacros are also used in this package, which can combine files in a specific order and run URDF files with parameters.

5.2 Simulation World

The visualization of world elements such as mines, ocean, robot, and sensors are possible with the use of real mesh files. Mesh files are created and exported into the simulation with the proper visual elements, dynamic properties, collisions, and size.

Gazebo only accepts meshes described in Collada(.dae) type files and .obj so it was necessary the resort to 3 third-party software for processing and conversions. The first one, MeshLab[24], was used to import the Point Cloud file of the underground mine, reduce its quality using Poisson Sampling, reconstruct its surfaces, using Poisson Reconstruction and finally, to export the final mesh as a Collada file. The second one, FreeCad[25], was used to import the Step files from a Sphere Robot, where the sensors will be placed, and of the sensor obtained from the company's website, and exporting them as Collada files. The third one, Blender[26], was used mostly to add textures to the meshes without changing its Collada format, this was done simply to improve the visual realism of the simulation.

Finally, to import the meshes into Gazebo it was necessary to create a World file where the path to the Collada file was indicated.

5.3 UX Robot

The robot consists of a sphere with the diameter of 0.6m, with multiple attached sensors, being all of them implemented in Gazebo simulation. The weight of the sphere (robot) was calculated so that it would achieve neutral buoyancy, a condition that imposes that the density of the robot is the same as the water, 1000 kg/m3, obtaining the approximate value of 113 kg. Sphere model includes the visual and collision parameters from the mesh

Figure 5.3: UX-1 Robot in simulation world.

and, for a proper behavior of the sphere, it was also necessary to compute its moment of inertia matrix, based on its mass and radius.

All robots implemented in the simulation have the same dimensions and structure. Although, the sensors can be configured with different parameters and position. The sensor configuration can be different from robot to robot for distinct mission plans.

5.4 Sensors

5.4.1 MSIS - Micron DST

The mechanical scanning imaging sonar includes the dimensions of the sensor equal to the real one since its original mesh was loaded into the simulation. This also applies to its mass and position in the sphere robot. Collisions and inertial parameters are calculated for each part of the sensor. The sensor uses a Ray type Gazebo sensor to create a 3D point cloud.

The physical part of the sensor is separated by two links. The bottom link is attached directly to the sphere robot, with a fixed type joint, in a correct robot position. The top link is attached to the bottom link, through a continuous rotational type joint, without limits.

Heading Rotation

This Model plugin was created to simulate the mechanical rotation of the sensor. It rotates the joint located between the top and bottom parts of the MSIS sensor. Simulates a mechanical rotation with configurable parameters like mechanical resolution, velocity and right/left limits of the scan sector up to a continuous 360^{o} scan. The plugin is controlled by Gazebo world updates, ensuring constant and proper rotation of the joint.

2D Point Cloud Sonar Image

As referred above the MSIS sensor is simulated thought a Gazebo plugin that uses the sensor Ray's to form a 3D pointcloud. This 3D pointcloud is then transformed into a 2D pointcloud sonar image, similar to the real sensor output. The sensor plugin is attached to the top link.

The transformation process is based on an area density method, where a specific area is selected and, based on the 3D pointcloud density, the intensity of echo and gain on the new 2D pointcloud are calculated, as represented in figures 5.4 and 5.5.

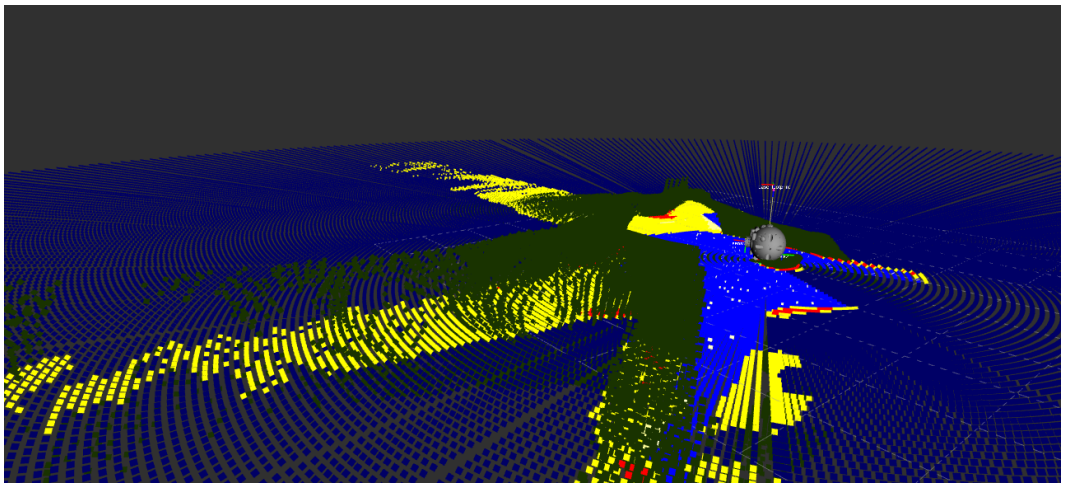


Figure 5.4: 3D to 2D pointcloud representation.

Rays are defined in horizontal and vertical planes, and each point is formed from intersection between horizontal and vertical rays. During the calculations of the points, the ray sensor is deactivated and is activated again after the pointcloud creation, this process occur in every update from Gazebo world, assuring that the calculations are made with current update. The Plugin obtains several parameters from Gazebo, such as range, horizontal angle and vertical angle from each simulated ray. The raw point

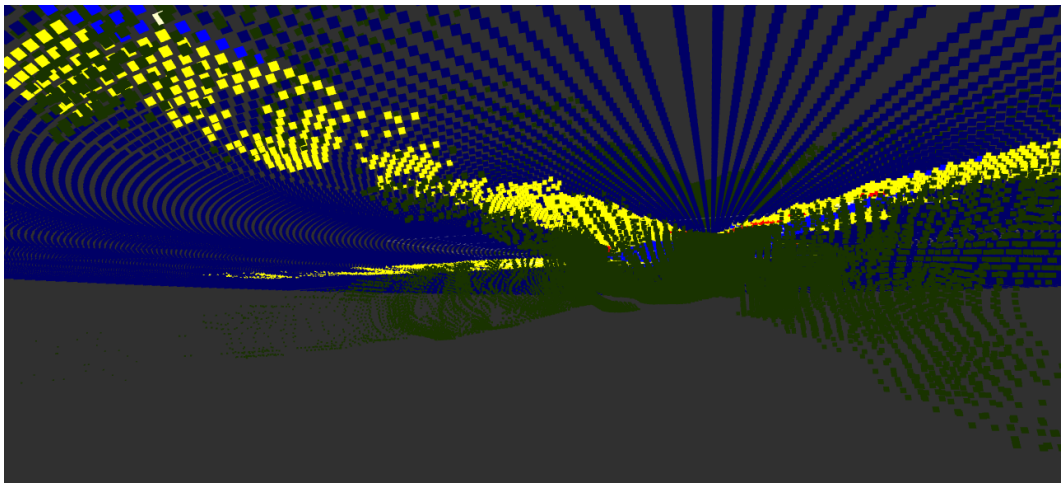


Figure 5.5: 3D to 2D pointcloud representation.

cloud is created with that parameters, where the points (X,Y,Z) are computed with the equations below:

$$X = range * cos(verticalAngle) * cos(horizontalAngle) \tag{5.1}$$

$$Y = range * cos(verticalAngle) * sin(horizontalAngle) \tag{5.2}$$

$$Z = range * sin(verticalAngle) \tag{5.3}$$

The Raw 3D pointcloud only includes points where a collision was detected (echo). This is used to create a new 2D pointcloud that is filled with points from the minimum range to the maximum range. The number of points created depends on two parameters:

- ***horizontal_point_resolution(HPR)*:** is a representation by how many points are created between the minimum and maximum ranges of the rays. For example, if *min = 0.3 m* and *max = 75 m*, with *resolution = 250*, there will be created 250 points in horizontal with 0.3 m of spacing between them. More resolution will decrease the spacing.

- ***vertical_point_resolution(VPR)*:** is a representation by how many points are created between the minimum horizontal angle and the maximum horizontal angle of the sensor. For example, if *min = -1.5º* and *max = 1.5º*, with *resolution = 10*, there will be created 10 points in vertical with 0.3^{o} of spacing between them. More

resolution will decrease the spacing.

The final point cloud created has *HPR* VPR* points. Inlcuding X, Y, Z, intesity of echoes and RGB (which represent different color for each intensity value). X, Y and Z are represented with following equations:

$$X = HPR\_range * cos(VPR\_Angle) \tag{5.4}$$

$$Y = HPR\_range * sin(VPR\_Angle) \tag{5.5}$$

$$Z = 0 \tag{5.6}$$

Noise Models

This package includes four types of noise models:

- **Gaussian Noise:** The Gaussian noise is applied directly into the Ray sensor before the 3D point cloud generation. The simulation of this noise is included in the Gazebo sensor parameters. The standard deviation is based on the Micron DST specifications.
- **Salt And Pepper:** Even with Gaussian noise included, simulation provides results far too perfect, to simulate a more realistic output, salt and pepper noise was included. This noise can be simply described by the introduction of random echoes.
- **Acoustic Shadow:** Another common effect in sonar sensors is the acoustic shadow. This noise model improves even more veracity to our sensor output.
- **Artifacts:** With few active rays, the simulation produce artifacts in the ground plane. This noise model includes echoes between these artifacts maintaining a constant echo in the output.

