Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

Vision-based Autonomous Navigation for the BlueROV2 Underwater Vehicle

Panagiotis Siolis

Thesis committee

Professor Michail G. Lagoudakis (ECE)

Professor Michail Zervakis (ECE)

Professor Panagiotis Partsinevelos (MRE)

Chania, July 2023

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Αυτόνομη Πλοήγηση βασισμένη σε Όραση για το Υποβρύχιο Όχημα BlueROV2

Παναγιώτης Σιώλης

Εξεταστική Επιτροπή

Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Καθηγητής Μιχαήλ Ζερβάκης (ΗΜΜΥ)

Καθηγητής Παναγιώτης Παρτσινέβελος (ΜΗΧΟΠ)

Χανιά, Ιούλιος 2023

Abstract

The use of Simultaneous Localization And Mapping (SLAM) algorithms is widespread in the field of robotics, especially when referring to ground robotic vehicles. SLAM algorithms, based on visual sensor information provided by a camera, require the inclusion of a procedure known as calibration, namely acquisition or estimation of all camera parameters required for the SLAM algorithm to work properly. In this diploma thesis, we focus on the use of visual SLAM algorithms, not on ground robotic vehicles, but on Remotely-Operated underwater robotic Vehicles (ROVs). In particular, a visual SLAM approach has been developed for the BlueROV2, which is small-size underwater robot, used for ocean research and exploration missions up to a depth of 100m. The proposed approach relies on the ORB-SLAM3 algorithm and is adapted for onboard execution on the ROV, using the Robot Operating System (ROS) framework. The successful deployment of our approach required two hardware modifications on the BlueROV2: replacement of the pre-installed Raspberry Pi 3 embedded computer with the more powerful Raspberry Pi 4 and replacement of the pre-installed monocular camera with an Intel RealSense T265 stereo camera to utilize the capabilities of ORB-SLAM3. At the same time, a control algorithm is proposed for the movement of the ROV, which is able to perform various motion patterns, such as moving along a line, a rectangle, a circle or a spiral, passing through points provided by the user. The combination of the proposed SLAM and motion control approaches make the vehicle able to move in an unknown environment without obstacles, with only minimal user intervention. Results were obtained through extensive simulations in water environments, but also in a real indoor environment, nevertheless outside the water, since the modified BlueROV2 is not yet 100% waterproof. In any case, the proposed approach enables successful navigation, as long as a sufficient number of visual features are identified in the environment.

Περίληψη

Η χρήση αλγορίθμων ταυτόχρονου εντοπισμού και χαρτογράφησης (Simultaneous Localization And Mapping - SLAM) είναι ευρέως διαδεδομένη στον τομέα της ρομποτικής, ειδικά όταν αναφερόμαστε σε ρομποτικά οχήματα εδάφους. Οι αλγόριθμοι SLAM που βασίζονται σε οπτικές πληροφορίες που παρέχονται από μία κάμερα, απαιτούν τη συμπερίληψη μιας διαδικασίας γνωστής ως βαθμονόμησης, δηλαδή την απόκτηση ή την εκτίμηση όλων των παραμέτρων της κάμερας που απαιτούνται για τη σωστή λειτουργία του αλγόριθμου SLAM. Στην παρούσα διπλωματική εργασία, εστιάζουμε στη χρήση αλγορίθμων visual SLAM, όχι σε επίγεια ρομποτικά οχήματα, αλλά σε υποβρύχια ρομποτικά οχήματα που μπορούν να ελεγχθούν εξ αποστάσεως (Remotely-Operated Vehicles - ROVs). Συγκεκριμένα, έχει αναπτυχθεί μια προσέγγιση visual SLAM για το BlueROV2, το οποίο είναι ένα μικρού μεγέθους υποβρύχιο ρομπότ που χρησιμοποιείται για αποστολές έρευνας και εξερεύνησης ωκεανών μέχρι το βάθος των 100 μέτρων. Η προτεινόμενη προσέγγιση βασίζεται στον αλγόριθμο ORB-SLAM3 και είναι προσαρμοσμένη για εκτέλεση επάνω στο ROV, χρησιμοποιώντας το Robot Operating System (ROS) framework. Η επιτυχής ανάπτυξη της προσέγγισής μας απαιτούσε δύο τροποποιήσεις υλικού στο BlueROV2: αντικατάσταση του προεγκατεστημένου ενσωματωμένου υπολογιστή Raspberry Pi 3 με τον πιο ισχυρό Raspberry Pi 4 και αντικατάσταση της προεγκατεστημένης μονο-κάμερας με στερεο-κάμερα Intel RealSense T265 για αξιοποίηση των δυνατοτήτων του ORB-SLAM3. Ταυτόχρονα, προτείνεται ένας αλγόριθμος ελέγχου για την κίνηση του ROV, ο οποίος είναι σε θέση να εκτελεί διάφορα μοτίβα κίνησης, όπως κίνηση κατά μήκος μιας γραμμής, ενός τετραγώνου, ενός κύκλου ή μιας σπείρας, περνώντας από σημεία που παρέχει ο χρήστης. Ο συνδυασμός των προτεινόμενων προσεγγίσεων SLAM και ελέγχου κίνησης καθιστά το όχημα ικανό να κινείται σε ένα άγνωστο περιβάλλον χωρίς εμπόδια, με ελάχιστη μόνο παρέμβαση του χρήστη. Τα αποτελέσματα προέκυψαν μέσα από εκτενείς προσομοιώσεις σε

υδάτινα περιβάλλοντα, αλλά και σε ένα πραγματικό εσωτερικό περιβάλλον, ωστόσο εκτός νερού, εφόσον το τροποποιημένο BlueROV2 δεν είναι ακόμα 100% αδιάβροχο. Σε κάθε περίπτωση, η προτεινόμενη προσέγγιση επιτρέπει την επιτυχή πλοήγηση, αρκεί να εντοπιστεί επαρκής αριθμός οπτικών χαρακτηριστικών στο περιβάλλον.

ACKNOWLEDGEMENTS

To begin, I would like to thank my professors, Michail G. Lagoudakis and Panagiotis Partsinevelos, for their help and guidance. I would also like to thank the researchers at SenseLab, especially Angelos Antonopoulos and Stathis Bikos, for their guidance in my thesis. Finally, I would like to thank my friends and fellow students who have been by my side all this time, and of course, my parents, who support me all the time and are by my side every step of the way, both in the easy and the difficult ones.

Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Thesis Motivation

The use of robots for both ground and aerial applications is particularly widespread nowadays among the general public. However, robots are also used for underwater missions, and their use is becoming increasingly popular among both the research community and hobbyists. Remotely operated underwater vehicles, or "ROVs", have a wide range of applications. Some of these are seabed mapping, exploration in inaccessible places and even assistance in rescue missions. In some of the applications we mentioned, such as rescue, losing communication with the vehicle during the mission can be disastrous. The use of autonomous navigation by such vehicles thus becomes essential for such scenarios.

1.2 Thesis Contribution

This thesis proposes a solution to the problem of autonomous navigation of such vehicles by using a Simultaneous Localization and Mapping (SLAM) algorithm based on vision. The SLAM algorithm used is ORB-SLAM3, an open source SLAM algorithm that supports stereo vision camera and IMU. Knowing the position of the ROV from SLAM and the coordinates of the target, the underwater vehicle can be moved with the help of the motion control algorithm implemented for autonomous navigation.

1.3 Thesis Outline

In Chapter 2, we present all the background information needed for this thesis. We refer to basic knowledge of concepts, like computer vision and SLAM, as well as describing the software packages used.

In Chapter 3, we refer to the basic problem that this thesis proposes to solve, while related work is also presented.

In chapter 4 we describe our approach to the problem and the steps we took to accomplish the goal of this thesis.

In Chapter 5, we present the results in the simulator, as well as in a real-world indoor environment. We also discuss the reason why we do not have results from a real-world underwater environment.

Finally, in Chapter 6, we conclude and suggest future work to extend our approach.

Chapter 2

Theoretical Background

2.1 Robot Operating System (ROS)

The Robot Operating System (ROS) [3] is an open-source set of software libraries and tools that help community of engineers, developers and hobbyists to build their robots. ROS is not an operating system. ROS is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime. It simplifies many of the challenges of robotics development, such as sensor integration, communication between different components, and managing multiple robots. ROS also has a large and active community that contributes to the development of libraries, tools, and algorithms, making it a valuable resource for roboticists.

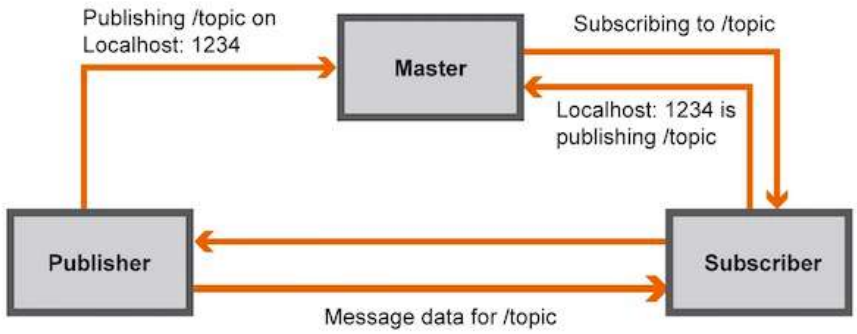

Figure 2.1: ROS architecture diagram. Image created by Yahya Tawil

ROS supports multiple programming languages, including C++, and Python, allowing developers to choose the language that best suits their needs. ROS follows a publish-subscribe messaging model, known as the ROS communication

infrastructure, where nodes (individual software modules) can publish and subscribe to messages on specific topics. This decoupled architecture allows for easy integration of various sensors, actuators, and algorithms, making it ideal for complex robotic applications.

ROS MASTER

The ROS Master is a crucial component of the ROS communication infrastructure. It acts as a centralized coordination point for all nodes in a ROS system. The ROS Master manages the registration and discovery of nodes, facilitates communication between nodes, and provides other essential services. When a ROS node starts, it typically registers with the ROS Master, informing it about its presence, name, and the topics it publishes or subscribes to. This registration allows other nodes in the system to discover and communicate with the newly registered node.

NODES

Each node in ROS can communicate with other nodes by sending and receiving messages. This communication is facilitated by the ROS communication infrastructure, which follows a publish-subscribe model. Nodes can publish messages on specific topics, and other nodes can subscribe to those topics to receive the messages.

TOPICS

In Robot Operating System (ROS), topics are a communication mechanism that allows nodes to exchange messages with each other. Topics facilitate the exchange of data between nodes in a publish-subscribe manner. They are a fundamental component of the ROS communication infrastructure.

SERVICES

Services provide a request-response communication mechanism between nodes. They enable nodes to send specific requests to other nodes and receive corresponding responses. ROS services allow for synchronous, bi-directional communication, where the requesting node waits for a response before proceeding further.

2.2 Sensors

2.2.1 Inertial Measurement Unit (IMU)

An Inertial Measurement Unit (IMU) is a device that consists of multiple sensors to measure accelerations and angular velocities in multiple axes. It can also integrate these measurements to determine orientation and position over time. IMUs are commonly used in a wide range of applications including aerospace, robotics, virtual reality, and autonomous vehicles.

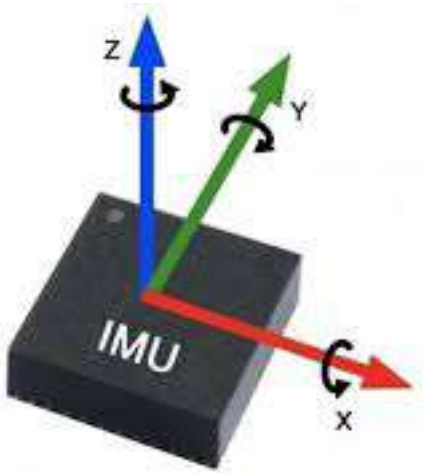

Figure 2.2: Inertial Measurement Unit (IMU), source: Utmel Electronic

The low price of an IMU sensor compared with the data which provides, makes it one of the main options in these applications. The inertial measurement unit can be divided in four components. Accelerometer that provides the acceleration along the 3-axes, gyroscope that provides the angular velocity about the axes, magnetometer that measures the magnetic field and a barometer that measures air pressure.

The IMU sensor, when the main goal is the position estimation, operates additionally with other type of sensors due to accumulated error. For example, double integration of the noisy acceleration leads to an inaccurate position.

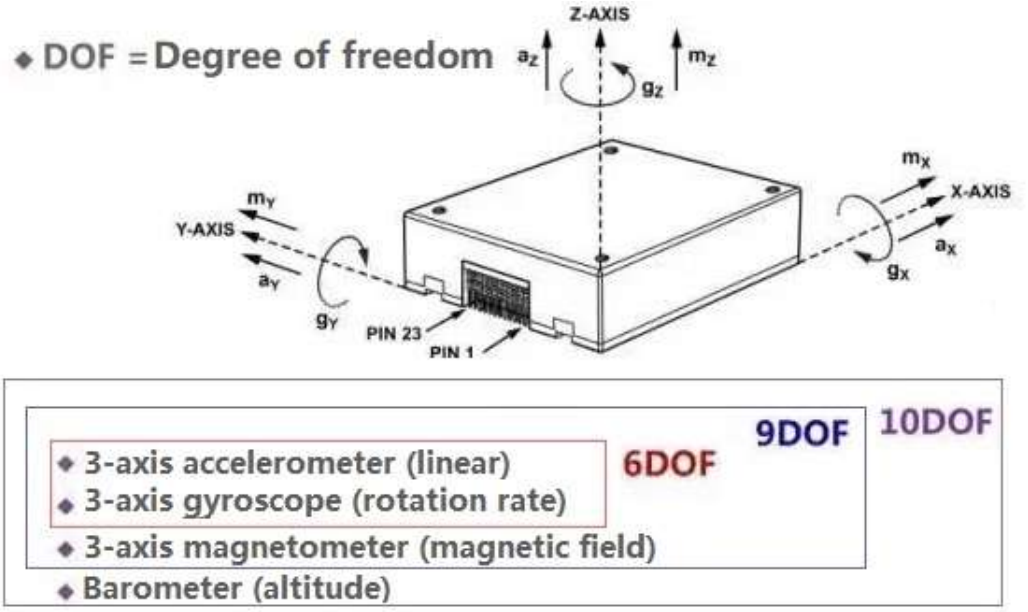

Figure 2.3: IMU's Degree of freedom, source: Utmel Electronic

2.2.2 Camera

This section describes how a point in three-dimensional space can be transformed into a 2D pixel and how to extract the intrinsic parameters of the camera [4].

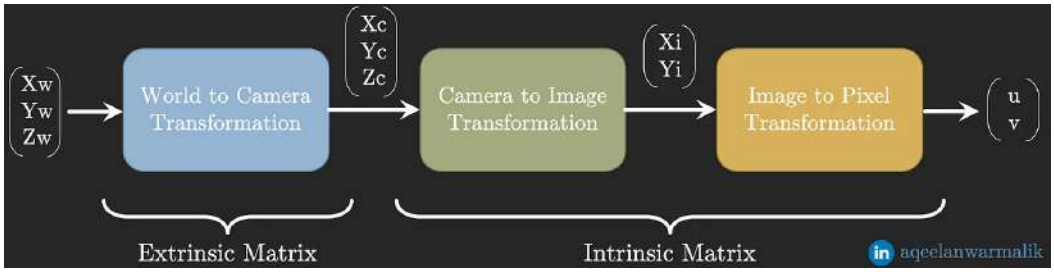

Figure 2.4: From three-dimensional point to pixel, image by Aqeel Anwar

Assume a random point in space $P_w = (X_w, Y_w, Z_w)$ in a Cartesian coordinate system with an arbitrary origin. In a second coordinate system with origin the center of camera, the point which was selected previously has new coordinates $P_c = (X_c, Y_c, Z_c)$. Conversion from the first coordinate system to the camera coordinate system is possible through the Camera Extrinsic Matrix.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$\mathbf{P} = \begin{bmatrix} \mathbf{R}|\mathbf{t} \end{bmatrix}$, $\mathbf{P}$ includes a rotation matrix ($\mathbf{R}$) and a translation vector ($\mathbf{t}$)

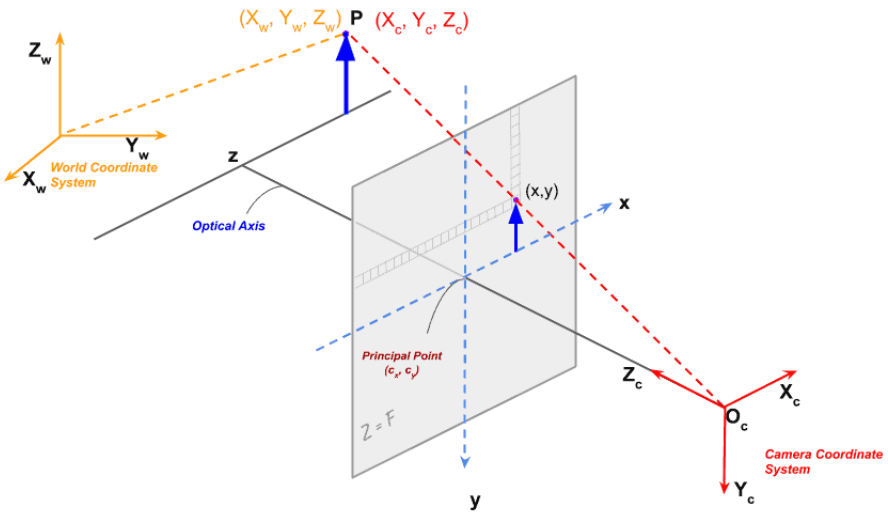

Figure 2.5: Pinhole camera model, source: learnopencv

Next, the point will be reflected onto the 2D plane. After passing through the center of the camera hole, the rays are projected onto the 2D plane, resulting in the point in 3D space projected by the reflected rays. The new coordinates of the point in 2D can be found by the law of similar triangles. The distance between the 2D plane and camera hole is called the focal length (f).