Output and Parameters

This package publishes two ROS topics:

- */tritech_micron/scan*: include the *sensor_msgs/PointCloud2* message with custom structure (x, y, z, intensity, RGB);

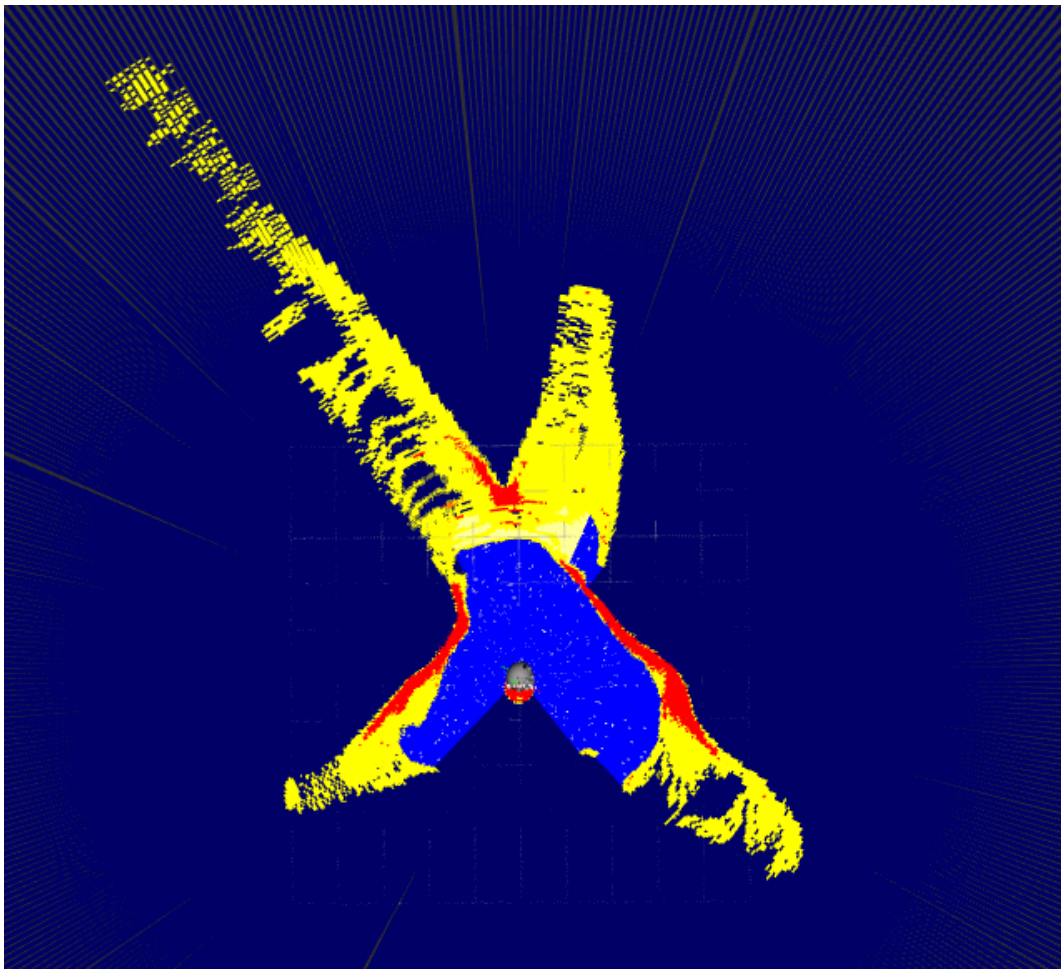


Figure 5.6: Final 2D pointcloud representation.

- */msis/msis_r*: include the heading angle of the rotating Micron DST sensor in radians;

The metapackage includes highly customized parameters, like update rate, beamwidth, range, noise, pointcloud resolution, gain, step angle size, scanned sector (up to 360^{o} continuous scan), etc. With these parameters is possible to simulate any MSIS type sensor. These parameters are described in a launch file. Figure 5.6 represents the */tritech_micron/scan* 360^{o} scan output in Rviz. Colors represent:

- **Red:** High intensity echoes;
- **Yellow:** Medium intensity echoes;
- **White yellow:** Low intesity echoes;

- **Blue:** No echoes;
- **Dark blue:** Shadow artifacts.

5.4.2 DVL

DVL sensor has already been developed in UUV Simulation ROS package, and its base is used in this simulation platform. The dimensions, parameters, and specifications are modified to assimilate Nortek DVL 1000, as it is used in the real robot.

DVL plugin publishes next ROS topics:

- **/dvl**: Velocity (m/s), velocity covariance;
- **/dvl_sonar0-3**: Field of view, minimum range, maximum range and range measurement (m);
- **/dvl_twist**: Linear (x,y,z), angular (x,y,z) and covariance.

In order to publish similar message as used in the real robot (Nortek DVL 1000), a ROS node was created that transforms DVL topics described before into a single DVL ROS topic similar to the one published by the real robot. In order to do so, *dvl_nortek* package from real robot was used for it contains the message description. Noise models have obtained from Nortek DVL 100 datasheet and are introduced in this package.

New ROS topic message type is **dvl_nortek/DvlBottomWaterTrack** and published topic **/dvl_water_track**, with next information inside:

- Sound speed;
- Temperature;
- Pressure;
- Range (m) for each beam (4);
- Velocity beam (4), in beam coordinates;
- fom beam;
- dt1 beam;
- dt2 beam;
- time beam;

- Velocity xyzz (4), in instrument coordinates;
- fom xyzz;
- dt1 xyzz;
- dt2 xyzz;
- time xyzz;

Messages are published with 8 Hz rate which is same as real message from Nortek DVL drivers.

5.4.3 IMU

IMU sensor is already included in Gazebo library and has also been developed/improved by the community for better performance. This simulation platform uses an already developed IMU sensor and modifies the specifications to follow KVH 1775 sensor, which is used in the real robot.

The simulated IMU KVH 1775 sensor encompass a gyroscope, accelerometer, and a magnetometer. In order to simulate an exact copy of KVH 1775, a new ROS node was created to add all integrated sensors and publish the same messages as the real one. The published ROS topic follows the original driver with next parameters:

- Accelerometer [x,y,z] (m/s2);
- Gyroscope [x,y,z] (rad/s);
- Magnetometer [x,y,z] (mG);
- temperature_imu (º);
- pressure (bar);
- temperature_pressure (º).

The type of published message is *kvh_1775/ImuRawKvh*, and it is published at 500 Hz rate which is equal to the real sensor. Noise models are obtained from KVH 1775 datasheet and are introduced in this package.

5.4.4 M3

The Gazebo simulator only possess one sonar sensor model, and the characterization of that model is insufficient. In this case, the better option to simulate M3 multibeam sonar was the simulation with Ray plugin, already included in Gazebo library, which returns a pointcloud ROS topic with type *sensor_msgs/LaserScan*. That way, the simulated sonar only returns first echoes from each scan. As real sensor implemented in the vehicle return ROS topic type *PointCloudXYZ*, a ROS node was created that transforms the *LaserScan* type of message into a new ROS topic that is similar to the one used in the vehicle, *PointCloudXYZ*.

Simulated M3 sensor follows the same specifications as the real sensor (M3 Kingsberg) in profiling mode:

- *Update Rate:* 40 Hz;
- *Range:* 0.2m to 150m;
- *Range resolution:* 1cm;
- *Horizontal Field of View:* 120^{o};
- *Vertical Field of View:* 3^{o};
- *Number of Beams:* 256.

M3 sonar include visuals, collisions and position equal to the one in real robot configuration.

5.4.5 SLS

The SLS sensor is composed of laser, camera, and light to illuminate dark areas underwater. This sensor is developed in INESC-TEC laboratory and each part of the sensor is simulated. The laser scan line is between 90^{o} and 110^{o} and a camera has the resolution of 2054x1544.

Projector

The simulation of the laser part that is visible on the camera of the sensor is obtained via the **Projector** plugin, which is included in the Gazebo library. This plugin requires only an image as an input, and it projects the image in the simulation world. The projected image is a simple green line that is visually similar to the real laser used on

the vehicle. A new ROS node was created in order to simulate various SLS sensors, multiple projectors were thus installed in different positions on the robot.

Ray

The simulation part of the laser that returns a pointcloud of the scanned world is simulated via Ray plugin. Each SLS sensor has one Ray plugin attached in order to obtain pointclouds from each sensor. Ray plugin publishes pointcloud in *sensor_msgs::LaserScan* type of ROS message. Another plugin was created in order to transform *sensor_msgs::LaserScan* to *sensor_msgs::PointCloud2*. This was necessary for the implementation of SLS system lasers with Octomaps, as octompas require *sensor_msgs::PointCloud2* type of message in input.

The published pointcloud only returns laser line that is seen on the camera. From each camera, only the observable part of the laser (projector) is transformed into a final pointcloud. This is performed by projecting the camera field of view in world frame and then only publishing 3D laser point that are inside the field of view.

Camera

The simulation of the camera already exists in Gazebo library and its used in this part of the sensor. Each SLS sensor is related to one camera and it is positioned in the place designed by vehicle. Each camera has the same resolution and distortion parameters as the real robot, which makes the output image similar to the real one.

The camera can publish images with configurable fps. This parameter is actualized with real robot output fps.

Light

The lights of each SLS sensor are simulated with **Attach Light** Gazebo plugin. This is a world plugin and its located in the same position and orientation as every SLS sensor LED component in real robot. The light brightness, dispersion and other parameters are also adjustable to reassemble the real robot lights.

Motor

The rotation of the laser motor is simulated with a joint rotation plugin. The SLS system is composed of two links, that are connected with a single rotational joint. This plugin accepts three parameters that are velocity, left and right limits. With this parameters,

it's possible to adjust the rotation of the SLS system and test different sensor acquisition situations.

5.5 World

The world model in the Gazebo simulation platform is a collection of robots, objects, and global parameters. Those include the sky, light, shadow, environmental physics, ocean, mines, lakes, etc.

Every scenario has a different world file. Main underwater mine scenario include mine mesh, ocean waves, underwater visibility, attach lights models, projector models, and underwater current, with the parameters:

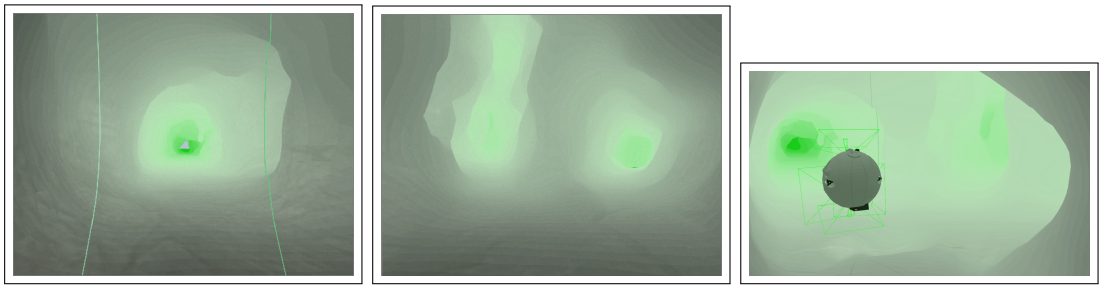
- Topic name;
- Velocity: mean, minimum, maximum, and noise amplitude;
- Horizontal angle: mean, minimum, maximum;
- Vertical angle: mean, minimum, maximum, and noise amplitude;

This plugin publishes a ROS topic. Then the vehicle that uses the underwater current subscribes to the topic.

5.5.1 Underwater visibility

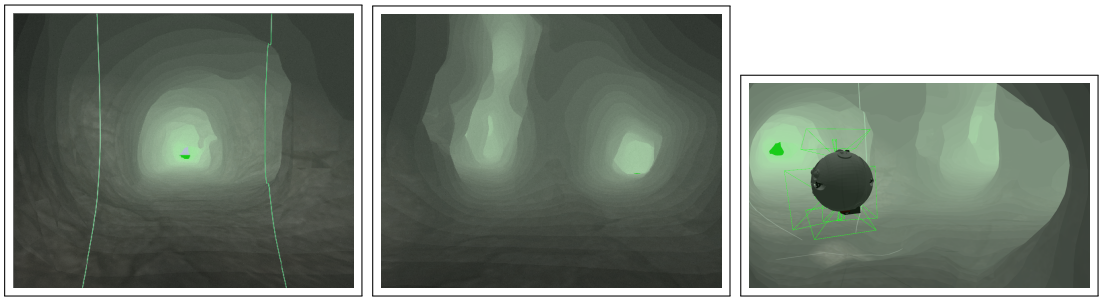
Underwater properties in the Gazebo simulator are absent. In order to add more visual fidelity and realistic underwater environment for simulated underwater cameras, the muddy water effect is added to the simulation package.

This effect uses the fog parameter within Gazebo scenes description, which can be configured for particual colour and density. As demonstrated in figures 5.7, , 5.8, 5.9 and 5.10. This effect reduces the visibility and details of the simulated world. With muddy water effect, it is harder to detect SLS laser lines within the image.



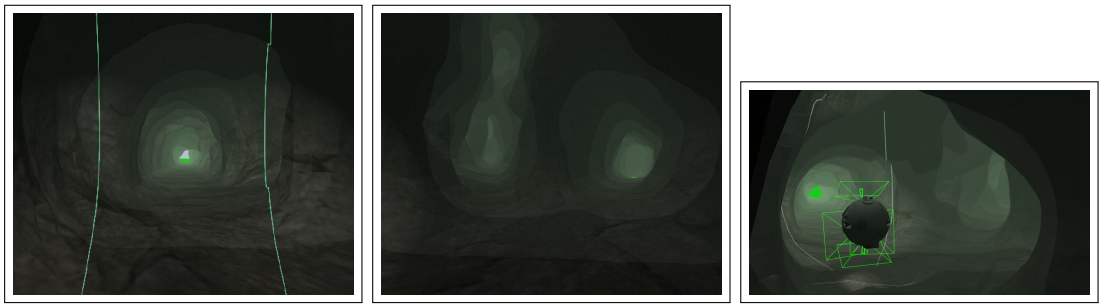
(a) Front Camera (b) Right Camera (c) GUI

Figure 5.7: Simulation image with high muddy water effect of (a) Front Camera, (b) Right Camera and (c) GUI.



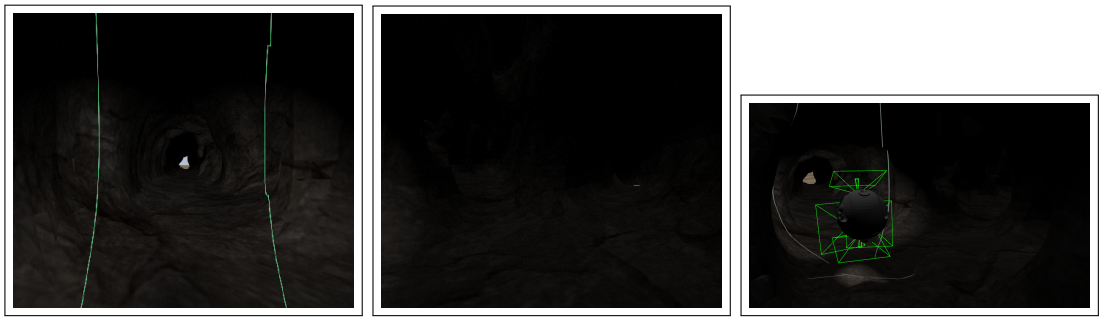
(a) Front Camera (b) Right Camera (c) GUI

Figure 5.8: Simulation image with medium muddy water effect of (a) Front Camera, (b) Right Camera and (c) GUI.



(a) Front Camera (b) Right Camera (c) GUI

Figure 5.9: Simulation image with low muddy water effect of (a) Front Camera, (b) Right Camera and (c) GUI.



(a) Front Camera (b) Right Camera (c) GUI

Figure 5.10: Simulation image with no muddy water effect of (a) Front Camera, (b) Right Camera and (c) GUI.

5.6 Hydrodynamics plugins

This simulation platform hydrodynamics are based on [27]. The hydrodynamic forces are expressed by:

$$M_A\dot{v}_r + C_A(v_r)v_r + D(v_r)v_r + g(\eta) = \tau \tag{5.7}$$

v is linear and angular velocities of the vehicle calculated by gazebo simulator. Where, added mass is a matrix of 6x6 is represented by:

$$M_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{\dot{r}} \end{bmatrix} \tag{5.8}$$

Coriolis matrix is represented by an 6x6 maxtrix:

$$C_A(v) = \begin{bmatrix} 0 & 0 & 0 & 0 & -a_3 & a_2 \\ 0 & 0 & 0 & a_3 & 0 & -a_1 \\ 0 & 0 & 0 & -a_2 & a_1 & 0 \\ 0 & -a_3 & a_2 & 0 & -a_6 & a_5 \\ a_3 & 0 & -a_1 & a_6 & 0 & -a_4 \\ -a_2 & a_1 & 0 & -a_5 & a_4 & 0 \end{bmatrix} \tag{5.9}$$

Where:

$$\begin{aligned} a_1 &= X_{\dot{u}}u + X_{\dot{v}}v + X_{\dot{w}}w + X_{\dot{p}}p + X_{\dot{q}}q + X_{\dot{r}}r \\ a_2 &= Y_{\dot{u}}u + Y_{\dot{v}}v + Y_{\dot{w}}w + Y_{\dot{p}}p + Y_{\dot{q}}q + Y_{\dot{r}}r \\ a_3 &= Z_{\dot{u}}u + Z_{\dot{v}}v + Z_{\dot{w}}w + Z_{\dot{p}}p + Z_{\dot{q}}q + Z_{\dot{r}}r \\ a_4 &= X_{\dot{p}}u + Y_{\dot{p}}v + Z_{\dot{p}}w + K_{\dot{p}}p + K_{\dot{q}}q + K_{\dot{r}}r \\ a_5 &= X_{\dot{q}}u + Y_{\dot{q}}v + Z_{\dot{q}}w + K_{\dot{q}}p + M_{\dot{q}}q + M_{\dot{r}}r \\ a_6 &= X_{\dot{r}}u + Y_{\dot{r}}v + Z_{\dot{r}}w + K_{\dot{r}}p + M_{\dot{r}}q + N_{\dot{r}}r \end{aligned} \tag{5.10}$$

$D(v_r)$ is damping matrix consisted of linear(vector of 6) and quadratic(vector of 6) damping. $g(\eta)$ is buoyancy force. Then, τ is converted into forces and moments applied to Gazebo's reference frame. Those forces and moments are simulated thought external propulsion originating from thrusters.

$$\tau = \begin{bmatrix} X & Y & Z & K & M & N \end{bmatrix}^T \tag{5.11}$$

Buoyancy force are already developed in Gazebo and other ROS packages and it is used in this simulation.

A model plugin applies the hydrodynamics and hydrostatic forces described above in the vehicle. The plugin require a list of vehicle and environment parameters, such as:

- Fluid density;
- Volume;
- Type of object (sphere, square, ...);
- Center of buoyancy;
- Hydrodynamic model;

5.7 Thrusters

Each thruster included in the vehicle has its own link and joint. It includes rotating visualization that comes with same propellers mesh file as in real robot.

Thruster dynamics are described in thruster Gazebo model plugin, with next parameters:

- Thruster ID;
- Publish thrust topic - *"/RobotName/thrusters/ID/thrust"*;
- Subscribe to input topic - *"/RobotName/thrusters/ID/input"*;
- Dynamics;
- Conversion.

Plugin is subscribed to */RobotName/thrusters/ID/input* topic for force input, which is converted to thruster thrust in x-forces, as represented in figure 5.11.

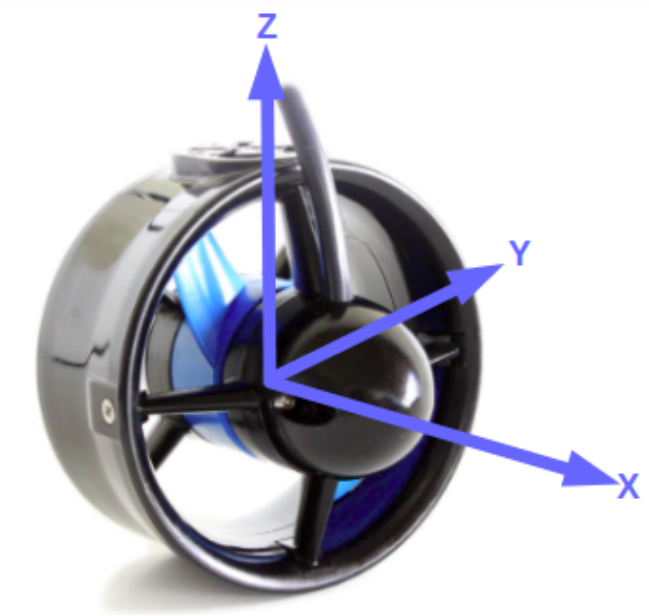


Figure 5.11: Thrusters axis.

5.8 Pendulum

Based on the configuration of the external thrusters and center of the mass, the UX-1 vehicle cannot pitch. To achieve it, an internal pendulum is used. Its movement displaces the internal mass of the robot, which then changes the center of mass and achieves new equilibrium position.

The pendulum model plugin simulates the pitch rotation of UX_1 robot and functionality of pendulum system implemented in the robot. The only plugin parameter input is the name of the topic to subscribe(*/UX_1/pendulum*).

The movement of a pendulum is simulated with a joint between robot model link and interior pendulum link. After receiving angle(α) command from ROS topic, it rotates with constant velocity to the desired position and lock's the pendulum in that position. The velocity is another parameter and can be set to different values.