By the law of similar triangles: $\frac{X_i}{f} = \frac{X_c}{Z_c}$, $\frac{Y_i}{f} = \frac{Y_c}{Z_c}$

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Where the matrix containing the focal length is the camera to image transformation matrix.

The last part is the discretization. To do that, divide the image coordinates by the pixels' width and height (pixel's width and height denoted as p_u and p_v respectively) .

$$x_i = \frac{1}{p_u} \cdot X_i \Rightarrow x_i = \frac{1}{p_u} \cdot f \cdot \frac{X_c}{Z_c}$$
$$y_i = \frac{1}{p_v} \cdot Y_i \Rightarrow y_i = \frac{1}{p_u} \cdot f \cdot \frac{Y_c}{Z_c}$$

Define $\frac{1}{p_u} \cdot f = f_x$ and $\frac{1}{p_v} \cdot f = f_y$. f_x and f_y are two of the intrinsic parameters and are the focal length of the camera in pixels for the x and y axes, respectively.

Because the origin of the pixel coordinate system is in the left-top corner, a translation operator (c_x,c_y) is also necessary in addition to the discretization.

$$u = x_i + c_x, v = y_i + c_y,$$

where c_x and c_x are the other two intrinsic parameters and are the optical center of the camera in pixels for the x and y axes, respectively.

The last transformation matrix has the form seen below :

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \frac{1}{p_u} & 0 & c_x \\ 0 & \frac{1}{p_v} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_c \end{bmatrix}$$

Now the process has been completed and a point has been converted from 3D space to pixel coordinate system.

2.2.3 Stereo Camera

Figure 2.6: Stereo Camera, source: theimagingsource

There are applications in which we are interested in knowing the depth of an image, like in localization and mapping cases. However, the basic function of cameras, as discussed above, is to project the three-dimensional space onto a two-dimensional plane. Various solutions have been proposed for the problem of depth estimation; here, we analyze the use of stereo cameras for depth estimation. A camera with two or more lenses and a separate image sensor or film frame for each lens is called a stereo camera. Due of this, the camera is able to replicate human binocular vision.

By projecting a point in space onto the two-dimensional plane, we lose the depth information due to this projection. We cannot recover information about the position of the original point with a monocular camera.

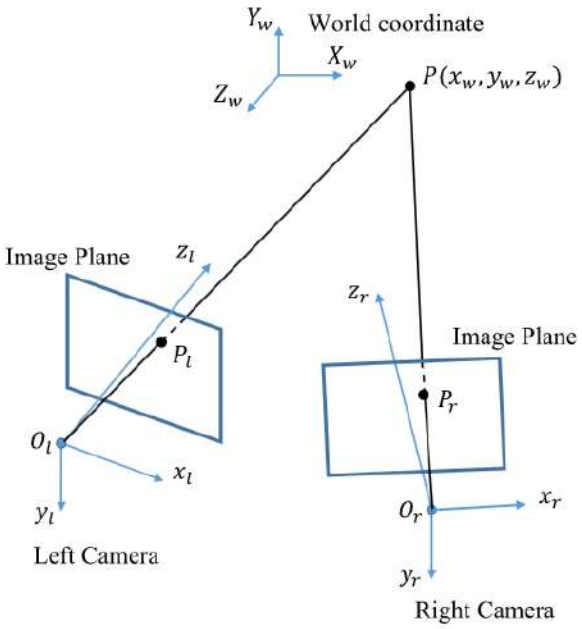

Figure 2.7: Pinhole Stereo Camera [1]

Assume we place an identical second camera at a distance (b) from the first camera on the x-axis and that there is a point P (x_w, y_w, z_w) in both cameras' fields of view, with projections P_l and P_r for the left and right cameras, respectively. The origin of the axes of the two cameras is O_l for the left and O_r for the right.

Knowing O_l and point P_l, we find the line that passes through these points and certainly passes through point P, but we do not know the exact coordinates of P. Similarly for the second camera, knowing O_r and P_r, we find a second line that also passes through point P. Therefore, we can now find the coordinates of the point, by finding where these two lines intersect.

The projections' coordinates (P_l and P_r) are calculated as follows:

$$P_l(u_l, v_l) = P_l(f_x \cdot \frac{X_c}{Z_c} + c_x, f_y \cdot \frac{Y_c}{Z_c} + c_y) \ , \ P_r(u_r, v_r) = P_r(f_x \cdot \frac{X_c - b}{Z_c} + c_x, f_y \cdot \frac{Y_c}{Z_c} + c_y)$$

f_x, f_y, c_x, c_y, b are known.

Solving for x,y,z :

$$X_c = \frac{b \cdot (u_l - c_x)}{(u_l - u_r)}$$

$$Y_c = \frac{b \cdot f_x \cdot (v_l - c_y)}{f_y \cdot (u_l - u_r)}$$

$$\boxed{Z_c = \frac{b \cdot f_x}{(u_l - u_r)}}$$

2.3 Simultaneous Localization And Mapping (SLAM)

In autonomous navigation and path planning tasks, in both indoors and outdoors environments, the robot needs the ability to localize itself based on the constructed maps. In cases in which the environment is unknown, the use of SLAM (simultaneous localization and mapping) algorithms is necessary. SLAM finds wide application in automatic car piloting, rescue, exploration and many more scenarios.

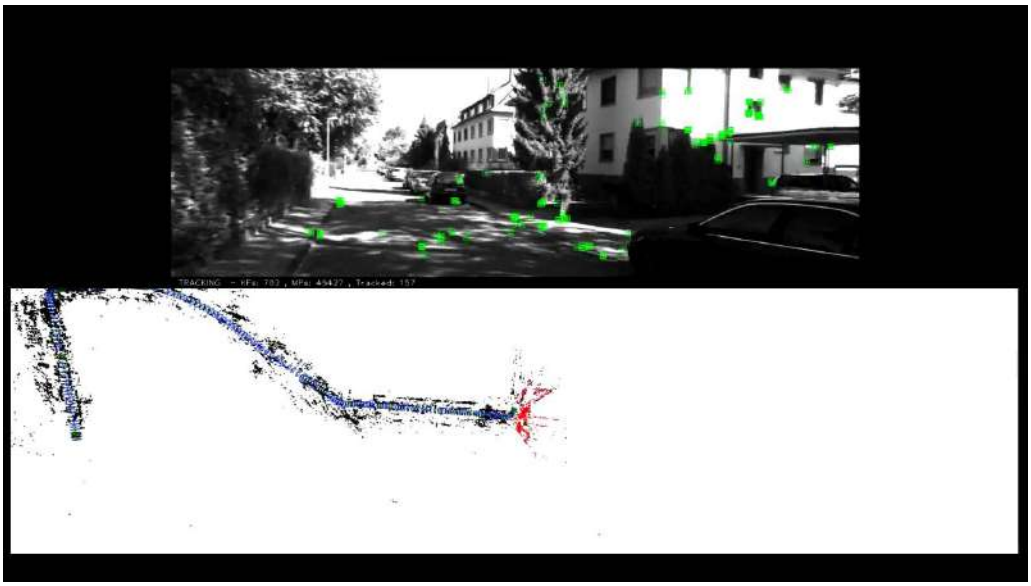

Figure 2.8: An example of SLAM in an outdoor environment, source: Universidad Zaragoza

The process typically involves two main steps: mapping and localization [5],[6]. In Mapping, the device builds a map of the environment by creating a set of features from the sensor data, such as corners or edges in a camera image, and registering these features with the current position of the device. Over time, the map becomes more complete and accurate, as the device collects more

sensor data. In Localization, the device determines its position within the map by comparing the current sensor data with the map and finding the best match between the two. This process is repeated in real-time as the device moves, allowing it to continuously update its location and the map.

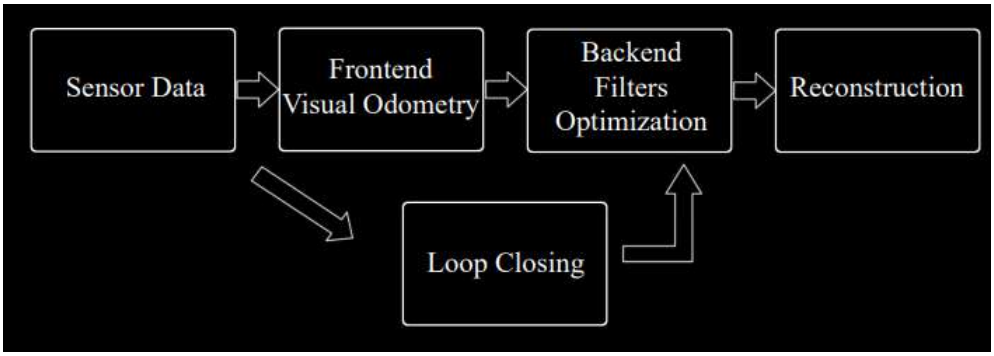

Figure 2.9: Stages from sensor's data to SLAM estimation, source: Github slambook-en.pdf

The sensors used in SLAM are typically separated in two main caterories: interoceptive sensors and exteroceptive sensors. Interoceptive sensors generate relative position measurements whose error is used cumulatively in measurement uncertainty. Such sensors are wheel odometers and IMUs. Exteroceptive sensors provide absolute position measurements. In Exteroceptive sensors are included acoustic sensors, lasers and visuals sensors. As acoustic sensors, sonars are mostly used underwater. Lasers have high speed and accuracy, however the price is the usual stumbling block. On condition, visual sensors mainly refer to monocular, stereo and RGB-D cameras. The two main sensors categories, if used alongside each other, they could compensate for errors, like odometry drift.

2.3.1 ORB-SLAM3

ORB-SLAM3 is a feature-based SLAM that is able to perform visual-inertial SLAM with monocular, stereo, and RGB-D cameras in both indoors and outdoors

environments. The term “feature-based SLAM” refers to SLAM that minimizes the feature reprojection error by extracting a set of sparse features from the input images, matching the features derived from various poses, and solving the SLAM issue. Simple point characteristics, like corners, more complicated features, like edges and blobs, and even complex things, like doorways and windows, are among the features that are of interest.

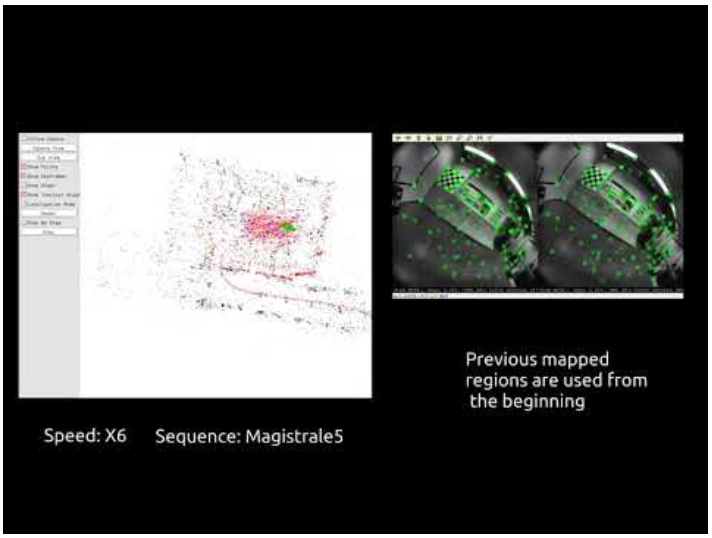

Figure 2.10: ORB-SLAM3 running with a stereo camera, source: GitHub - UZ-SLAMLab/ORB_SLAM3

The function of the algorithm can be divided into 3 parts: tracking, local mapping and loop closing. Tracking: This determines where the camera is in each frame and when to add fresh keyframes to the map. Keyframes keep track of the camera postures and the extracted ORB features that reduce information duplication. If it becomes lost, it will search the previously saved maps for the scenario; if not, it will create a new one. Local mapping: enhances the active map by adding keyframes and points, removes the redundant ones, and refines the map. Loop closing: The process of observing the same scene by non-adjacent frames and adding a constraint between them, significantly reducing the accumulated drift in the pose estimate.

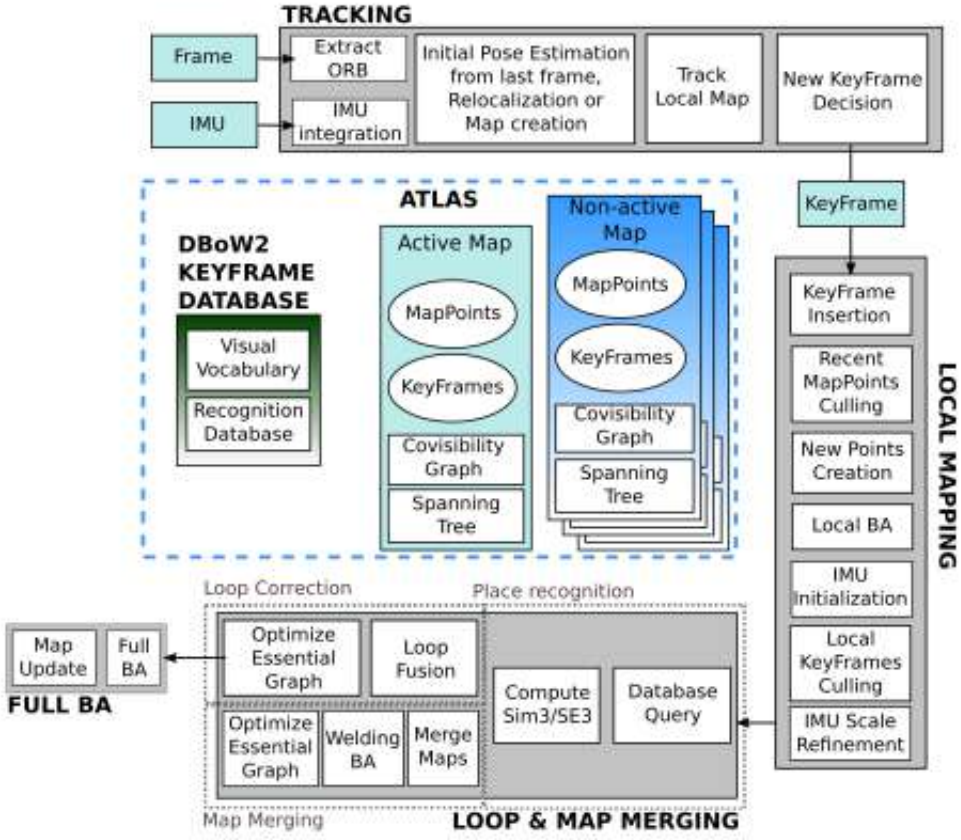

Figure 2.11: Diagram of ORB-SLAM3 [2]

2.4 SITL

SITL, or Software-in-the-Loop, is a type of simulator used in the field of robotics and unmanned aerial vehicles (UAVs). SITL simulators allow engineers and developers to test and develop software in a virtual environment, without the need for physical hardware. The SITL simulator operates by creating a virtual environment that mimics the physical behavior of the UAV or robot. The simulator uses models of the physical system, such as aerodynamic models for UAVs or kinematic models for robots, to simulate the movement and behavior of the system. The software being tested is then integrated with the simulated environment, allowing the developer to test the software's functionality in a realistic and controlled setting. SITL simulators are useful in a variety of applications, such as testing autonomous flight algorithms for UAVs or developing control software for robotic manipulators. Figure 2.12 shows SITL running on the computer, simu-

lating BlueROV2. On the left, the ROV, which is in bold blue, is located at a random point chosen on the map as a starting point. On the top right is the terminal screen running SITL, and on the bottom right is the console showing us useful information about the ROV (battery power, speed, depth, etc.).

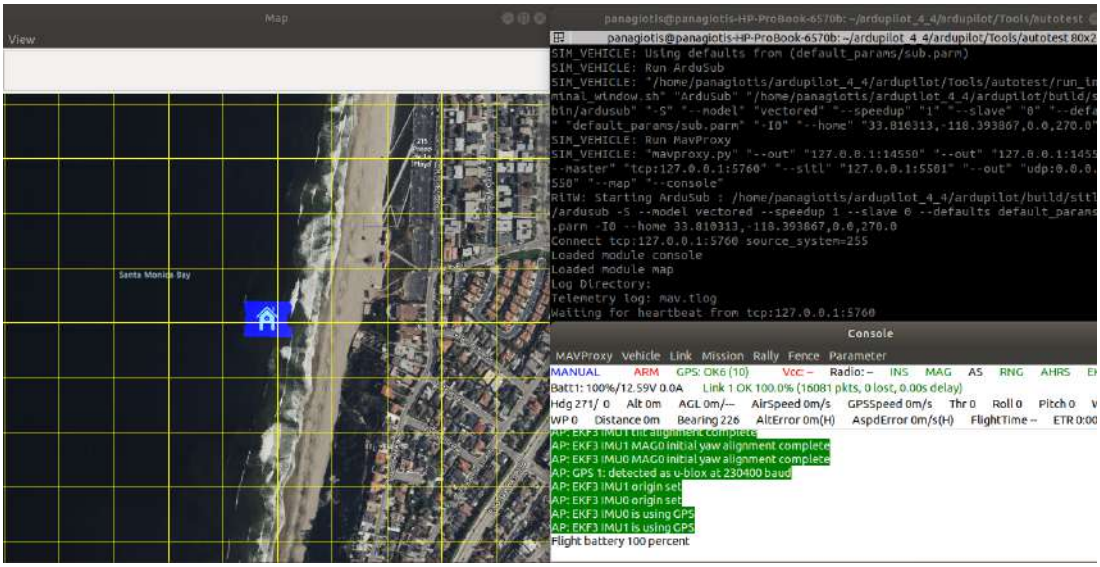

Figure 2.12: SITL with BlueROV2 model

2.5 Gazebo-Simulator

Gazebo Simulator is an open-source software tool that enables users to simulate and test their robotics and autonomous systems. Gazebo is based on a physics engine that simulates the behavior of objects in a 3D environment, making it an ideal tool for testing robotic systems. It allows users to create virtual environments that mimic real-world situations, enabling them to test their systems in a safe and controlled environment. One of the key benefits of using Gazebo is its flexibility. It supports a wide range of sensors and actuators, allowing users to simulate a variety of different robots and systems. It also provides a range of customization options, allowing users to adjust the parameters of their simulations to meet their specific needs. Gazebo is also widely used in the robotics community, because of its integration with other software tools. It can be used with ROS

(Robot Operating System). Figure 2.13 shows BlueROV2 in the gazebo world we created.

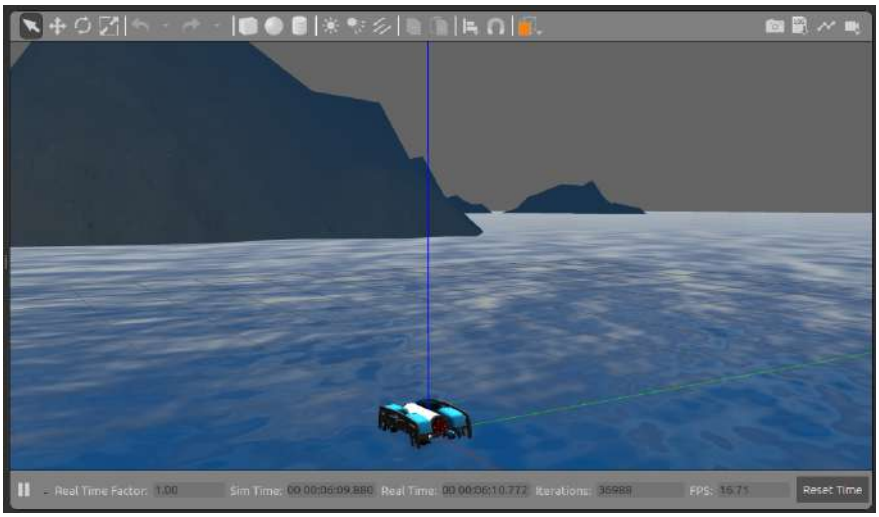

Figure 2.13: A Gazebo world with the BlueROV2 model

Chapter 3

Problem Statement

3.1 BlueROV2

BlueROV2 is an underwater remotely operated vehicle (ROV) designed for use in marine research, exploration, and inspection tasks. It is a compact and modular ROV that is capable of diving to depths of up to 100 meters (328 feet) and provides high-quality video and data collection capabilities. The BlueROV2 has a durable, watertight design and is equipped with six thrusters for precise movement and control. It is also equipped with a suite of sensors, including high-resolution camera, IMU, and other instruments for collecting data about the underwater environment. The BlueROV2 is designed to be operated remotely and can be deployed quickly and easily for a variety of missions. The ROV can be configured to meet the specific requirements of each mission, making it a versatile tool for ocean research and exploration.

A Raspberry Pi 3 is located inside the BlueROV2's as main control board and is wired via Ethernet to a top-side PC. The electrical tube also has a Pixhawk in it in addition to the Raspberry Pi 3. A Fathom-X Tether Interface, which offers a long-distance Ethernet connection over a single twisted pair of wires, is used to transfer the Ethernet communication top-side. On the front side a 1080p USB camera is mounted and on the back side there is a pressure sensor.

Figure 3.1: BlueROV 2, source: Blue Robotics

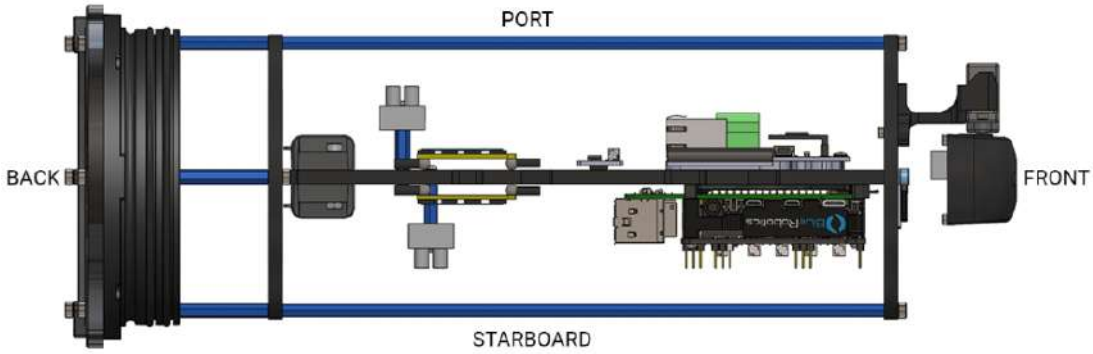

Figure 3.2: BlueROV2 inside (Top View), source: Blue Robotics

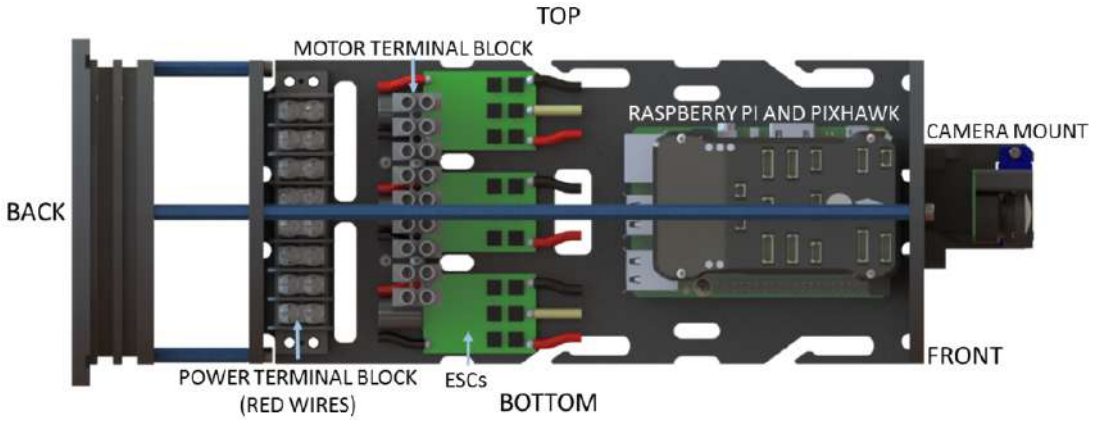

Figure 3.3: BlueROV2 inside (Starboard View), source: Blue Robotics

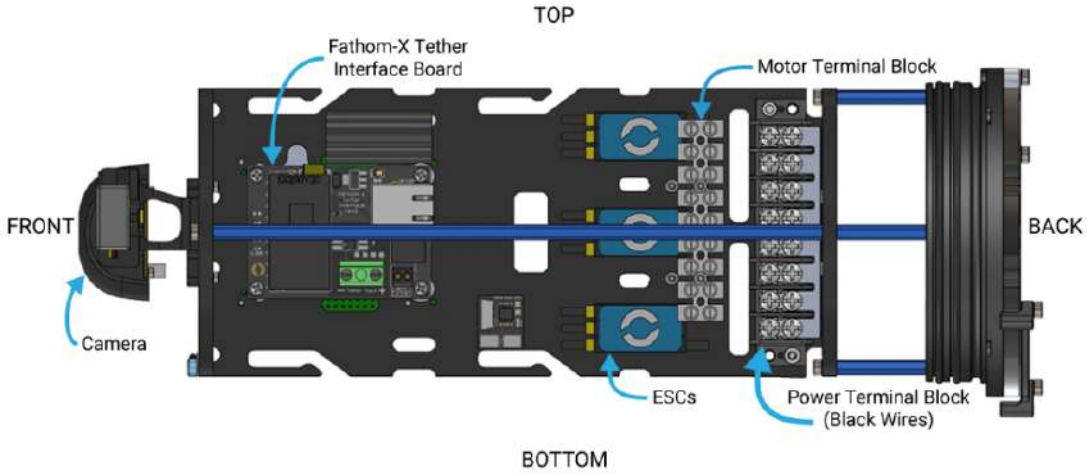

Figure 3.4: BlueROV2 inside (Port View), source: Blue Robotics

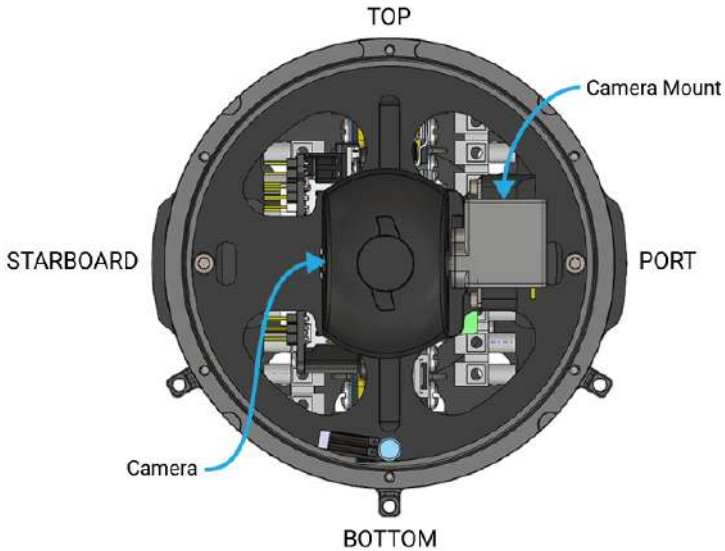

Figure 3.5: BlueROV2 inside (Front View), source: Blue Robotics

3.1.1 Pixhawk

Pixhawk 1 is an advanced autopilot system that runs the ArduSub control software for the BlueROV2, also supports the ArduPilot, ArduCopter, ArduRover software. Pixhawk 1 is based on the PX4 autopilot system and uses a 32-bit ARM Cortex-M4F processor for fast and efficient control. It also has a wide range of sensors, including accelerometers, gyroscopes, magnetometers, barometers, and GPS. A MPU6000 is used as the main accelerometer and gyroscope. One of the key features of Pixhawk 1 is its modularity and flexibility. It has a wide range of compatible peripherals and accessories, allowing users to customize their vehicle for their specific needs. Additionally, it can be programmed with different modes and mission parameters, making it suitable for a variety of applications in the world of autonomous vehicles.

Figure 3.6: Pixhawk 1 Flight Controller, source: Blue Robotics

3.1.2 Connection with Computer

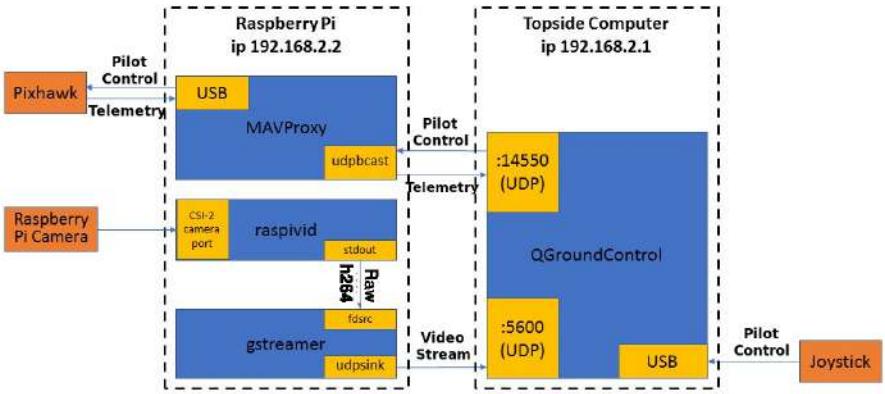

Figure 3.7: BlueROV2 communication diagram, source: Blue Robotics

The communication of BlueROV2 with the computer is achieved through a single twisted pair of wires on two ports: on port 14550, the data from the pixhawk is sent via the MavLink protocol with MAVROS, while the image is sent on port 5600, as we can see in the diagram. Using the QGroundControl program, we have a visualization of the data received from the ROV, both from the camera and the IMU.

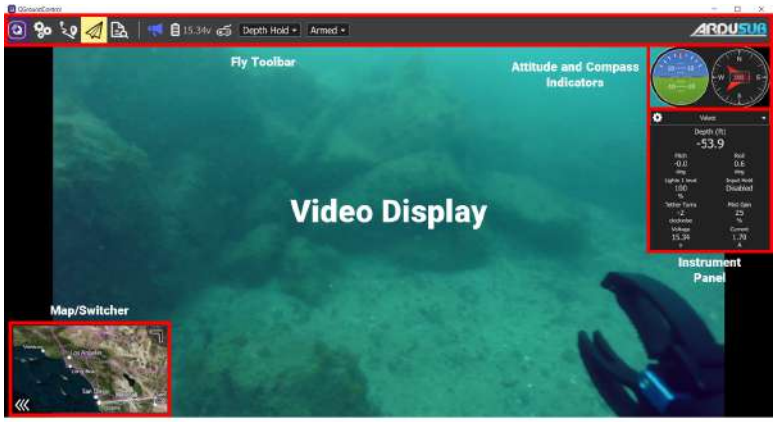

Figure 3.8: QGroundControl

3.2 Underwater Localization and Mapping

For an autonomous underwater vehicle (AUV) to perform underwater missions, it is necessary to know its position, map the unknown environment, and, of course, move autonomously in this space. The use of GPS, when we are below the sea surface, which is the focus of this thesis, is not possible. The receiver cannot pick up the electromagnetic signals from the orbiting satellites, because they are significantly dampened in water. Our goal in this thesis is to solve the problems of localization and mapping with a SLAM algorithm based on vision for the BlueROV2 Underwater Vehicle.

3.3 Related Work

From the literature, we identified other approaches to the problem of localization and mapping, such as underwater GPS (UGPS) and the use of SLAM with sonar. Underwater GPS [7],[8] uses base stations on the sea surface or on board a ship that know their position via GPS. They then communicate with the ROV via acoustic signals and can calculate its exact position. In the case of underwater SLAM, as we read in the literature, the main sensor used is the sonar. Several versions have been proposed. One case is the use of Doppler Velocity Log (DVL) and ring gyro as complementary sensors [9], and there is a publication [10] with visual, inertial, and depth sensors in parallel with sonar. ORB-SLAM2 modified to enable acoustic odometry estimation [11] has also been used.

Chapter 4

Our Approach

4.1 SLAM algorithm selection

There are many SLAM open-source algorithms that one can find and experiment with, like MapLab, LDSO, VINS-Mono, and OpenVINS. However, the one that was finally chosen for this thesis is ORB-SLAM3. The criteria that were chosen are that it supports stereo-inertial SLAM, it supports ROS [12], and it has real-time performance without the need for GPUs. The possibility of not using a GPU is very important in applications like ours because it reduces the processing power requirements, so we gain in energy consumption and can be implemented in embedded systems like the Raspberry Pi we used. Furthermore, ORB-SLAM3 shows good results [13] in different environments, sensor setups and even with dynamic objects. The reasons for that are a quality implementation and both fast and reliable visual features and the back-end part of the algorithm.

4.2 First approach

Our first approach was an architecture which consists of the following steps: The data is sent to the top-side computer. Then the ORB-SLAM3 reads the camera and IMU's data from the appropriate ROS topics and subsequently the odometry messages, which is the output data of the SLAM, are sent back to the ROV to update its position coordinates.