5.9 Variable Ballast System

In order to allow for accurate trimming of the vehicle weight and compensation of hull compression due to depth, Variable Ballast System (VBS) is implemented in the robot. The system makes possible the very precise adjusts to the weight of the robot for its stability and maneuver control.

The VBS model plugin simulates the management of mass in the robot model. The only plugin parameter input is the name of the topic to subscribe (*/UX_1/vbs*).

The data received from the subscribed topic is expressed in mass, then its converted to force in Newton and is applied as a function of force in Z-axis of robot in world model. This changes the vehicle density inside a simulated underwater world.

5.10 Control

The control of the simulated vehicle follows the same philosophy as the real robot. Its movement is simulated through the rotation of the thrusters and thrust force. The pitch movement is performed by a pendulum system and VBS helps to maintain stability and control of vehicle weight. All those control properties were implemented in this simulation platform and will be described in this section.

Two types of control were implemented. First, consist of control performed by keyboard applying only simulation packages. The second one makes use of control packages from a real robot and publishes/subscribe to topics of the real robot, allowing hardware

in loop simulation control.

5.10.1 UNEXMIN Keyboard Operative

UNEXMIN Keyboard Operative is a simulation control package for UNEXMIN simulated vehicle performed only by keyboard inputs.

This package includes a python file that publishes data to simulation movement topics like thrusters, pendulum and VBS system.

Keyboard input keys and correspondent functionality are represente in table 5.1.

Table 5.1: Keyboard Operative inputs.

Controls	**Functionality**
W	Up
S	Down
A	-Yaw
D	+Yaw
I	Forward
K	Backward
J	Left
L	Right
Z/X	Increase/Decrease linear speed
C/V	Increase/Decrease angular speed
E/R	Increase/Decrease pendulum angle by 1 degree
Y/U	Increase/Decrease VBS mass by 0.01 kg

The output of the controller is published in three different topics:

- Thruster control is published to */force_moments* topic;
- Pendulum is published to */UX_1/pendulum* topic;
- Ballast system is published to */UX_1/vbs* topic.

5.10.2 Simulation Thruster Allocator

Thruster control is performed by *Thruster_allocator_sim* ROS node. Which subscribes to */forces_moments*. */forces_moments* contains an array with forces (X,Y,Z) and moments (K,M,N), these are the external moments and forces provided by the thrusters in the body reference frame. The thruster allocation matrix (5.12) provides the mapping between each thruster and the generating forces acting on the vehicle. Control node publish data (force) information into each thruster input topic. Hardware in loop

simulation is achieved with */ux1/command/motors* topic, which is used by real robot Electronic Speed Controller (ESC) driver.

Published thrusters data is defined by thrusters matrix, which is the same as implemented in real robot model:

$$\begin{bmatrix} Motors \\ X \\ Y \\ Z \\ K \\ M \\ N \end{bmatrix} = \begin{bmatrix} M0 & M1 & M2 & M3 & M4 & M5 & M6 & M7 \\ 0.5 & 0.0 & -0.50 & 0.0 & 0.50 & 0.0 & -0.50 & 0.0 \\ -0.25 & -0.25 & -0.25 & 0.25 & 0.25 & -0.25 & 0.25 & -0.25 \\ 0.0 & -0.50 & 0.0 & -0.5 & 0.0 & 0.50 & 0.0 & -0.50 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.25 & 0.0 & 0.25 & 0.0 & 0.25 & 0.0 & -0.25 & 0.0 \end{bmatrix} \quad (5.12)$$

Each thruster receives input based on the matrix described above. The simulation platform publishes two types of messages, */forces_moments* that is received in simulation to control the simulated vehicle and */ux1/command/motors*, which is received by the real robot for thruster control. This makes possible the control of simulated and real robot at the same moment, in real time.

5.10.3 Simulation ESC Driver

Another type of control is performed by following the real robot commands. This control is performed by another node, the simulated ESC driver.

ESC simulation driver subscribes to */ux1/command/motors* topic and convert rpm/-duty cycle values into force data that is published to each simulation thruster.

This topic is responsible for missions reproduction and test in simulation. After doing a real test with the robot in the desired environment and saving a rosbag, it is possible to reproduce the experience in a simulation which will perform similar trajectory, with same thruster forces as in real experience.

5.11 Cross-workspace compatibility

The simulation platform packages work with other ROS workspaces without the complex installation process. The only requirement is ROS kinetic or higher, Gazebo 8 and dependent ROS packages. No installation is required. The simulation package was tested with other platforms and within virtual machine environments (virtual box). Although being possible to run the simulation on a virtual machine, the performance is greatly

reduced and some actions are almost impossible to perform.

5.12 Vehicle Description and Initiation

As described in this chapter, the robot is detailed in multiple URDF files. The initialization of all parameters, simulation, and spawn of the robot is defined in the launch file.

5.12.1 Launch file

The launch file includes all parameters initiation, for the robot, sensor and starts the simulation with robot included in the desired environment. This is one of the standard methods in ROS to start and configure multiple processes running in the system.

First, it starts the Gazebo simulator and loads the world file. Next, it spawns the robot(s) with all the parameters included for URDF files. The launch file also includes initialization of all ROS nodes:

- **DVL_msg:** Node providing DVL message that is identical to real robot message (translation from the simulated output to a robot compatible format for hardware in loop simulation);
- **KVH_msg:** Node providing KVH message that is identical to real robot message;
- **thruster_allocator:** Matrix transformation for thrusters input;
- **rviz_mesh_ros_node:** Mesh visualization in Rviz;
- **sls_pointcloud2:** ROS laser message transformation to pointcloud2 type;
- **M3_topic:** Node responsible for creation of multibeam message that is identical to real robot message;
- **Robot state publisher:** Publish the state of a robot to tf. The node takes the joint angles of the robot as input and publishes the 3D poses of the robot links, using a kinematic tree model of the robot.
- **Message to tf:** Translates robot pose information to tf.

Tf is a ROS package that lets the user keep track of multiple coordinate frames over time. Tf maintains the relationship between coordinate frames in a tree structure

buffered in time and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time.

A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. Tf keeps track of all these frames over time. Tf can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. There is no central server to transform information. In figure 5.12 one can observe the ROS computational graph (*rqt_graph*) detailing the running nodes and publish/subscribe topics in the robot simulation.

5.13 Multirobots

The Gazebo simulator permits multiple robot simulations. In this work, there were used up to three UX robots (simulating the UNEXMIN multi-robot configuration scenario). Those robots are spawn in the launch file.

Each robot has individual sensors, thrusters, models, dynamics, and controllers. Every Gazebo plugin and ROS node is unique to each robot with the use of parameters. This implies that there are individual ROS topics and information for each sensor that is simulated in Gazebo.

The configuration of each robot can be customized with different sensors, thrusters, and overall composition. However, with every additional spawn robot, the performance of the simulation is reduced.

Figure 5.12: ROS partial rqt grath.

Chapter 6

Results

The simulation platform was tested with a real robot in different environments and scenarios. A comparison between simulated robot scenarios and real ones were performed in order to achieve the fidelity of the simulator. This allows to determine the validity of testing developments in simulation and also to predict expected variations when translating results to real world scenarios.

Several tests were performed with the simulation platform, those include real mines implementation and laboratory tank tests.

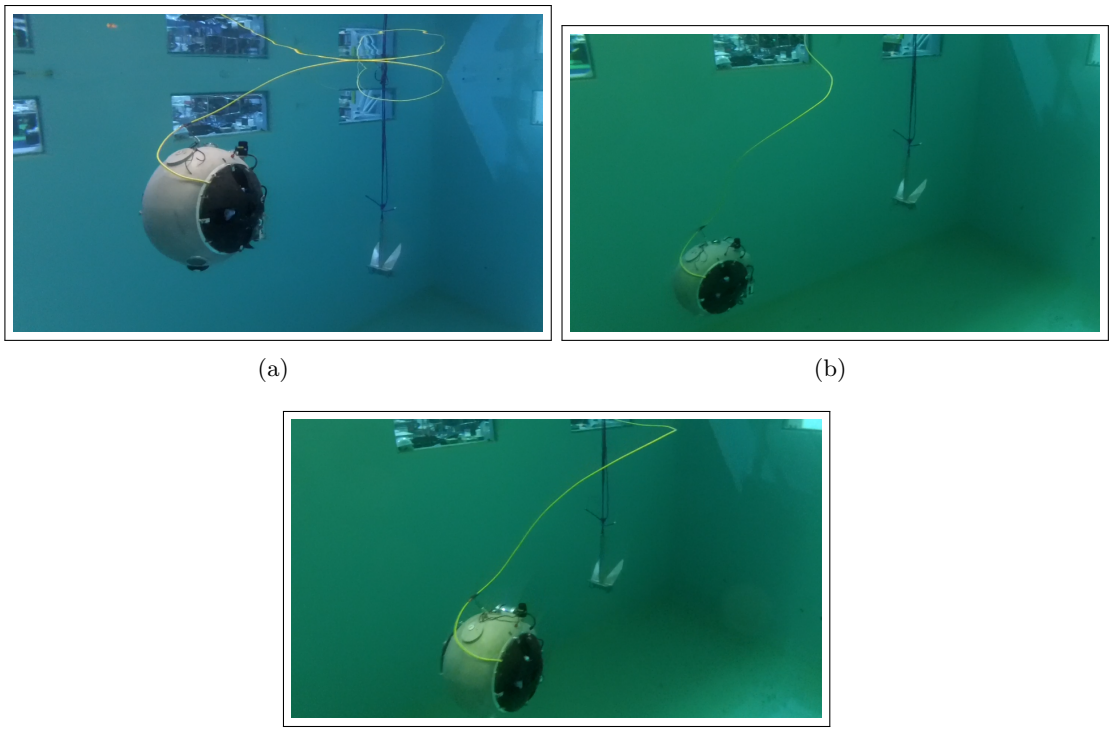
6.1 Laboratory Tank tests

To validate the simulation implementation, there were performed multiple tests with the UX-1 robot in a laboratory tank environment. The tests include the validation of:

- **Dynamics:** Simple dynamics adjust meant to assimilate real robot dynamics recorded inside tank;
- **Control:** Thruster control tests with both simulation control over robot and robot real control over simulation;
- **Imaging:** Imaging sensors validation, that include Micron DST sonar, M3 multibeam sonar, Cameras and SLS with distorted cameras.
- **Navigation:** DVL and IMU sensors data validation and comparison.
- **Other systems:** Pendulum and Ballast system validation.