Disadvantages:

- The data is sent to the computer and returned to the ROV by the same twisted pair cable, but the bandwidth is limited, so to avoid losing data we need to reduce the refresh rate of the sensor values.
- If communication with the computer is lost, the ROV loses its position.

Advantages:

- The top-side computer is more powerful, so the algorithm can use more frames per second and does not lose localization in sudden direction changes.

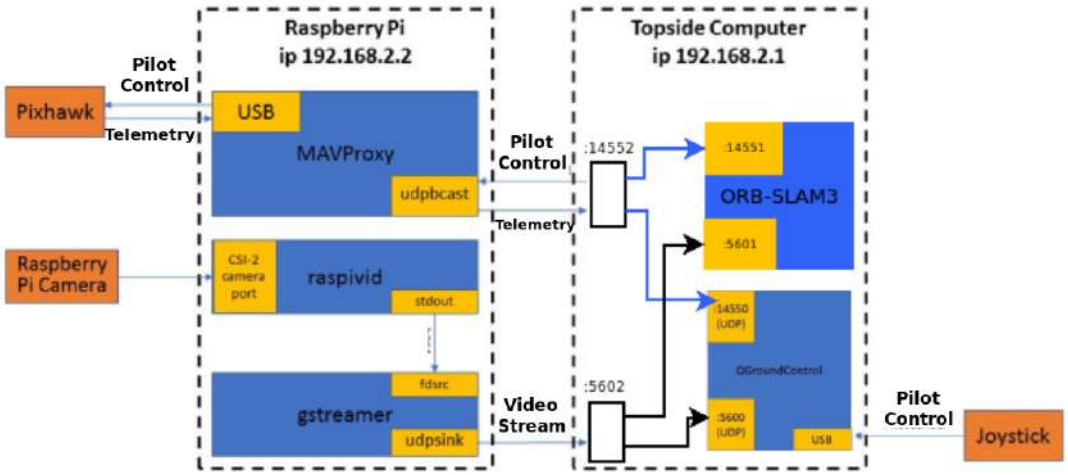

Figure 4.1: Communication diagram of the first approach

4.3 Proposed approach

Trying to solve the problems of the previous implementation, I ended up changing both the architecture of the communication between the ROV and the top-side computer and upgrading the ROV's hardware.

4.3.1 Proposed architecture

The ORB-SLAM3 will now run on the main board of the ROV, so the vehicle does not need external intervention for localization and mapping, which reduces the amount of data that needs to be sent. Therefore, the data from the sensors is sent to the Raspberry Pi, where the SLAM algorithm takes over to process the camera and IMU data. The odometry message updates the coordinates of the position of the vehicle itself and is also sent to the top-side computer in order to visualize its movement.

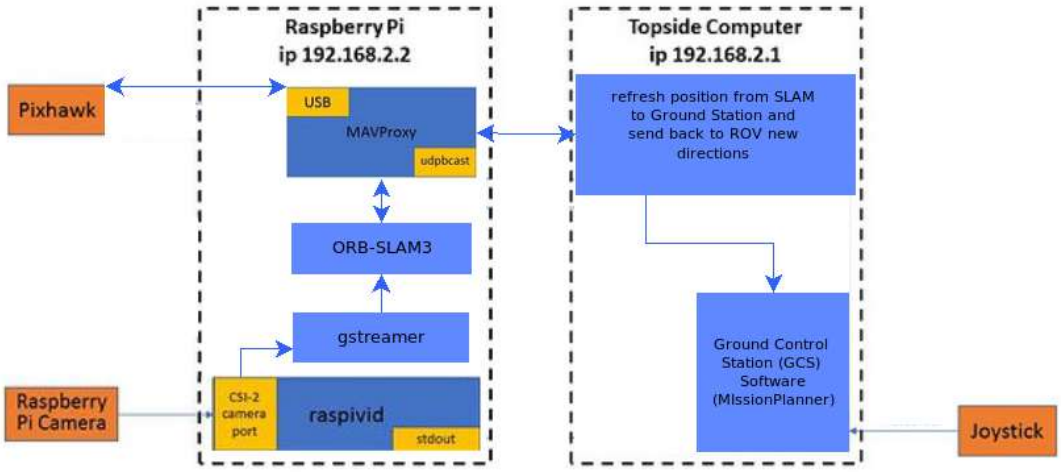

Figure 4.2: Communication diagram of the new approach

4.3.2 Hardware Upgrades

Embedded Systems

The implementation of the proposed architecture was not feasible using the Raspberry Pi 3 (which came pre-installed on ROV) as the hardware on which the algorithm would run. Therefore, we needed to choose a microcomputer that met our requirements both in terms of performance and consumption. Because of these factors, the BlueROV2's main board (the Raspberry Pi 3) was replaced with the Raspberry Pi 4 (8 GB).

Figure 4.3: Raspberry Pi 4, source: wikipedia

The Raspberry Pi 4 is a single-board computer that was introduced in 2019 as the fourth generation of the Raspberry Pi series. It is a low-cost and high-performance device designed to be versatile and suitable for a wide range of applications. One of the key features of the Raspberry Pi 4 is its powerful Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz processor, which makes it much faster and more capable than its predecessors. It also comes with up to 8GB of LPDDR4 RAM, which makes it more suitable for demanding applications that require more memory. In addition, the Raspberry Pi 4 has dual-band 802.11ac wireless and Bluetooth 5.0 connectivity, which allows it to connect to a wide range of devices and networks. It also has two micro-HDMI ports (up

to 4Kp60 supported) for dual monitor support, USB 3.0 and USB 2.0 ports, and a 40-pin GPIO header for connecting to electronics projects. The Raspberry Pi 4 runs on a variety of operating systems, including Raspberry Pi OS, Ubuntu, and more. In this project Raspberry Pi 4 runs on Ubuntu 18.04 server edition (stable version for ORB-SLAM3).

Stereo Camera (Intel Realsense T265)

The Intel RealSense T265 is a device equipped with a fisheye stereo camera and an IMU (Inertial Measurement Unit) that provides real-time positional tracking for use in augmented and virtual reality applications. The device uses computer vision algorithms to track the movements of the camera and to provide accurate and low-latency 6DoF (six degrees of freedom) position and orientation information. The T265 is designed to work as a standalone device or in conjunction with other sensors to provide even more accurate tracking information. Intel's Visual SLAM algorithm runs directly on the T265. However, its use was limited only as a stereo camera in our implementation, because real-time positional tracking was implemented via ORB-SLAM3. The use of this camera over the factory camera was aimed at better SLAM performance, since it utilizes the stereo-inertial capability of the ORB-SLAM3.

Figure 4.4: T265 tracking camera on the left (source: Intel RealSense) and a frame of the T265 on the right (source: ROS)

4.4 Calibration

To run the algorithm, ORB-SLAM3 looks for the intrinsic parameters (f_x, f_y, c_x, and c_y) and the distortion coefficients of the camera, as well as the IMU noise parameters. The coefficient parameters [14],[15], model the radial and tangential lens distortion to represent a real camera. An ideal pinhole camera, such as the one described in Chapter 2, does not have a lens, and for this reason, only the camera's intrinsic parameters were analyzed. The IMU noise parameters, model the sensor's error.

4.4.1 Distortion Coefficients

Radial Distortion

When light rays bend differently near a lens's edges and in its optical center, the result is radial distortion. The distortion increases with lens size. The radial distortion coefficients model is described by the following equations:

$$x_r = x(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6)$$
$$y_r = y(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6)$$

x_r,y_r : distorted points

x,y : undistorted pixel locations

k_1, k_2, and k_3 : radial distortion coefficients of the lens

r : $r^2 = x^2 + y^2$

Tangential Distortion

When the lens and the image plane are not parallel, tangential distortion happens. This kind of distortion is modeled by the tangential distortion coefficients. The model's equations are shown below:

$$x_t = x + [2 \cdot p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2 \cdot x^2)$$
$$y_t = y + [p_1 \cdot (r^2 + 2 \cdot y^2) + 2 \cdot p_2 \cdot x \cdot y]$$

x_t,y_t : distorted points

x,y : undistorted pixel locations

p_1 and p_2 : tangential distortion coefficients of the lens

r : $r^2 = x^2 + y^2$

4.4.2 IMU noise parameters

Briefly, the noise parameters of the IMU are as follows: accelerometer noise density, accelerometer random walk, gyroscope noise density and gyroscope random walk.

Noise Density

The noise density is equal to the noise divided by the sampling rate's square root. For instance, the noise density for an accelerometer can be expressed as $m/s^2/\sqrt{hz}$ and that for a gyroscope as $rad/s/\sqrt{hz}$.

Random Walk

When a noisy sensor output signal is integrated, such as when an angular rate signal is integrated to derive an angle, the integration will drift over time as a result of the noise. Since it will seem as though the integration is moving at random from one sample to the next, this drift is known as a random walk. Angle random walk, which applies to gyroscopes, and accelaration random walk, which applies to accelerometers, are the two primary varieties of random walk for inertial sensors. For gyroscopes and accelerometers, the specification for random walks is commonly stated in units of $rad/s^2/\sqrt{hz}$ and $m/s^3/\sqrt{hz}$, respectively.

4.4.3 Software to calculate parameters

The parameters needed for the algorithm were calculated with the help of allan_variance_ros and Kalibr.

Allan_variance_ros

The Allan variance ROS package is an implementation of the Allan variance algorithm for ROS. It is designed to work with ROS sensor messages and provides a way to evaluate the stability of the IMU measurements. The package computes the Allan deviation and Allan variance for different time intervals and provides plots to visualize the results. To use the Allan variance ROS package for the calibration of the IMU, the IMU data needs to be published on a ROS topic. For our calibration, a 3-hour rosbag is recorded, during which the IMU was at rest. The package then subscribes to this topic, computes the Allan deviation and variance, and extracts the parameters as shown in figure 4.5.

```
#Accelerometer
accelerometer_noise_density: 0.004494919864148074
accelerometer_random_walk: 0.00039302293390133333

#Gyroscope
gyroscope_noise_density: 0.0004871158387229554
gyroscope_random_walk: 4.920256144620228e-06

rostopic: 'gx5/imu/data'
update_rate: 30.0
```

Figure 4.5: Calibration file of ROV's IMU from allan_variance_ros

Kalibr

The Kalibr camera calibration toolbox [16] is a powerful open-source software package designed for calibrating cameras and other sensors in robotics and computer vision applications. Camera calibration is a crucial step in many computer vision and robotics applications, as it enables accurate measurements of the 3D world from 2D images. The Kalibr toolbox provides a suite of tools for calibrating cameras and other sensors, including monocular, stereo, and RGB-D cameras, as well as IMUs and LIDARs.

4.4.4 Description of the calibration process

For the process of camera calibration, we used the aprilgrid, which is shown in Figure 4.16.

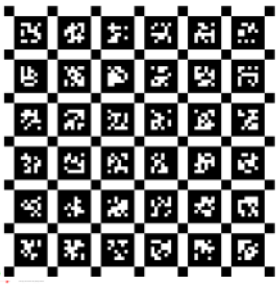

Figure 4.6: Aprilgrid, source: OpenCV

The Aprilgrid was placed on a glass, and then a record was made on a rosbag of the camera and IMU topics, and we recorded the movement of the ROV in front of the Aprilgrid in all axes, with the Aprilgrid however staying in the field of view of the camera.

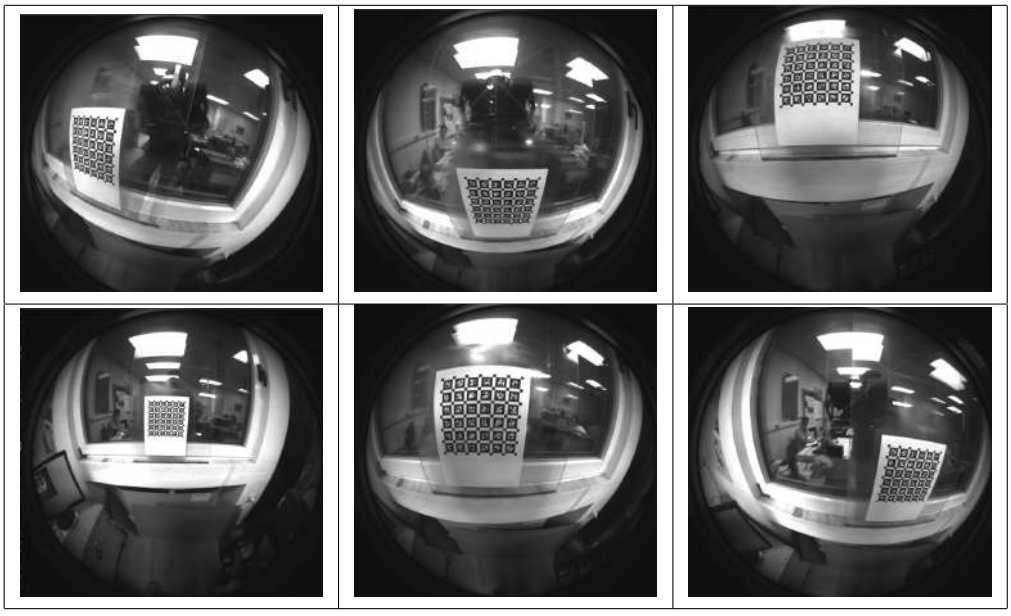

Figure 4.7: Screenshots from the time of the rosbag recording

Then the .bag file and the file from the Allan_variance_ros package is imported into Kalibr and we have the final file from calibration. This file gives information both for the parameters mentioned above and for the transformations in the case

of the stereo camera from the coordinate system of the right to the left camera and for the coordinate system of the IMU with respect to the left camera.

4.5 Ground Truth

In order to be able to compare the results from ORB-SLAM3 with real-world distances, a 3D representation of the interior of the Senselab was created with the help of a 3D scanner. As shown in Figure 4.8, 15 points have been used, which are marked with a red dot on the floor and walls.

Figure 4.8: Points which were used in the 3d scanner

In order to be able to scan the whole space inside with more detail, we placed the 3D scanner in two places. So, what we see in Figures 4.9 and 4.10 is the result of merging the two different 3D images we took. The blue circles in Figures 4.9 and 4.10 are the points where we placed the 3D scanner, and the green line that appears in these figures is the imaginary line that connects these two points that we chose.

Figure 4.9: 3D-scanner's result

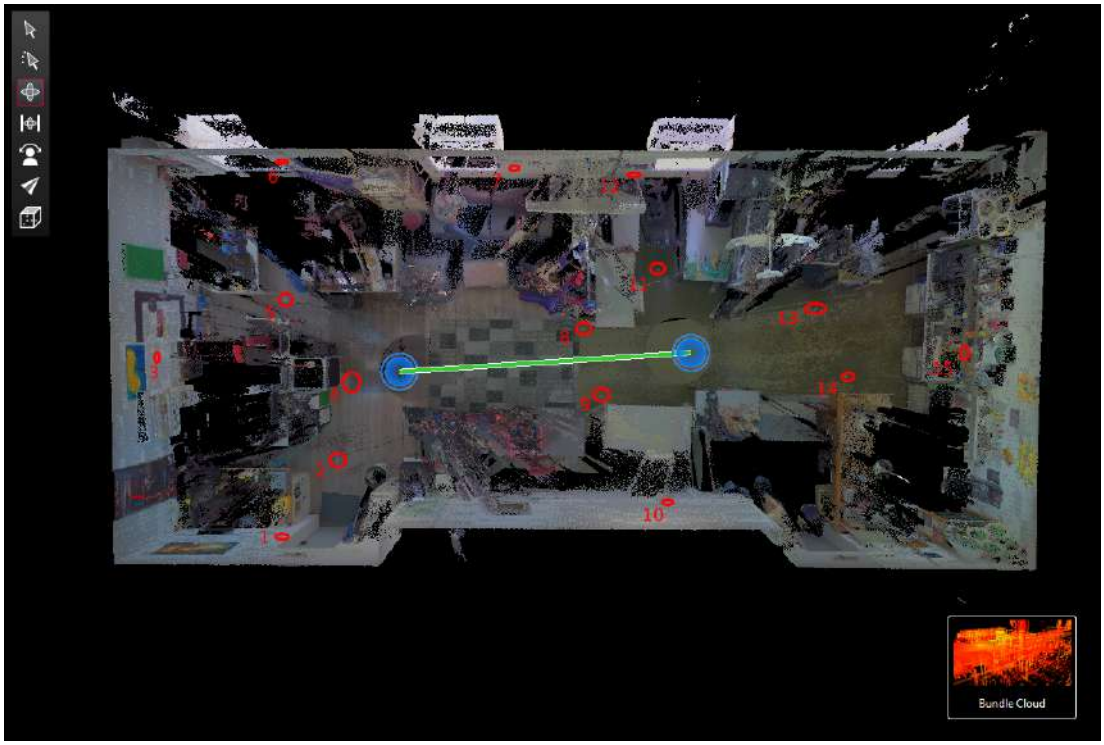

Figure 4.10: 3D-scanner's result with the points we choose

On the 3D visualization, we found these points; they are the points shown circled in Figure 4.10, and then we took the distances between them and compared them with the actual ones. The deviation was of the order of $\pm 0.01m$.

These points are then plotted in a 3D plot (Figure 4.11). We then removed from the 3D plot the points referring to the dots on the walls, leaving only the points referring to the dots on the floor. We created a path between the points that remained, which was then the path we used as ground truth to compare with the results of ORB-SLAM3. The path is shown in Figure 4.12. The results of ORB-SLAM3 are shown and analyzed in Section 5.2.

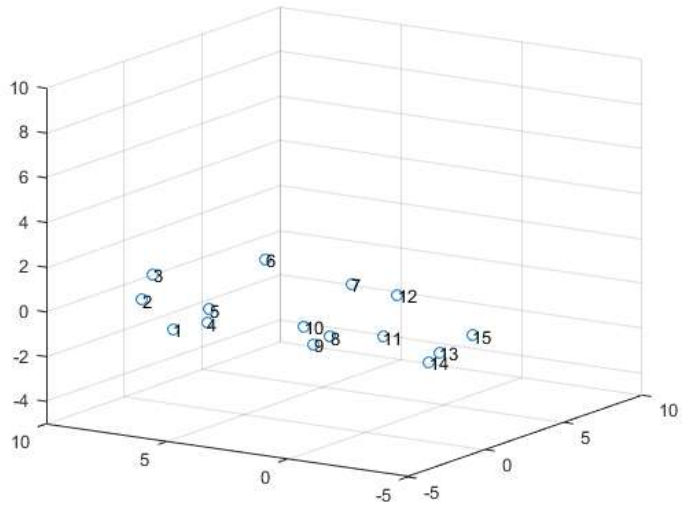

Figure 4.11: Points in 3D-plot

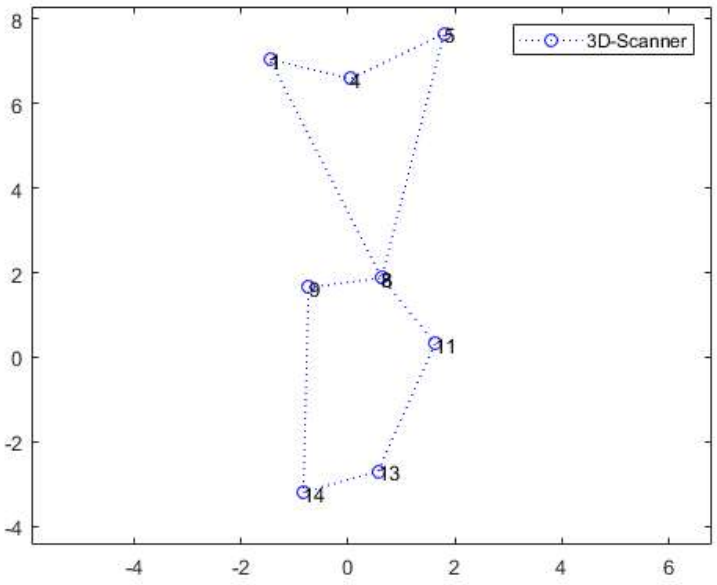

Figure 4.12: Path between the points referring to the dots on the floor

4.6 Simulation

In order to implement the autonomous movement of the ROV, a virtual environment was created in order to test it safely, before the final tests in real conditions. The simulation required several simulator programs for both the physical system and communication over the MavLink protocol with MAVROS and the virtual 3D environment and hydrodynamic model.

4.6.1 Github repositories

There are many repositories out there that implement scripts for drones, rovers, and underwater vehicles for the Gazebo Simulator. *bluerov_ros_playground* and *uuv_simulator* are these were used to create a virtual environment for gazebo simulator to test ORB-SLAM3 and autonomous driving of BlueROV2.

bluerov_ros_playground

bluerov_ros_playground [17] consist of scripts to help BlueROV2 integration with ROS. It provides video streaming capture with OpenCV, read and write over mavlink protocol with MAVROS, Joystick interaction, BlueROV2 model and Gazebo simulation.

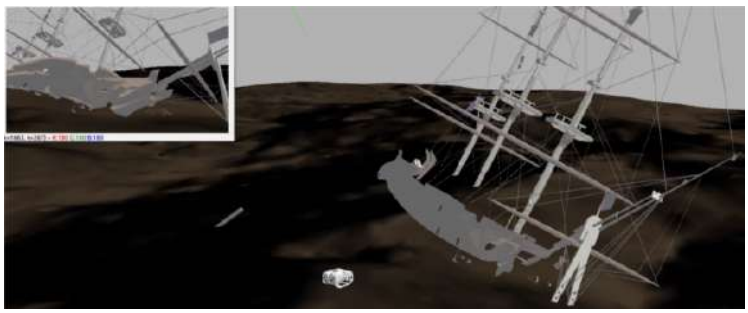

Figure 4.13: Underwater map from bluerov_ros_playground, source: Github BlueRov-ROS-playground

UUV Simulator

The Gazebo plugins and ROS nodes included in the UUV Simulator package [18] enable the simulation of unmanned underwater vehicles like ROVs and AUVs. The UUV Simulator provides many examples of underwater maps. It was exploited to create a new map based on implemented maps of the repository and test ORB-SLAM3 in different light and fog conditions.

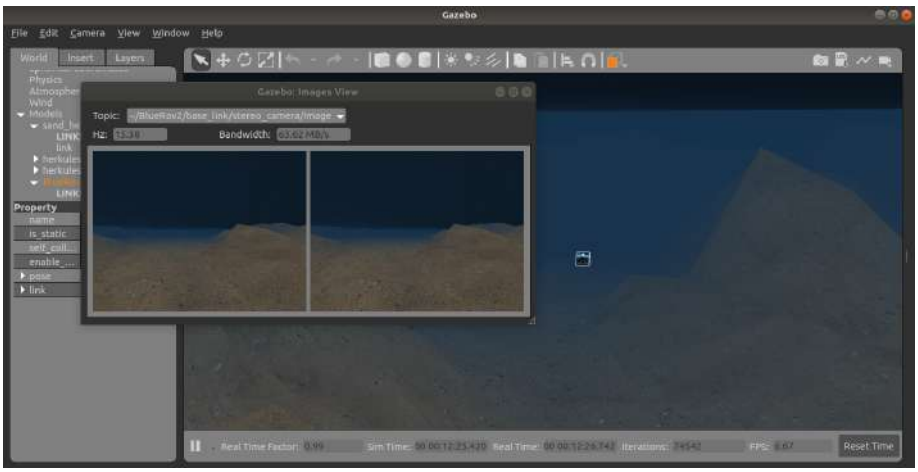

Figure 4.14: Stereo camera on BlueROV2

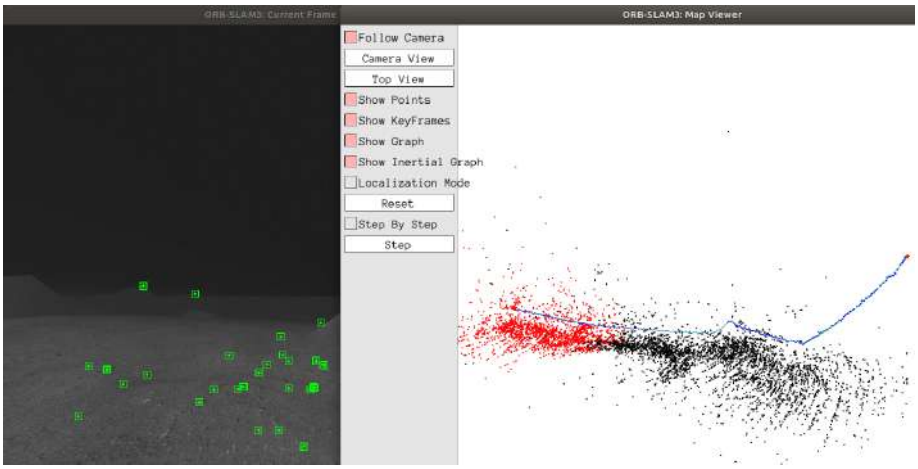

Figure 4.15: ORB-SLAM3 running in the map we created

4.7 Flow Diagrams

The flow diagram in Figure 4.16 describes the process by which, using the camera and IMU data, we get to give motion to the ROV. The arrows indicate the topics that each node publishes at or subscribes to (publishing is done from the node where the arrow starts, while subscribing is done from the node where the arrow reaches). ORB-SLAM3 uses the camera and IMU data and outputs an Odomentry message. Then, from this message, we keep the position and orientation and update the position of the ROV in MissionPlanner, while inserting the information in the code responsible for the ROV movement by publishing it in topic */mavros/vision_pose/pose*. Finally, after calculating the distance from the target and the angle of rotation that the vehicle will perform, we send a RC_OVERRIDE message back to the ROV to start its movement.

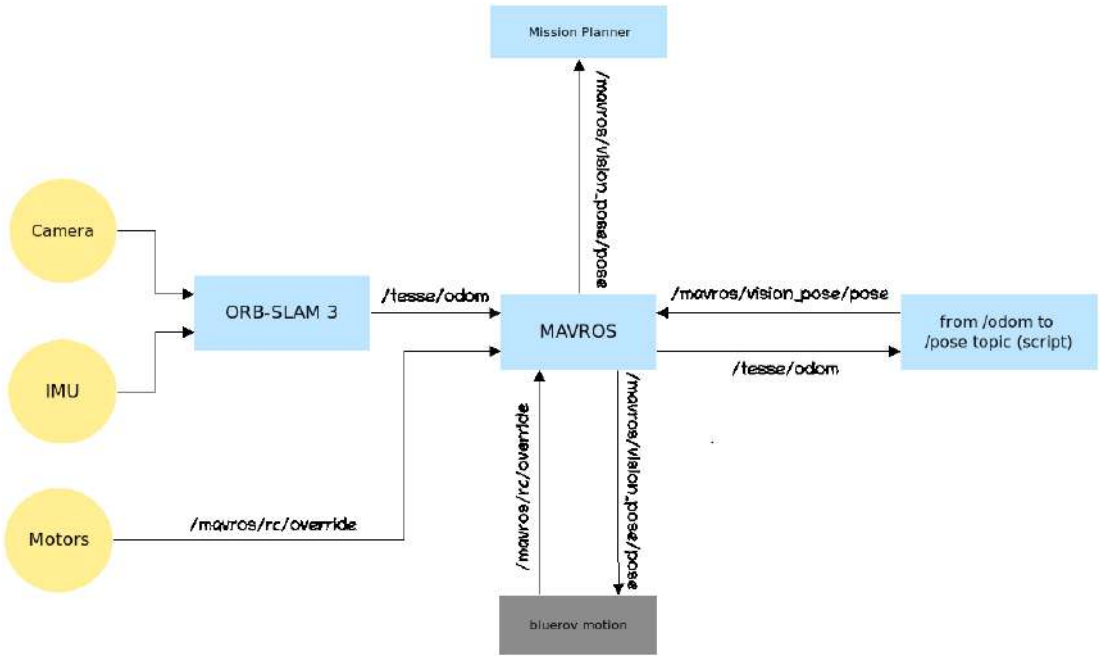

Figure 4.16: From camera and IMU data to ROV's motion

The ROV motion class is divided into four sub-classes: OnePoint, Multiple Points, Spiral and Rectangle as shown in Figure 4.17.

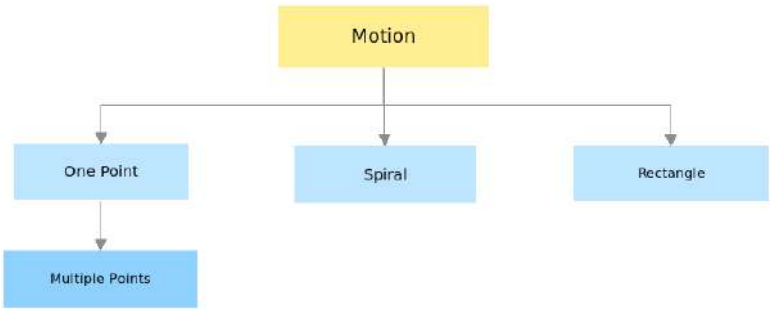

Figure 4.17: class diagram of ROV's motion algorithm

In the OnePoint and MultiplePoints classes, the user enters the points in coordinates (x,y,z) that he wants the ROV to follow. In the case of OnePoint we enter only one point, while in the other case, the user can enter a large number of points. In the case of Rectangle we enter the length and width, and then the points are calculated based on the position where the ROV is at that moment and starts to execute the path, until it completes the movement on the perimeter of the rectangle once. In all three classes the vehicle corrects the orientation and depth to the target and then starts moving along the target. During movement, it constantly corrects both the orientation and the depth at which it is located. In the case of the spiral trajectory, the vehicle does not stop moving until the user presses the corresponding exit button. In the spiral trajectory, the user can set the maximum and minimum depths the ROV will reach, as well as the velocity in the yaw, x, and z axes. The speed during the movement is constant on both the yaw axis and the x axis. On the z axis, the velocity magnitude remains constant, but changes direction depending on whether the ROV has crossed the limits of maximum and minimum depth.