The selected object for sonar sensors validation was a metal anchor. The anchor was placed in the middle of the laboratory tank at 1.8m depth. It was secured with a cord,

as seen in figure 6.1. In all comparison experiments, the simulated robot was positioned in the same position as a real robot experiment. This information was extracted from navigation software and injected in simulation for a fair comparison.



(a) (b)

(c)

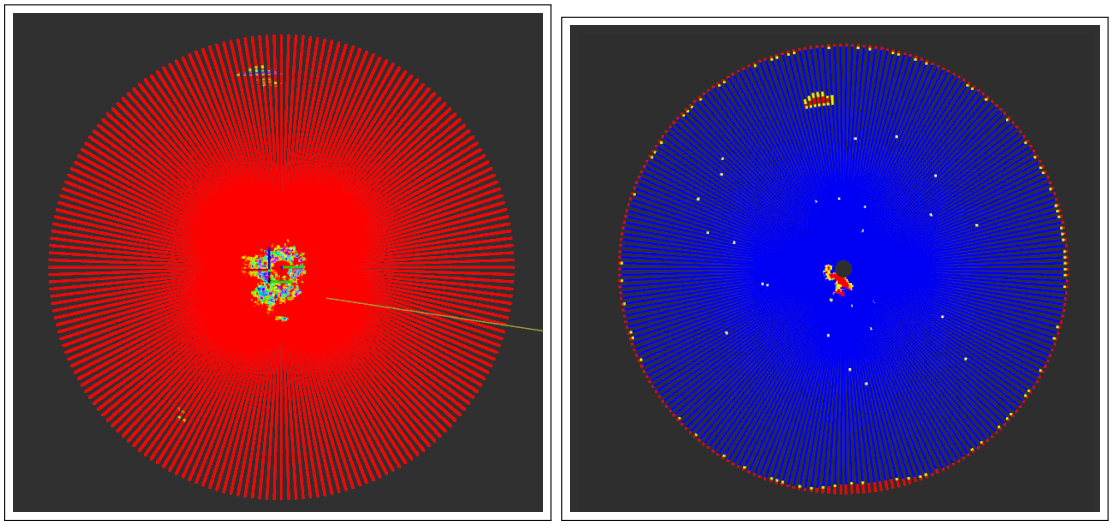
Figure 6.1: UX-1 robot in laboratory test tank.

6.1.1 Micron DST

Simulated Micron DST sensor is positioned in the same location as the robot as a real UX-1 robot. For a fair comparison, both simulated and real sensor have the same configurations:

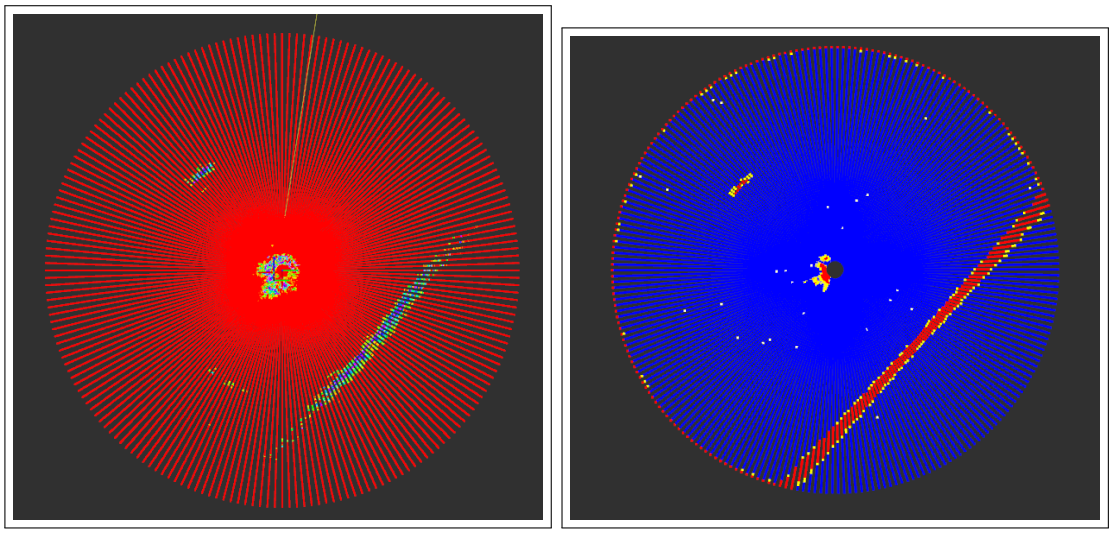
- **Speed:** 1500;
- **Range:** 3 meters;
- **nbins:** 400;
- **Continuous(360º):** True;
- **Gain:** Low.

Figures 6.2, 6.3 and 6.4represent pointcloud comparison between simulated and real sonar results in distinct positions. The real sensor represent more noise in center and it gets higher when the robot is closer to surface due to water reflections. As seen in the figures, from different angles, the shape of the object is represented distinctly in real and simulated sonar.



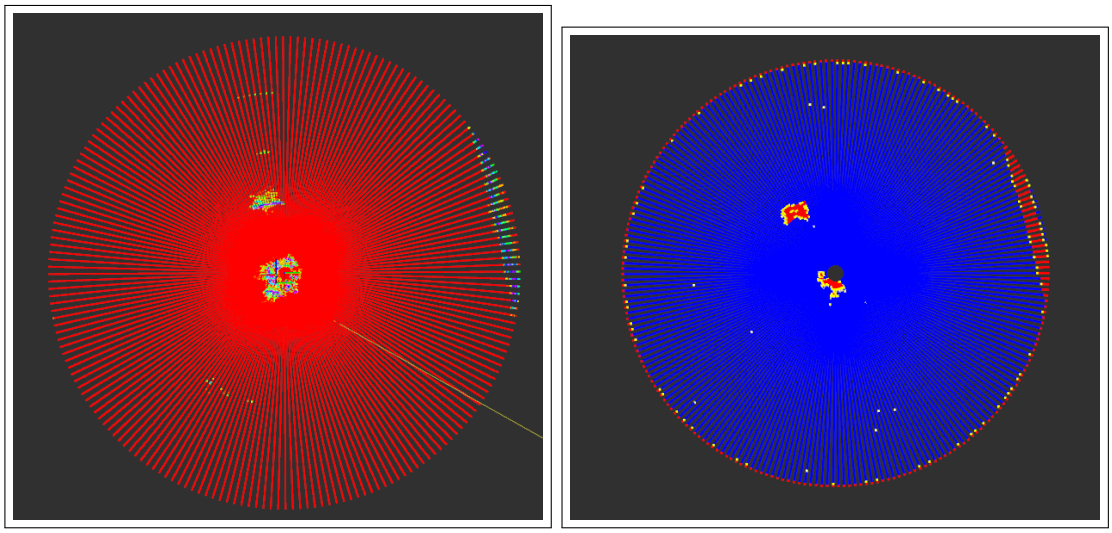
(a) Real (b) Simulated

Figure 6.2: Micron DST pointcloud with long range object, (a) Real and (b) Simulated.



(a) Real (b) Simulated

Figure 6.3: Micron DST pointcloud with medium range object, (a) Real and (b) Simulated.



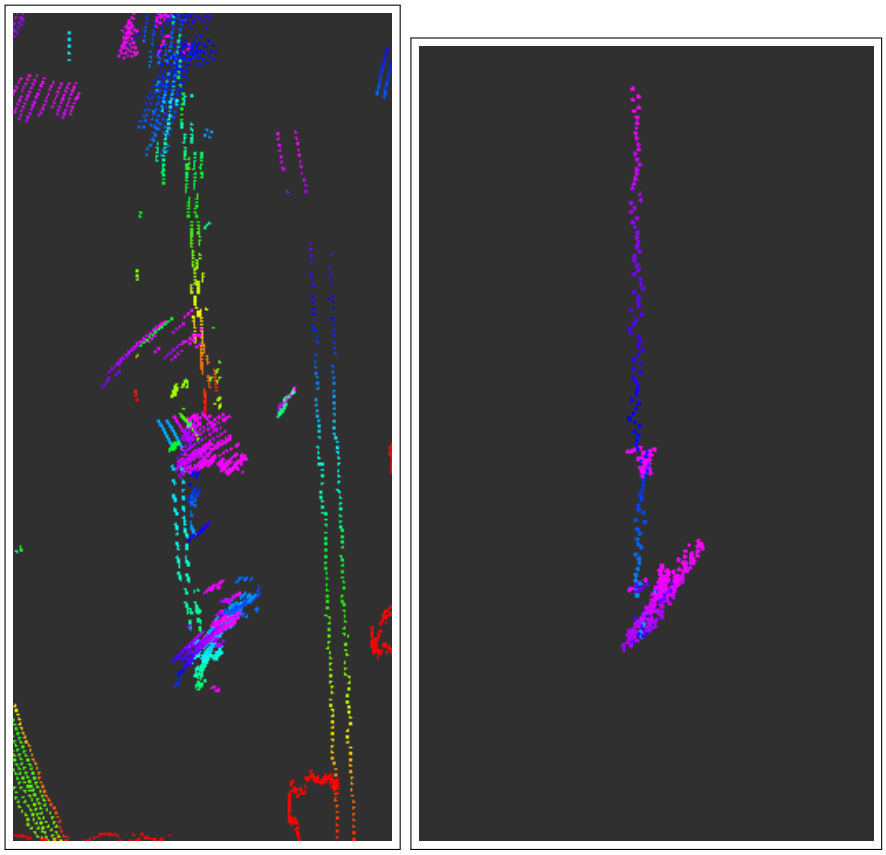
(a) Real (b) Simulated

Figure 6.4: Micron DST pointcloud with close range object, (a) Real and (b) Simulated.