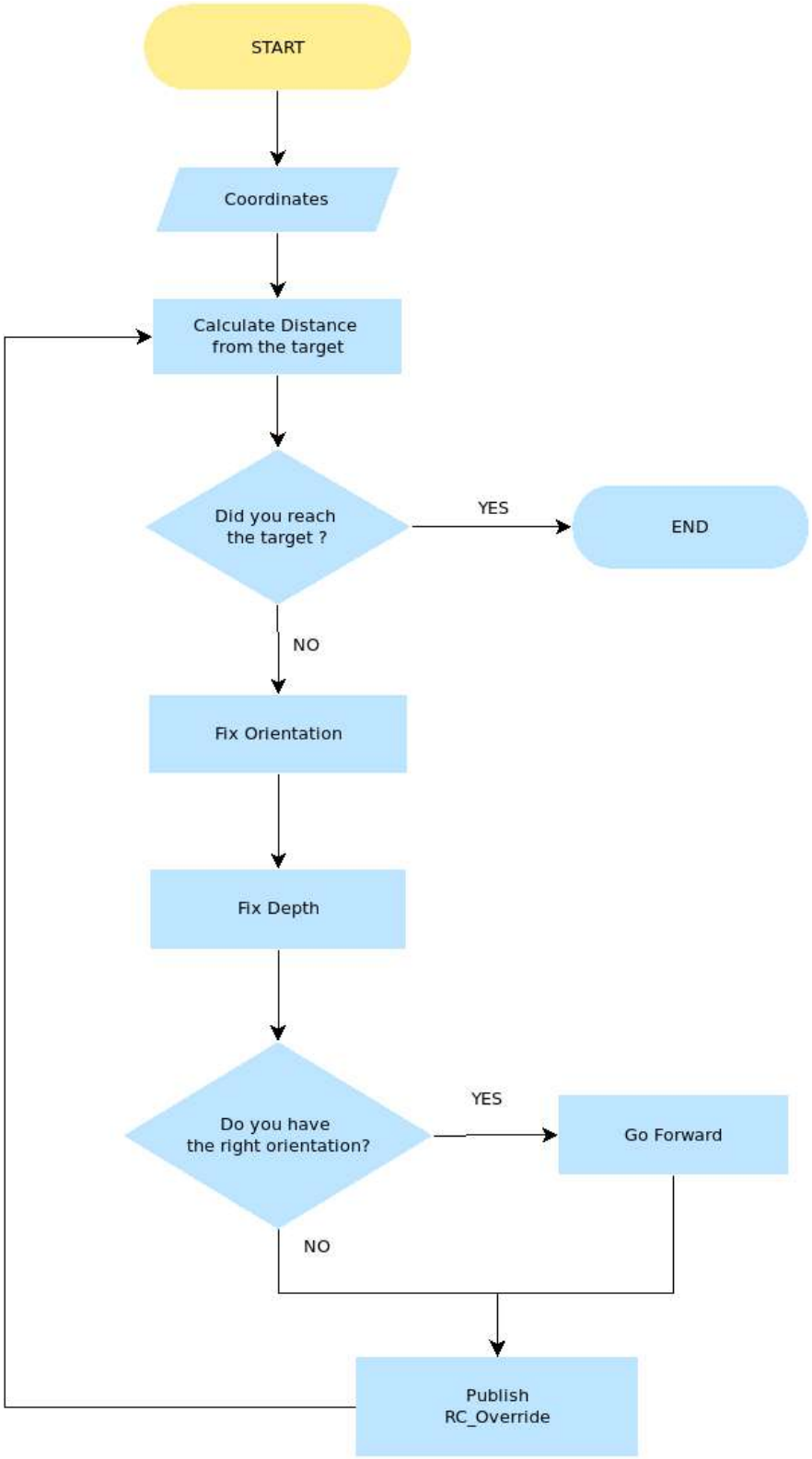

Figure 4.18: Flowchart diagram in the case of straight line motion

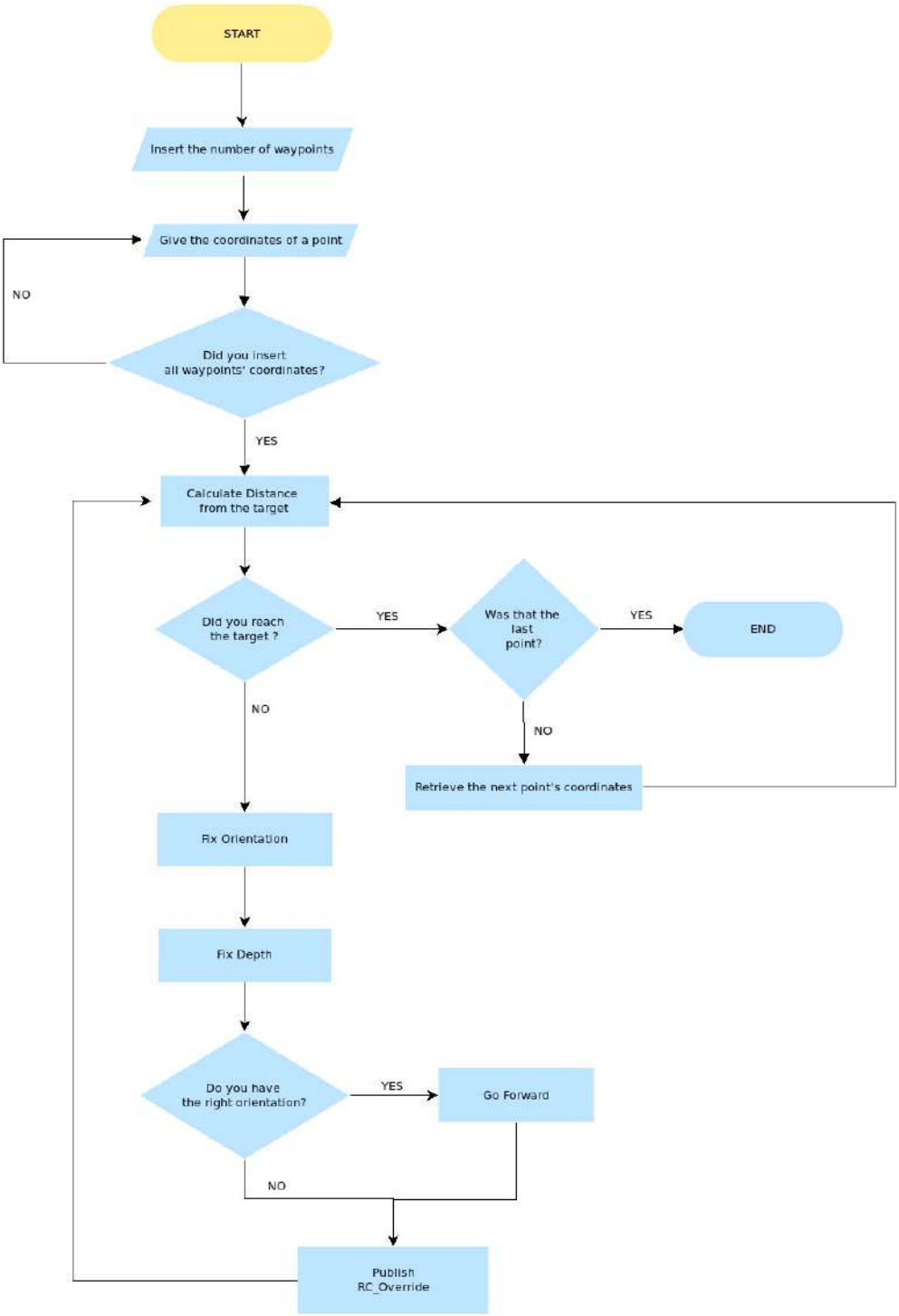

Figure 4.19: Flowchart diagram in the case of motion between multiple points

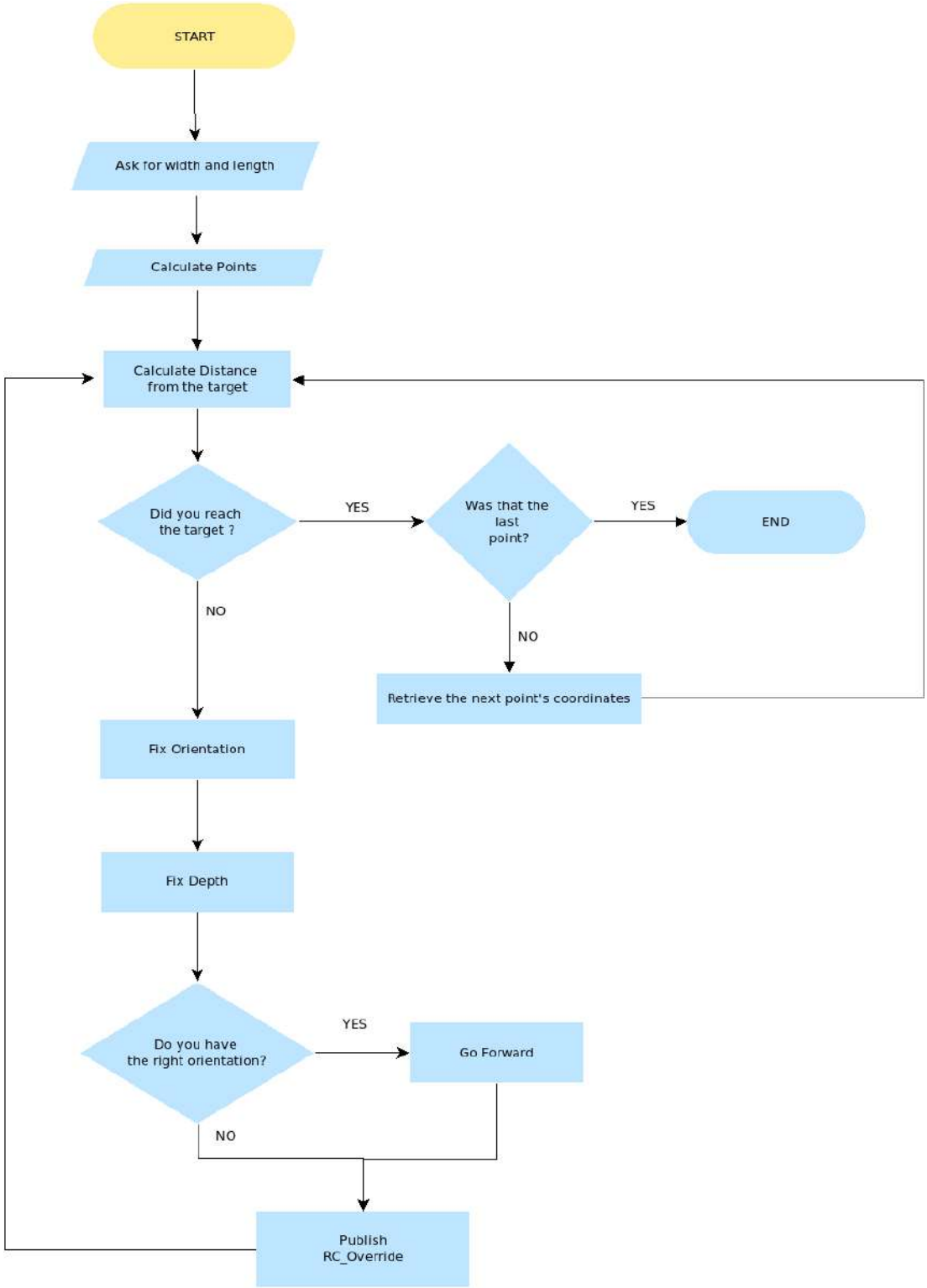

Figure 4.20: Flowchart diagram in the case of motion in a rectangle

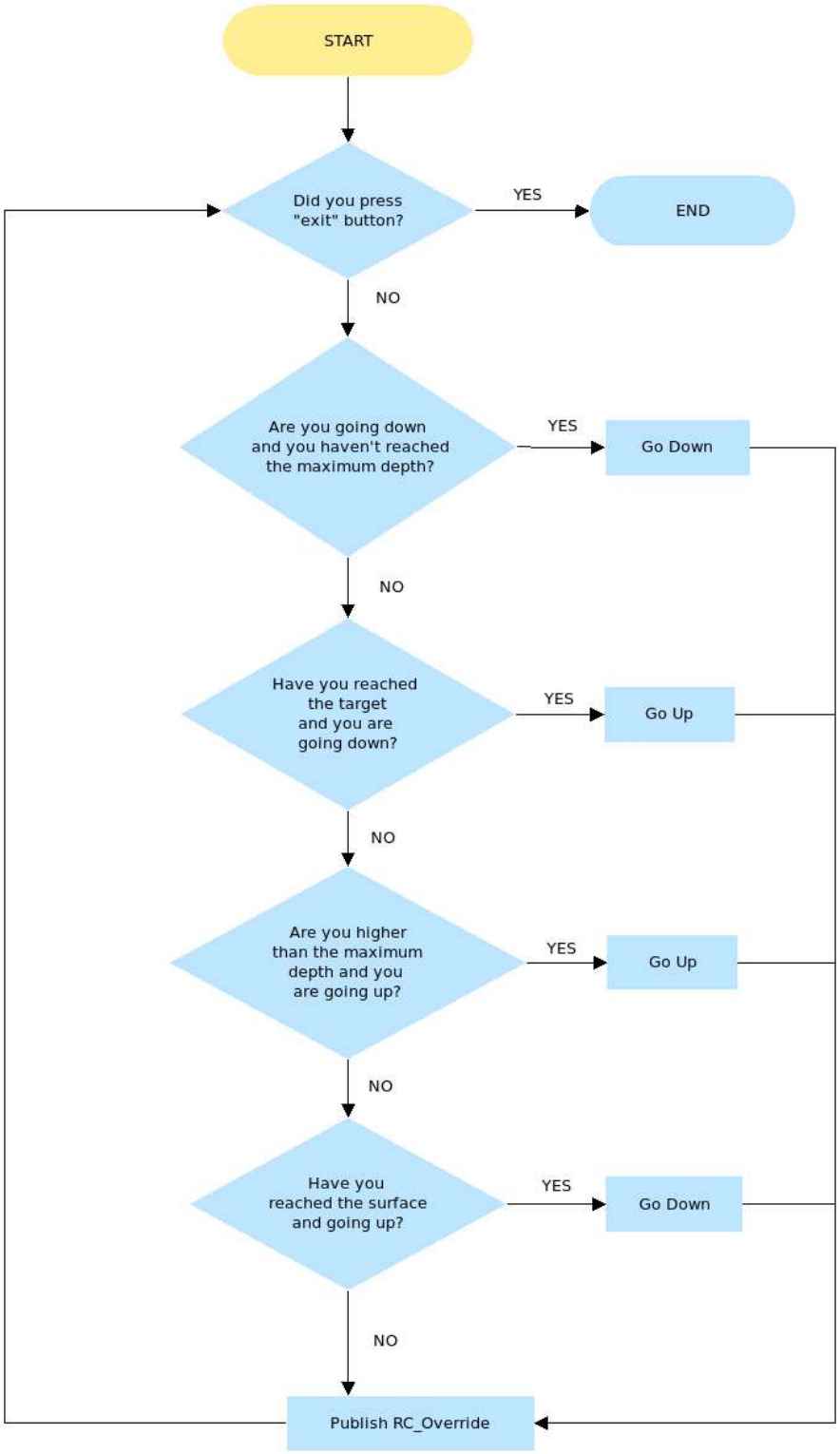

Figure 4.21: Flowchart diagram in the case of spiral trajectory

Chapter 5

Results

This chapter will present the results of the experiments, which are divided into two sections. The first section deals with the results of the simulation. In the following, the results of ORB-SLAM3 are taken from inside SenseLab.

5.1 Simulation Results

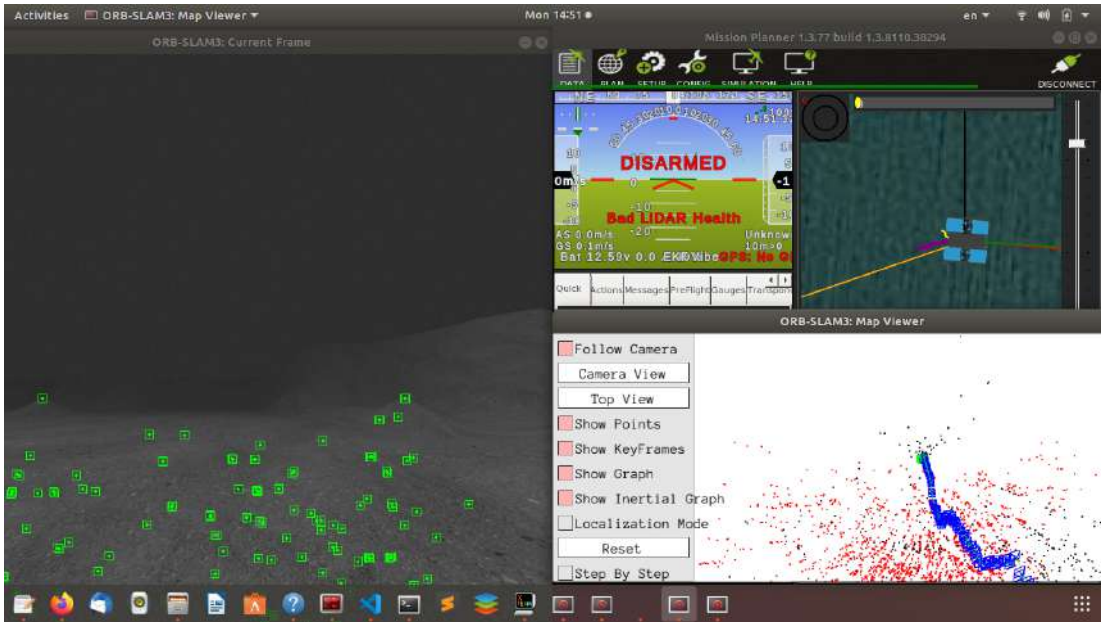

Figure 5.1: The picture above shows, on the left, the image from the camera with the ORB features of ORB-SLAM3. On the bottom right is the path followed according to the SLAM, and on the top right is the tracing of the path through the mission planner.

In Figure 5.1, we can see that, as the algorithm runs, the position is updated in mapViewer, and with the help of the script, which was described in the previous Chapter, it is also updated in the Mission Planner. In the Mission Planner, we

define the geographic coordinates of the ROV, so by giving the coordinates of the point where it is located, we know in real time the exact position of the vehicle on earth.

In the simulation process, four different motions were followed: a) motion in a straight line; b) motion in multiple points; c) motion in a rectangle; and d) motion in a spiral trajectory. For each movement, the result through MapViewer is given and then a plot with the movement of the ROV recorded by ORB-SLAM3 in a common plot with the ground truth of the path.

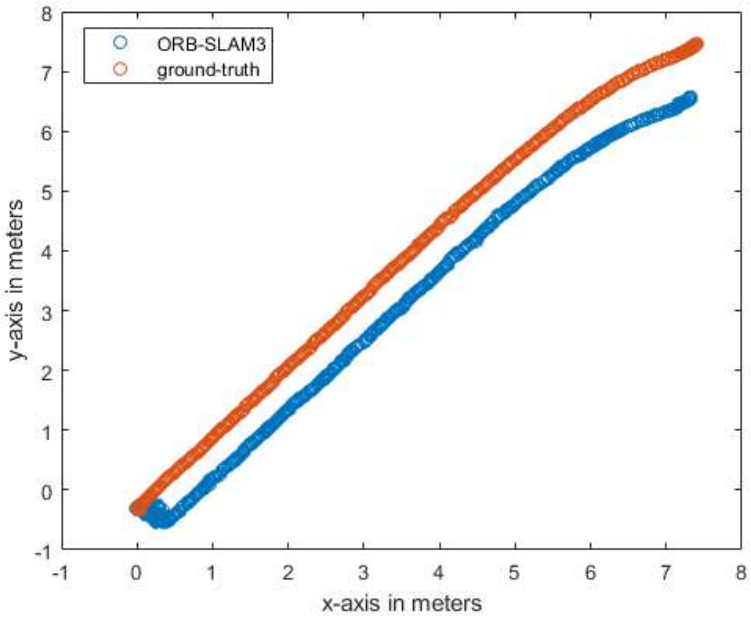

Figure 5.2: Ground truth and ORB-SLAM's path in common plot in the case of straight line motion.

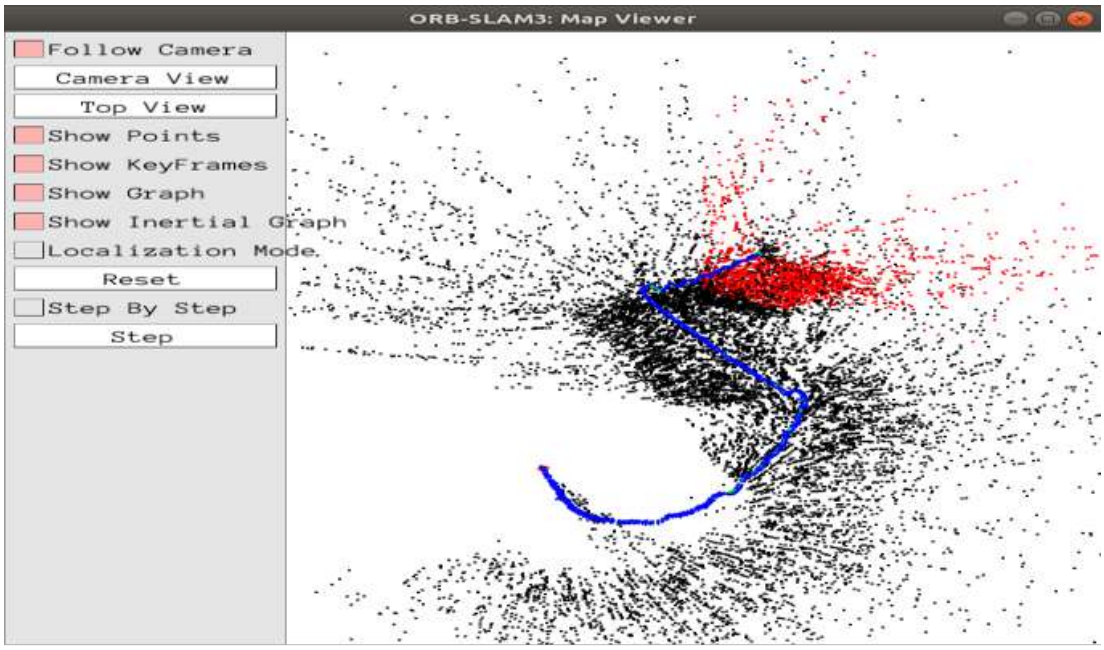

Figure 5.3: ORB-SLAM's path as recorded in MapViewer in the case of motion between waypoints.

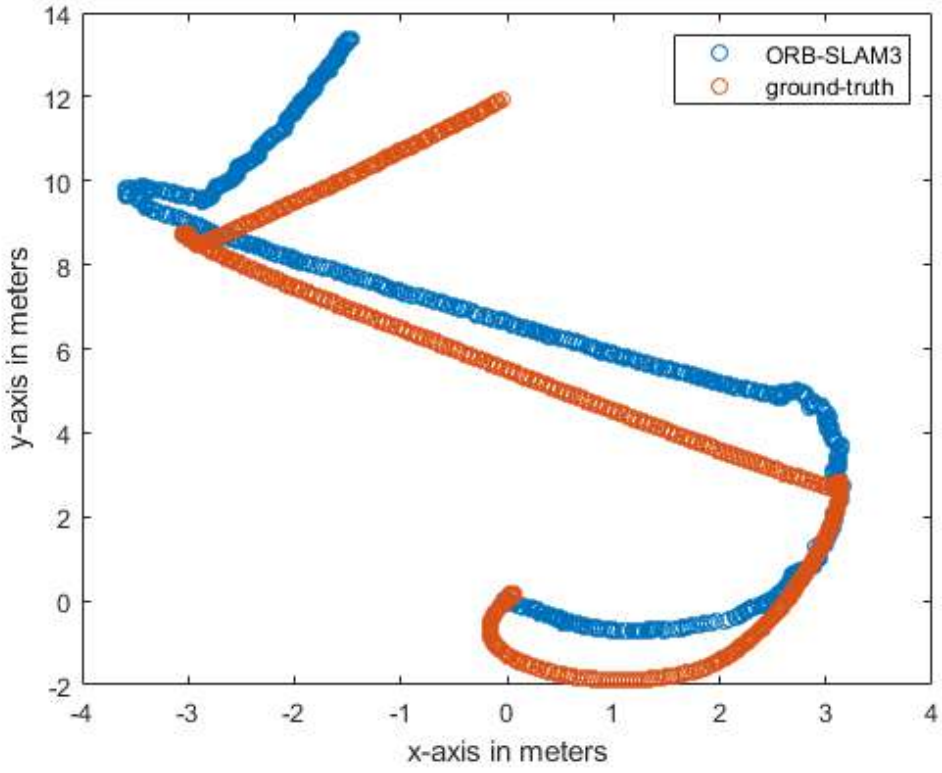

Figure 5.4: Ground truth and ORB-SLAM's path in common plot in the case of motion between waypoints.

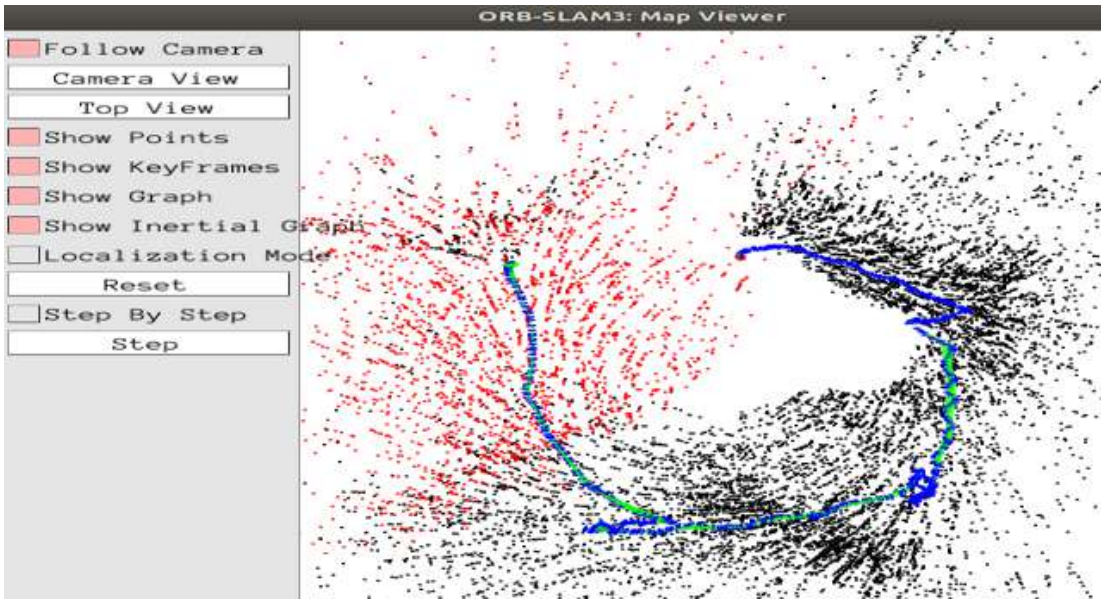

Figure 5.5: ORB-SLAM's path as recorded in MapViewer in the case of motion in a rectangle.

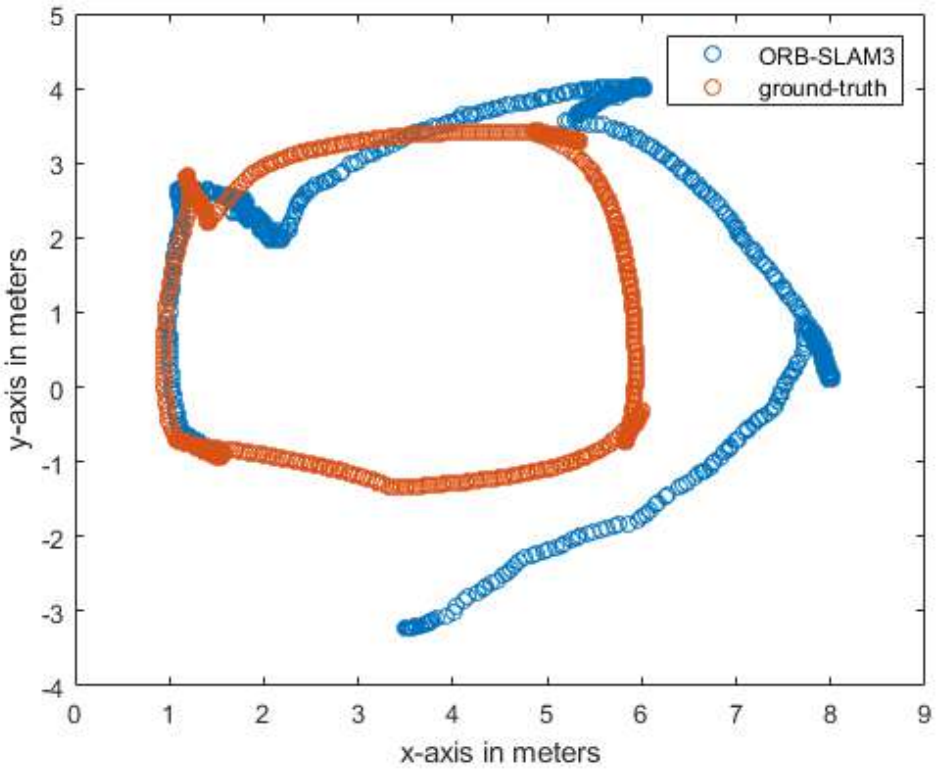

Figure 5.6: Ground truth and ORB-SLAM's path in common plot in the case of motion in a rectangle.

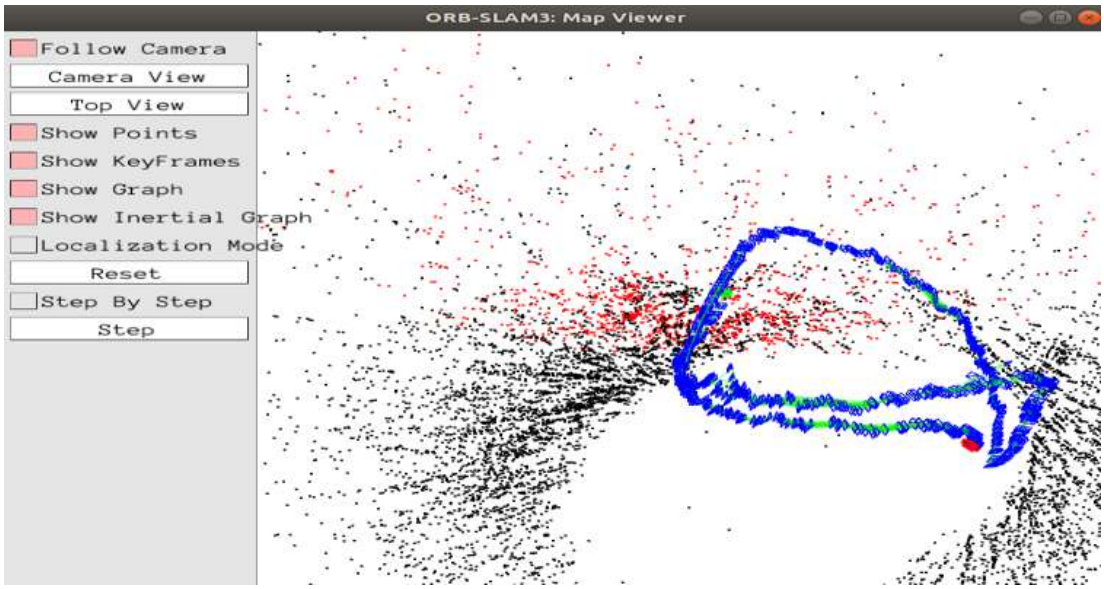

Figure 5.7: ORB-SLAM's path as recorded in MapViewer in the case of spiral trajectory (side view).

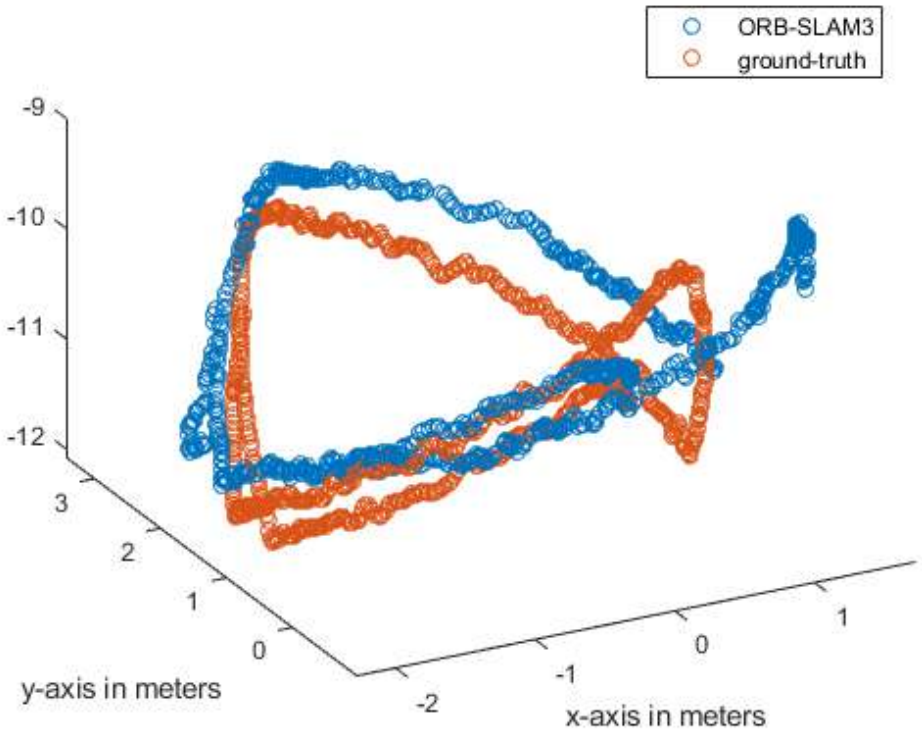

Figure 5.8: Ground truth and ORB-SLAM's path in common plot in the case of spiral trajectory (side view).

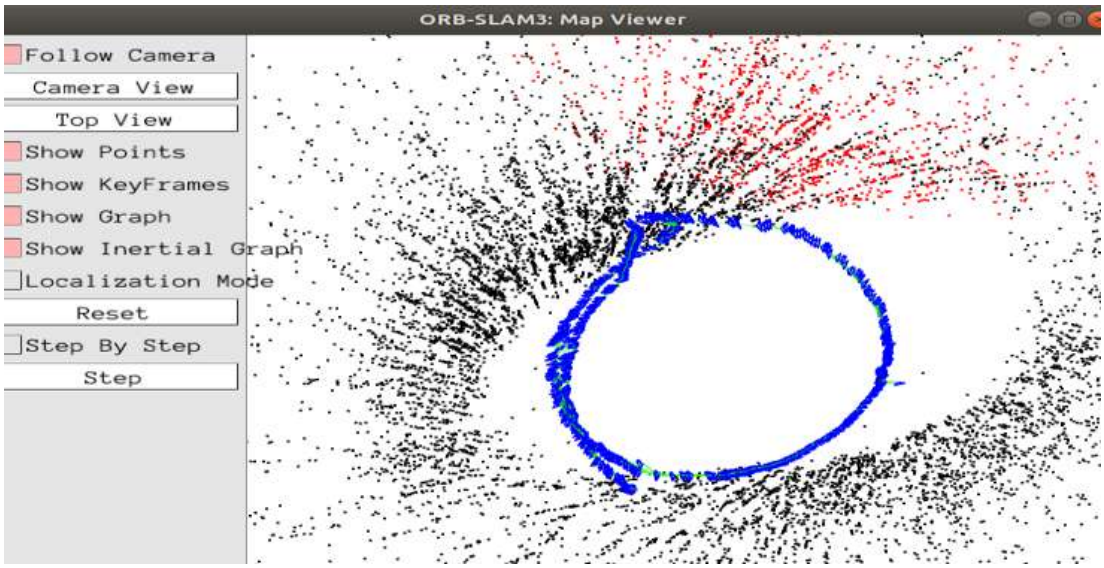

Figure 5.9: ORB-SLAM's path as recorded in MapViewer in the case of spiral trajectory (plan view).

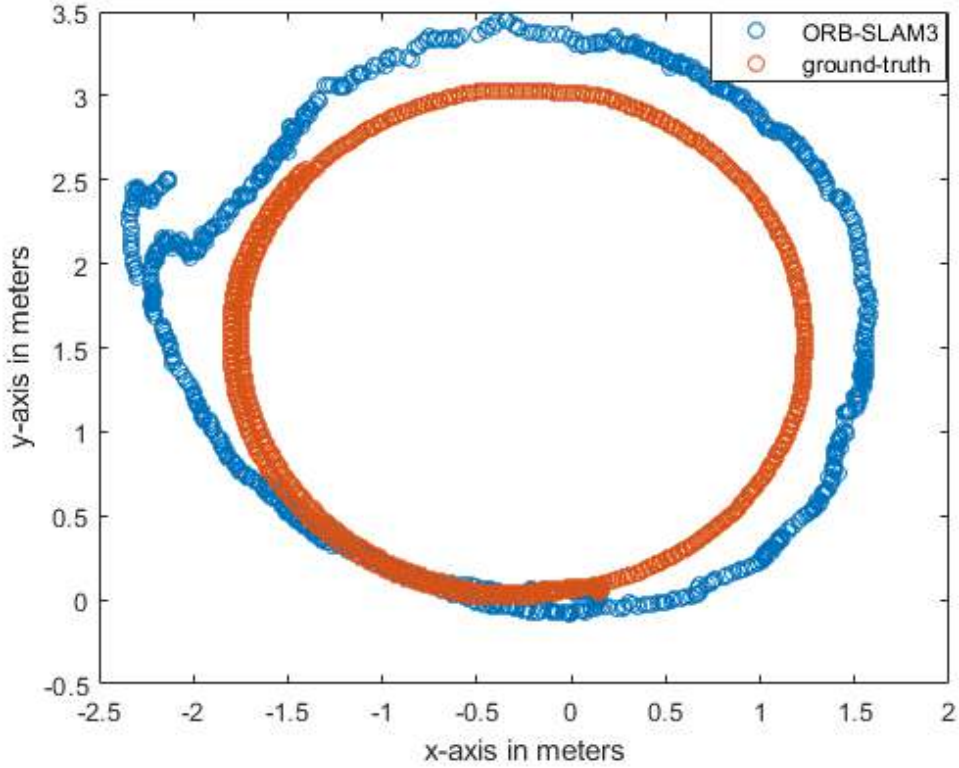

Figure 5.10: Ground truth and ORB-SLAM's path in common plot in the case of spiral trajectory (plan view).