6.1.2 M3 multibeam

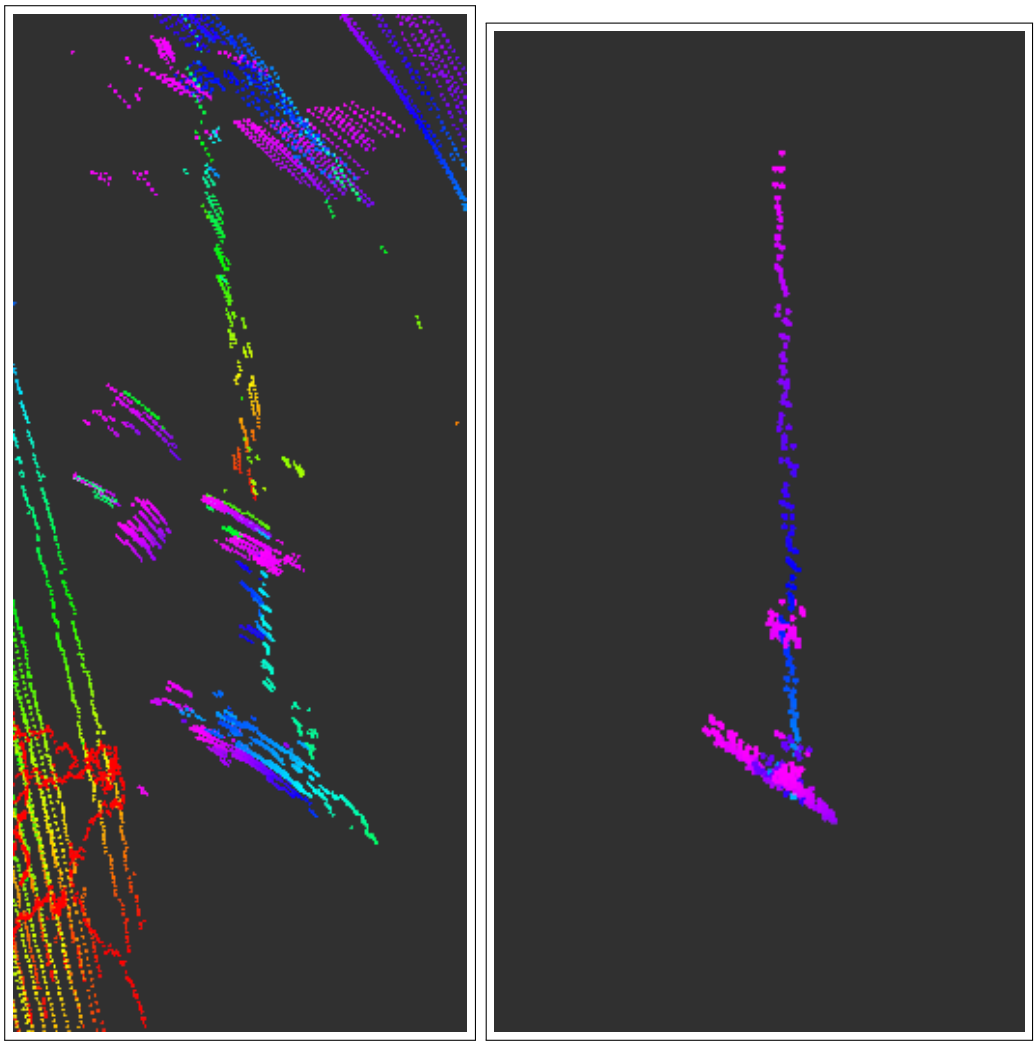
The M3 multibeam sensor was compared in a laboratory tank with the anchor as a target. In order to obtain an M3 multibeam scan area, there is an obligation to move/rotate the robot for the desired scan area. Scanned areas are same in simulated and real tests due to navigation information obtained from the real robot.

Sensor range was set to 5 meters and the results are represented in figures 6.5, 6.6, **??** and **??**. The real sonar have higher noise than the simulated one. The noise is higher in real sonar image due to multipath echoes relfections inside laboratory tank and configurations used in this test. The shape of the object is detected and observable in both real and simulated sonar images, with similar results.



(a) Real (b) Simulated

Figure 6.5: Simulated M3 multibeam pointcloud, (a) Real and (b) Simulated.



(a) Real (b) Simulated

Figure 6.6: Simulated M3 multibeam pointcloud, (a) Real and (b) Simulated.

6.2 SLS

SLS system comparison was performed in laboratory water tank with appropriate lightning. Results are presented in figures 6.7, 6.8, 6.9 and 6.10. The position of the robot and laser rotation is roughly the same in simulation and tank tests. As observed in figures, the simulated SLS system have high fidelity and is quite similar to the real one. The variations of laser line intesity in real SLS system is due to slight innacuracy on focus. In ideal situation, it should have same intensity in the laser line. In figure 6.11 is represented the simulated laser in Rviz.

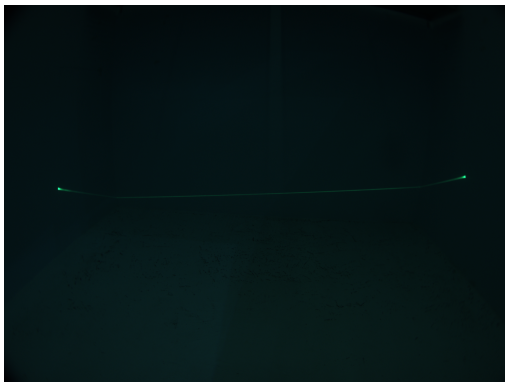


Figure 6.7: Real SLS laser image.

Figure 6.8: Simulated SLS laser image.

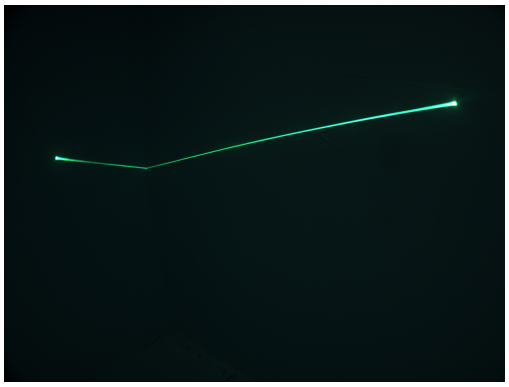


Figure 6.9: Real SLS laser image.

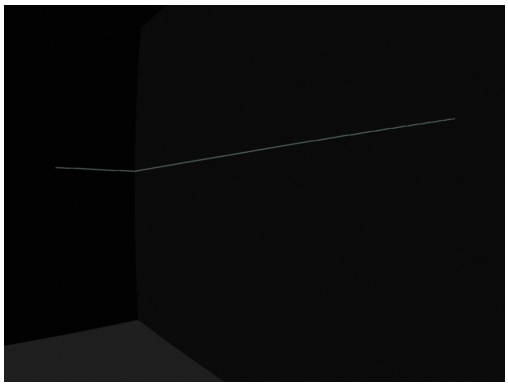


Figure 6.10: Simulated SLS laser image.

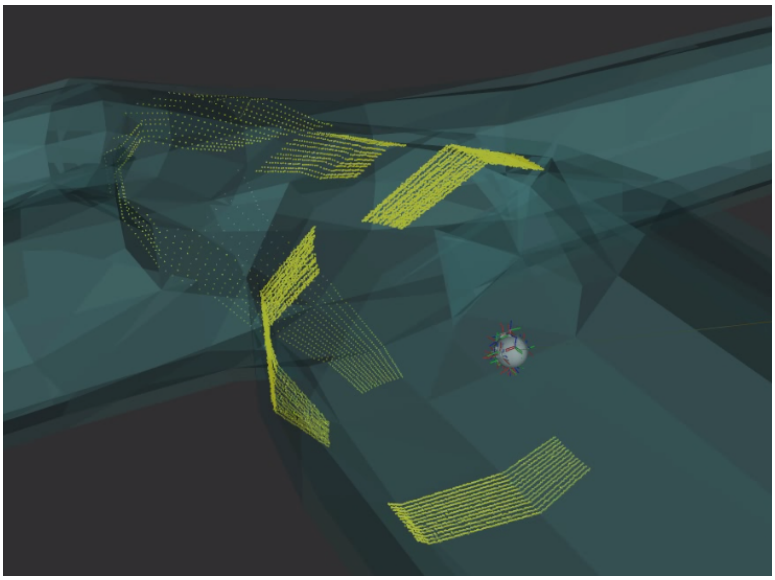


Figure 6.11: Simulated representation of 4 SLS lasers in Rviz.

6.3 Mapping

With SLS and M3 sensors is possible to obtain a 3D mapping of mine environment. Octomaps[28] are a great tool for 3D map generation and are easy to use in navigation software. In order to test the mapping feature of simulation, 4 SLS sensors and M3 Kingsberg sensor were used in conjunction with Octomaps. Pointclouds from each sensor were added into one and then processed with Octomaps, to create a global 3D map of the environment. One section of test map are represented in figures 6.12 and 6.13.

Kaatiala mine is located in the province of Western Finland, Kuortane town. Mine information:

- Max depth: 353 m;
- Max lenght: 620 m;
- Max width: 30 m;
- Size: 218860 m^2.

A model of Kaatiala mine was implemented inside the simulation. Kaatiala mine is one of the test mines in UNEXMIN project. Figure 6.14 represents the Kaatiala mine model inside the simulated world. This model include real size, visual and colision parameters to assimilate the real mine environment. In figure 6.15 is represented octomap of the Kaatiala mine.

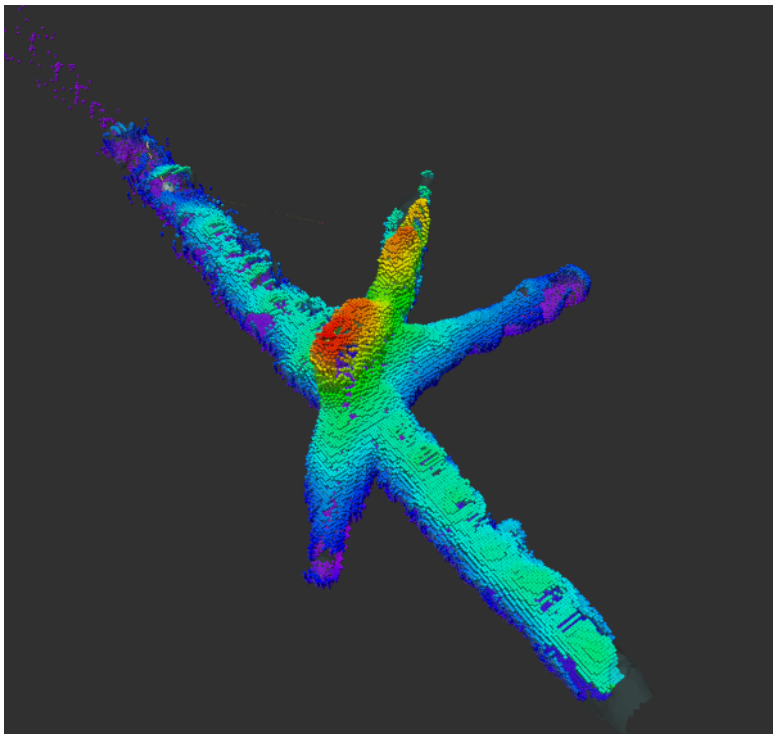


Figure 6.12: Octomap of a section of a test mine.

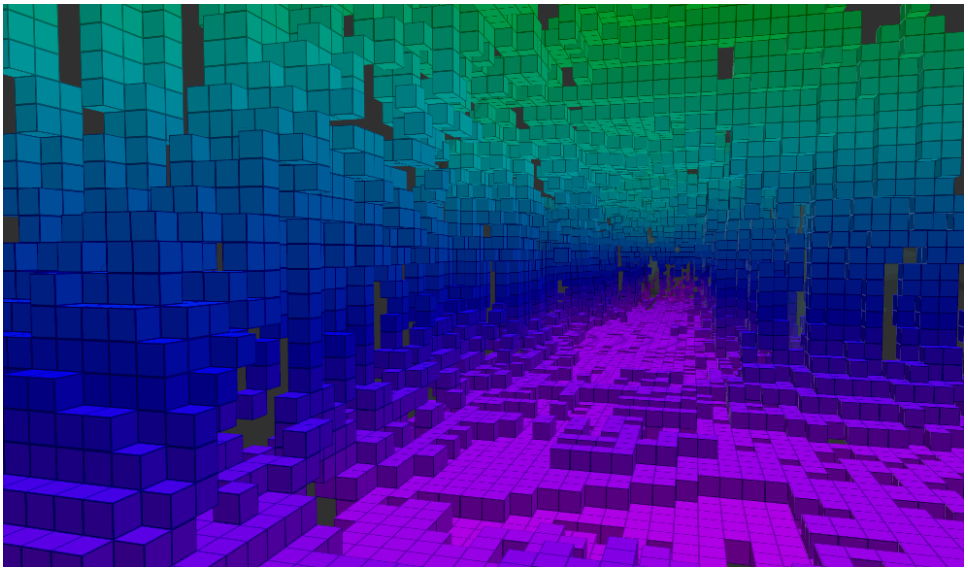


Figure 6.13: Octomap inside of test mine.

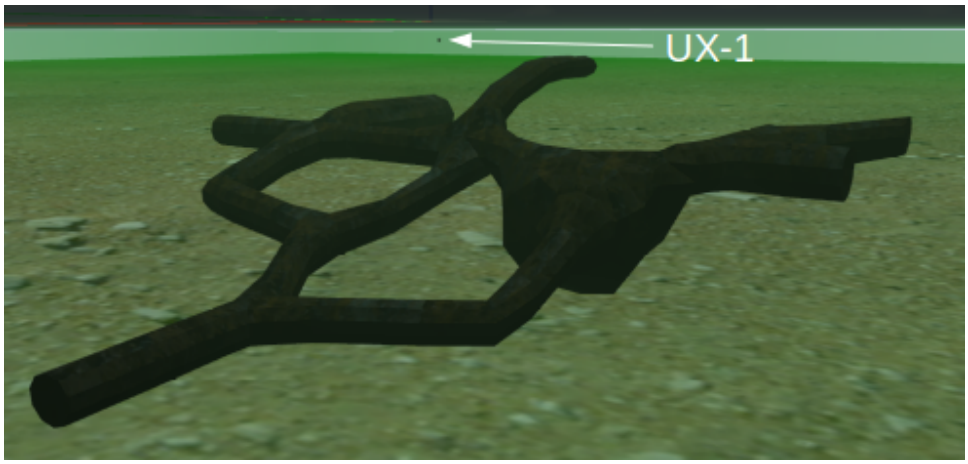


Figure 6.14: Kaatiala mine in simulation world.

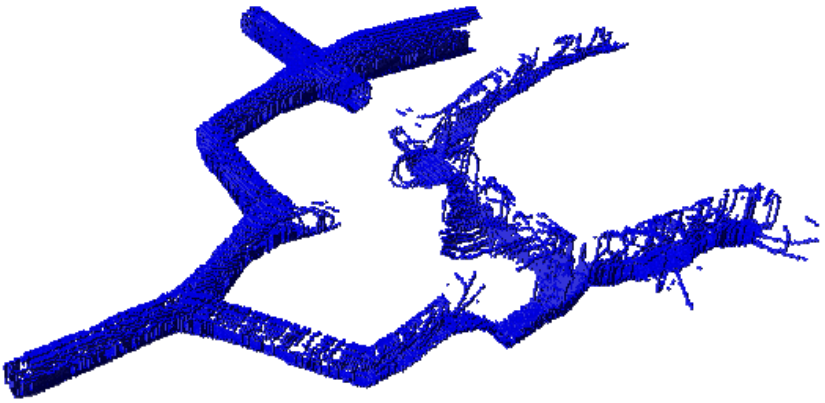


Figure 6.15: Octomap of a section of Kaatiala mine.

6.4 Performance

All tests were performed in a computer with following specifications:

- Intel(R) Core(TM) Quad-Core i5-3210M CPU @ 2.50GHz;
- Nvidia GeForce 630m graphic card;
- 6GB RAM;

- 500GB HDD.

In order to optimize simulation performance, operational and visual modes were implemented. The only difference between the modes is the addition of *Attached Lights* plugin and *Projector* plugin, which can add up to 40 Frames Per Second (FPS) in the simulation. These modes can be easily switched before the simulation start.

Next table represents performance with an operational mode for the multi-robot platform:

Operational Mode	**1 Robot**	**2 Robots**	**3 Robots**
CPU	80%	100%	100%
RAM	2.3GB	2.4GB	2.5GB
Simulation FPS	55	42	30
Simulation RTF	0.7	0.2	0.1

Simulated RTF represents Real Time Factor between the simulation time and real time. Where the simulation time is equal to real time is defined by value=1.

6.5 Navigation

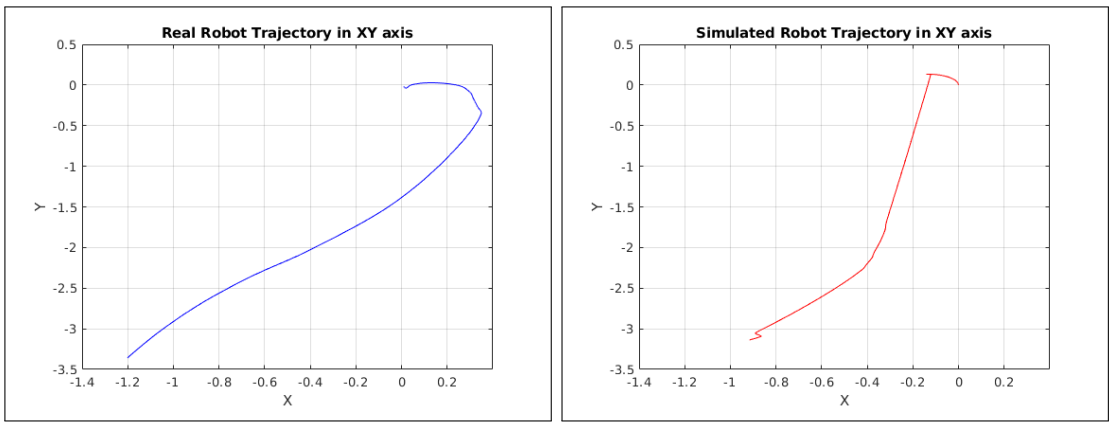
As mentioned before, the simulation has two navigation modes that interact with the real robot (hardware in the loop). The first mode consists in simulation software controlling directly real robot movement, performing the same trajectory as in simulation world, in real time. This mode uses *thruster_allocator_sim* plugin.

Second mode is providing from a ROS bag or in real time, the trajectory performed by real robot is replicated in simulation environment. This mode uses *vesc_sim_node* plugin. Figures 6.16, 6.17 ,6.18, and 6.19 represent second mode results.The trajectory was performed in real robot in laboratory tank with X, Y, Z and Yaw remote control.

Both modes work with no additional software conversions between real robot software and simulation. All names and information are the same as a real robot, and simulation package converts the information into respective nomenclatures for proper control with no conflicts involved.

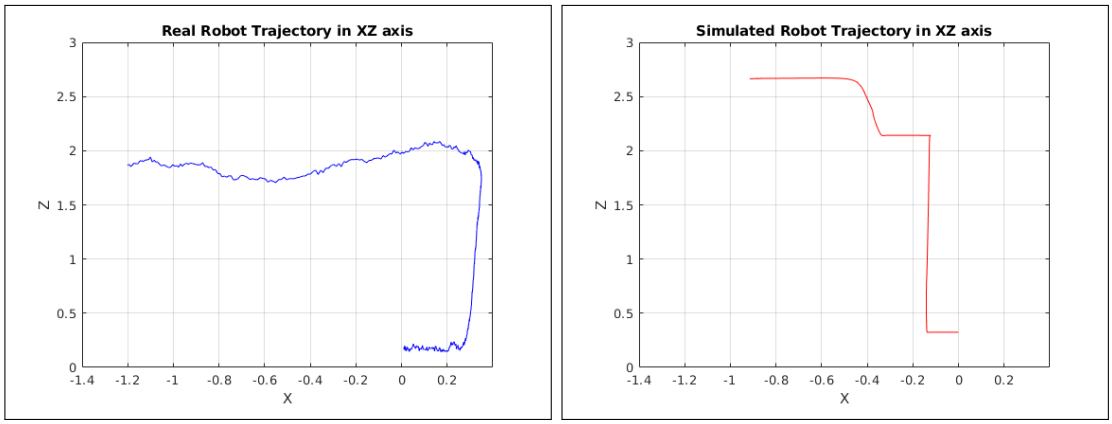
In the simulation, the robot has neutral buoyancy as it is supposed to be in real life robot, but due to weight management, new hardware implementations, and weight distributions, the real robot in experiments was slightly more positive buoyant. Because of irregular weight distribution, the vehicle was also slightly inclined in pitch and roll,

that made the center of mass not equal to the center of geometry. All that aspects made difference between the real and simulated trajectories.



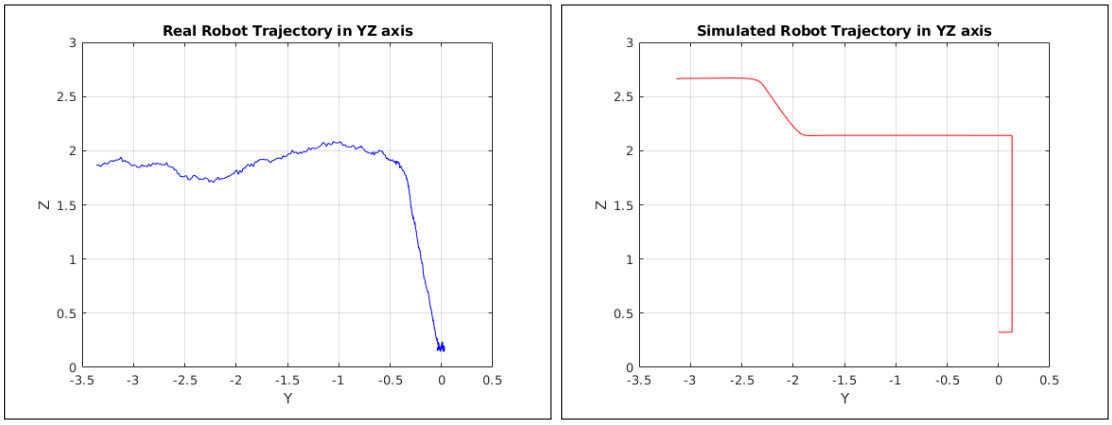
(a) Real (b) Simulated

Figure 6.16: Robot trajectory in XY axis, (a) Real and (b) Simulated.



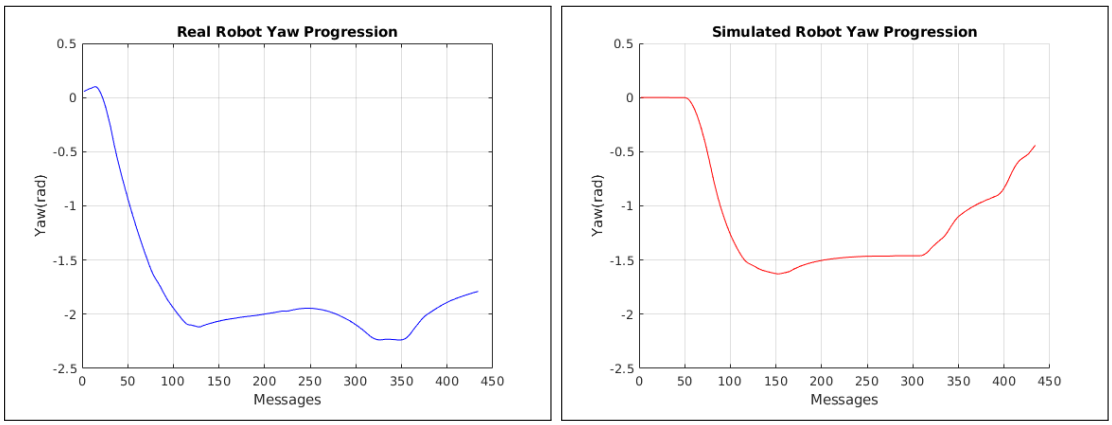
(a) Real (b) Simulated

Figure 6.17: Robot trajectory in XZ axis, (a) Real and (b) Simulated.



(a) Real (b) Simulated

Figure 6.18: Robot trajectory in YZ axis, (a) Real and (b) Simulated.



(a) Real (b) Simulated

Figure 6.19: Robot yaw plot, (a) Real and (b) Simulated.

This page was intentionally left blank.

Chapter 7

Conclusion and Future Work

A simulation platform for the UNEXMIN project was completed with the main objectives accomplished. With final tests, it was possible to validate the fidelity of the simulator and demonstrate the comparison between simulated and real results from various sensors. This information is important for future software developments and will indicate the level of proximity between the simulator and real robot.

The performance of the simulation platform is adequate when compared to other simulators, and with all the sensor models, environment, and hydrodynamics the recorded performance is acceptable even for real-time simulations. The only problem starts with the addition of multi-robots simulation, in which case, the performance drops significantly with every added robot. This is however a problem also existing in many other high fidelity 3D multi-robot simulation environments. Although, all tests were performed on a laptop with low specifications, compared to new CPU's. With a better computer, the performance of the simulation would increase greatly and make multi-robot simulations more usable.

Hardware in loop navigation tests confirmed the simulation capabilities of performing trajectories similar to the real robot with the same input commands. As expected, those trajectories are not completely equal due to the real robot imperfections during the test and the simulation dynamics. The robot during the tests was a prototype and had a slightly miss-placed center of the mass than the simulated one and had offset in pitch (14^{o}) and roll (8^{o}) which made the most difference between the real robot performed trajectory and simulated one. Also, the simulated dynamics model is simplified and for more accurate results, a more accurate dynamic model can be implemented. Although, it can take significant development time to create a highly reliable underwater dynamic model in the Gazebo simulator (which has no underwater plugins).

With the constant updates of the Gazebo simulator, the developed models of every aspect in the simulation can be further optimized and improved for better performance.

With all available modules in the simulation platform, it is possible to create or test software for underwater missions. Those can include confined mine environments or open pit mines.

Developed sensor models are universal, for the purpose of the UNEXMIN project, those sensors have been adapted to simulate the sensors attached in a real robot. Although, the same sensor models can be adapted to different models of the same type sensor. With the use of Xacro files and universal code type, it is simple to change the sensor specifications.

As stated in UNEXMIN simulation chapter, this simulation platform includes multi-robot models with individual control and sensor models. In UNEXMIN project those robots are strictly the same, but with little effort, it is possible to change the robot model for a completely different underwater vehicle type. Other robot models like EVA and MARA can be implemented in the simulator with little effort, because of system files distribution and description method. The only information needed is the visual mesh and values for the dynamics of the vehicle. The addition of the sensors, thrusters or other models are possible and can be placed in the desired position of the robot without the need of creating complex new models and structures. The possibilities even extend to the point of simulating different types of robots in the same environment, for distinct multi-robot missions.

The work described in this thesis was already resulted in the publishing of a research paper in the IEEE Oceans conference[29].

Bibliography

[1] Gazebo gazebo-ros packages. http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros. Accessed: 2018-10-18.

[2] VAMOS project description. http://www.unexmin.eu/. Accessed: 2018-10-08.

[3] UNEXMIN project description. http://vamos-project.eu//. Accessed: 2018-09-30.

[4] KVH 1775 description. https://www.kvh.com/Commercial-and-OEM/Gyros-and-Inertial-Systems-and-Compasses/Gyros-and-IMUs-and-INS/IMUs/1775-IMU.aspx. Accessed: 2018-10-24.

[5] Nortek DVL 1000 description. https://www.nortekgroup.com/products/dvl1000-4000m. Accessed: 2018-10-24.

[6] Kongsberg M3 sonar description. https://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/C2D49096683E47EFC125783E002C6E0F?OpenDocument. Accessed: 2018-10-24.

[7] Tritech Micron DST sonar description. https://www.tritech.co.uk/product/small-rov-mechanical-sector-scanning-sonar-tritech-micron. Accessed: 2018-10-24.

[8] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51. IEEE, 2011.

[9] Jeff Craighead, Robin Murphy, Jenny Burke, and Brian Goldiez. A survey of commercial & open source unmanned vehicle simulators. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 852–857. IEEE, 2007.

[10] Adam Harris and James M Conrad. Survey of popular robotics simulators, frameworks, and toolkits. In *Southeastcon, 2011 Proceedings of IEEE*, pages 243–249. IEEE, 2011.

[11] Patricio Castillo-Pizarro, Tomás V Arredondo, and Miguel Torres-Torriti. Introductory survey to open-source mobile robot simulation software. In *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*, pages 150–155. IEEE, 2010.

[12] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[13] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1400–1405. IEEE, 2007.

[14] Olivier Kermorgant. A dynamic simulator for underwater vehicle-manipulators. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 25–36. Springer, 2014.

[15] Kevin J DeMarco, Michael E West, and Ayanna M Howard. A simulator for underwater human-robot interaction scenarios. In *Oceans-San Diego, 2013*, pages 1–10. IEEE, 2013.

[16] Thomas Tosik and Erik Maehle. Mars: A simulation environment for marine robotics. In *Oceans-St. John's, 2014*, pages 1–7. IEEE, 2014.

[17] Andreas Birk, Gianluca Antonelli, Andrea Caiti, Giuseppe Casalino, Giovanni Indiveri, Antonio Pascoal, and Andrea Caffaz. The co 3 auvs (cooperative cognitive control for autonomous underwater vehicles) project: Overview and current progresses. In *OCEANS, 2011 IEEE-Spain*, pages 1–10. IEEE, 2011.

[18] Mario Prats, Javier Pérez, J Javier Fernández, 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.

[19] J. J. Fernandez, J. Perez, A. Penalver, J. Sales, D. Fornas, and P. J. Sanz. Benchmarking using UWSim, Simurv and ROS: An autonomous free floating dredging

intervention case study. *MTS/IEEE OCEANS 2015 - Genova: Discovering Sustainable Ocean Energy for a New World*, 2015.

[20] Weijia Yao, Wei Dai, Junhao Xiao, Huimin Lu, and Zhiqiang Zheng. A simulation system based on ROS and Gazebo for RoboCup Middle Size League. *2015 IEEE International Conference on Robotics and Biomimetics, IEEE-ROBIO 2015*, pages 54–59, 2016.

[21] 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*. IEEE, sep 2016.

[22] Jeremi Gancet, Peter Weiss, Gianluca Antonelli, Max Folkert Pfingsthorn, Sylvain Calinon, Alessio Turetta, Cees Walen, Diego Urbina, Shashank Govindaraj, Pierre Letier, et al. Dexterous undersea interventions with far distance onshore supervision: The dexrov project. *IFAC-PapersOnLine*, 49(23):414–419, 2016.

[23] marinesimulation. http://marinesimulation.com/. Accessed: 2018-09-30.

[24] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[25] FreeCAD. https://www.freecadweb.org/. Accessed: 2018-10-08.

[26] Blender. https://www.blender.org/. Accessed: 2018-10-08.

[27] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[28] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[29] D Sytnyk, R Pereira, D Pedrosa, J Rodrigues, A Martins, A Dias, J Almeida, and E Silva. Simulation environment for underground flooded mines robotic exploration. In *OCEANS 2017-Aberdeen*, pages 1–6. IEEE, 2017.