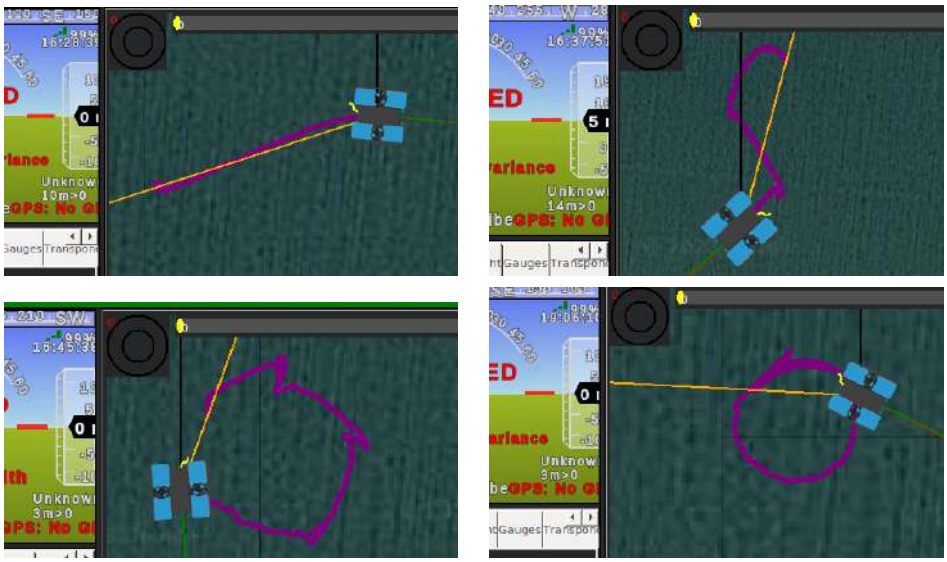

Figure 5.11: The results of the different styles of ROV's motion in MissionPlanner

Motion Cases	Mean squared error (MSE) (m^2)
Straight line	0.2552
Waypoints	1.9412
Rectangle	2.0149
Circle	0.3633
Spiral	0.2679

Table 5.1: Error estimation for every path was tested

From the results of the above table, we see that the algorithm has much less error in the cases where it performs motion in a straight line and in the spiral trajectory (the case of the circular trajectory reported in the table is derived from the spiral setting the minimum depth equal to the maximum). In the other two cases, the effects of the error are much larger as the error is additive, as can be seen in the figures with the paths followed by the vehicle. As we saw in the figures, the deviation from ground truth becomes larger as we add points that the ROV has to pass through.

5.2 ORB-SLAM3 Indoor Results

In this section, the experiments were performed on the path from the SenseLab interior shown in Chapter 4. In addition to comparing the results of ORB-SLAM3 with the ground truth in a common plot, we also have the results from the tracking of RealSense T265.

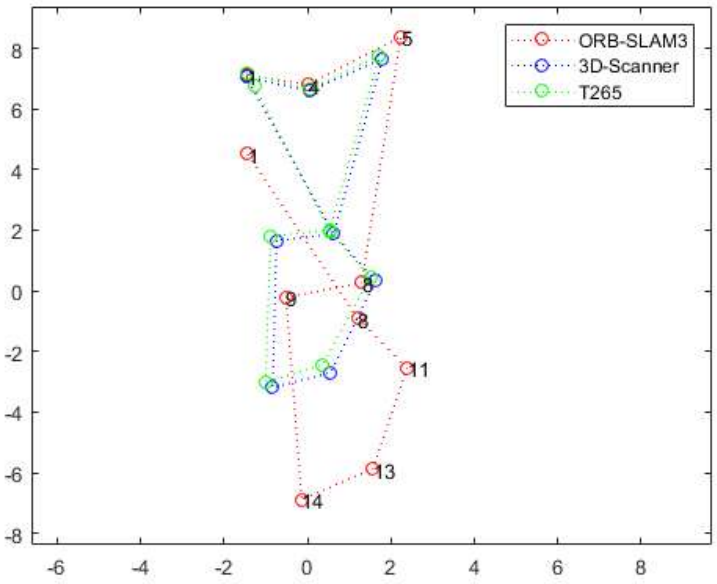

Figure 5.12: Ground truth, ORB-SLAM's path and slam from t265 in common plot

In Figure 5.12, we compared eight points on the path followed during the experiments. However, the path we followed in this experiment does not go from point to point in the graph above by moving along the straight line segment joining two adjacent points, but follows a more complex path. These eight points are shown in FIgure 5.13, on the path recorded by the SLAM algorithm.

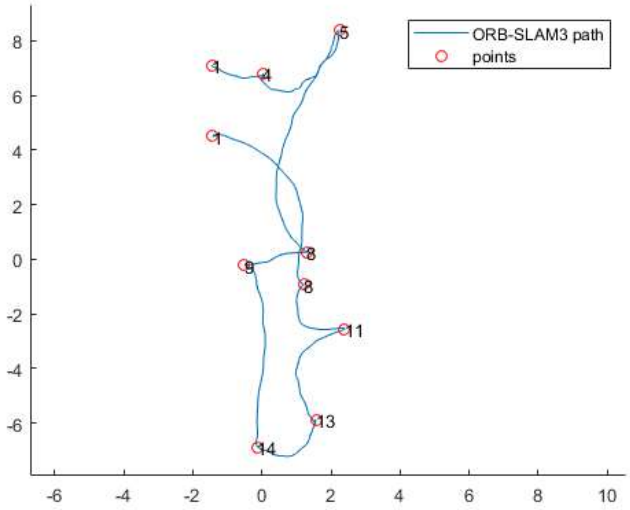

Figure 5.13: ORB-SLAM's points from Figure 5.12 with the entire path followed.

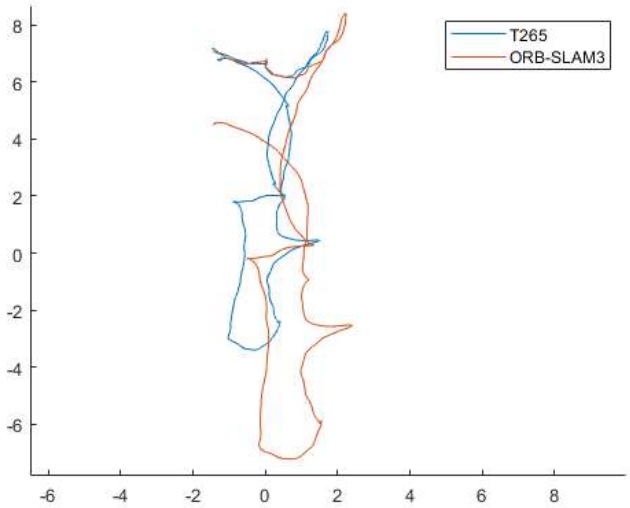

Figure 5.14: The entire path of the ORB-SLAM3 in comparison with the result of T265 tracking camera's slam.

SLAM	Mean squared error (MSE)
ORB-SLAM3	2.8031
T265	0.0210

Table 5.2: Error estimation for ORB-SLAM3 and T265 tracking camera

In the experiment inside the lab, the ORB-SLAM3 shows similar results to the previous case, where we had over a point where the ROV would pass. Compared to the result of the T265 acting as a tracking camera, the MSE error from RealSense T265 is much lower compared to the ORB-SLAM3 using the T265 as a stereo camera. A separate waterproof case was required, because the T265 camera could not fit into the existing ROV waterproof casing.

5.3 Waterproof case construction attempts

To get results from an underwater environment, several attempts were made to build a waterproof case for the T265 stereo camera, as in a test with the factory camera the ROV originally had, the ORB-SLAM3 failed to run underwater.

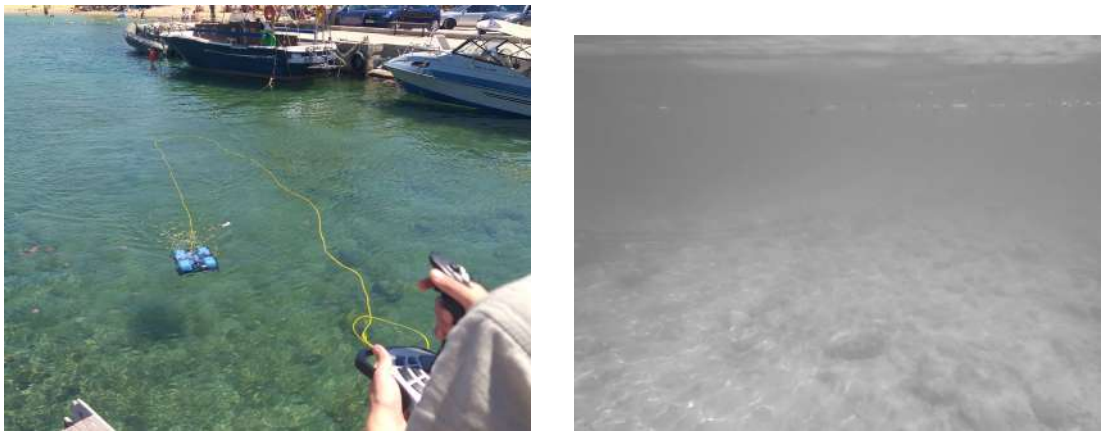

Figure 5.15: On the left we see BlueROV2's first test in the sea and on the right we see that ORB-SLAM3 cannot extract any features with the factory camera in this experiment.

First, we designed a 3D-printed mold for the case, placed the threads, and finally filled the inside of the mold with resin. Unfortunately, during the peeling process, the resin broke. We chose a more elastic material for the mold; however, we had problems during printing, and this implementation was not completed.

In the last attempt, we used a resin printer for printing, where we would not print the case mold first, but directly the case itself. Therefore, we redesigned the case design and started printing. Even in this print, there were imperfections that prevented the camera from being waterproofed.

Figure 5.16: The 3D-printed mold for the case on the left, and the mold filled with resin on the right

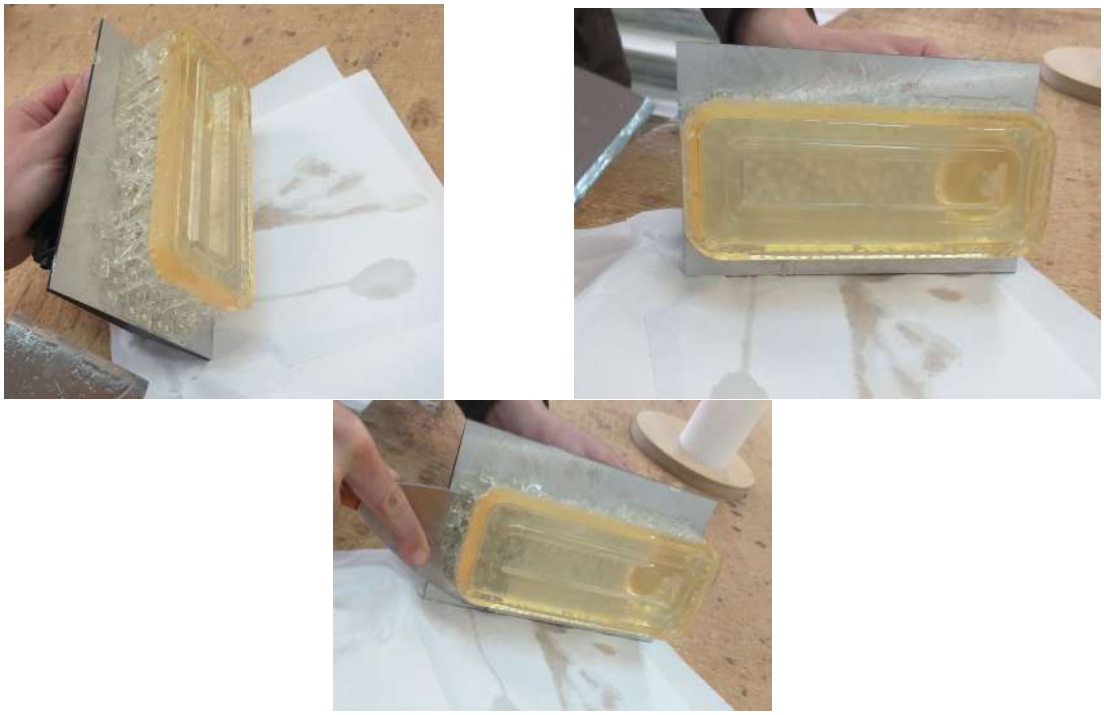

Figure 5.17: Waterproof case just after the printing

Therefore, the task of placing the T265 camera in a safe, waterproof case is still open for full deployment of the modified BlueROV2 underwater.

Figure 5.18: Waterproof case inside UV Curing Machine

Figure 5.19: Waterproof case just after the Curing Machine

Chapter 6

Conclusion

6.1 Summary

The main objectives of this thesis were two. The first was to construct an algorithm that is responsible for navigating autonomously the vehicle based on information about its position and the locations of the points it has to pass through.The second part concerned the localization and mapping of the environment to enable the underwater vehicle to move in this unknown environment. The localization and mapping process was vision-based using the open-source algorithm ORB-SLAM3. Since we are talking about autonomous navigation, this means that the user does not interfere in the navigation process of the ROV, and the ROV itself can continue its navigation even if it loses communication with the topside computer. This led us to choose an architecture in which the SLAM algorithm can run on the BlueROV2 embedded. This, in turn, led to the need to upgrade the hardware to be able to support our architecture. The SLAM part was tested both in simulation and in a real-world indoor environment, while the autonomous navigation part was only seen in simulation. What is missing is a test in a real-world underwater environment. This test did not take place as we could not solve the problem of waterproofing the stereo camera.

6.2 Future work

The section on autonomous navigation and using SLAM based on vision for underwater scenarios has many open topics that can be read and experimented with. In our own approach, certainly one of the things to be done is testing in real-world underwater environments, where the effects of lack of illumination and distortion are more pronounced. This, of course, requires the appropriate equipment, such as a waterproof stereo camera, which is a problem we need to solve. We could still work on ways in which we could integrate other sensors into the ORB-SLAM3, such as sonar, and, in this way, have more accuracy in our data. In terms of autonomous ROV navigation, the next steps to be taken are to recognize obstacles and avoid them, or people to help rescue them.

REFERENCES

[1] H. Nguyen, Z. Wang, P. Jones, and B. Zhao, "3d shape, deformation, and vibration measurements using infrared kinect sensors and digital image correlation," *Appl. Opt.*, vol. 56, no. 32, pp. 9030–9037, Nov 2017. [Online]. Available: https://opg.optica.org/ao/abstract.cfm?URI=ao-56-32-9030

[2] C. Campos, R. Elvira, J. J. Gomez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[3] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[4] A. Anwar, "What are intrinsic and extrinsic camera parameters in computer vision?" *Towards Data Science*, 2022.

[5] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[6] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): part ii," pp. 108–117, 2006.

[7] G. Taraldsen, T. A. Reinen, and T. Berg, "The underwater gps problem," in *OCEANS 2011 IEEE - Spain*, 2011, pp. 1–8.

[8] H. Elbehiery, "Underwater gps navigation," *Journal of American Science*, vol. Vol. 8, p. 978, 10 2012.

[9] H. Johannsson, M. Kaess, B. Englot, F. Hover, and J. Leonard, "Imaging sonar-aided navigation for autonomous underwater harbor surveillance," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4396–4403.

[10] S. Rahman, A. Quattrini Li, and I. Rekleitis, "Svin2: An underwater slam system using sonar, visual, inertial, and depth sensor," 11 2019, pp. 1861–1868.

[11] E. Vargas, R. Scona, J. S. Willners, T. Luczynski, Y. Cao, S. Wang, and Y. R. Petillot, "Robust underwater visual slam fusing acoustic sensing," in

2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 2140–2146.

[12] Z. Chen, "ORB-SLAM3 ROS wrapper," Sep. 2021. [Online]. Available: https://github.com/zhaozhongch/orbslam3_ros

[13] D. Sharafutdinov, M. Griguletskii, P. Kopanev, M. Kurenkov, G. Ferrer, A. Burkov, A. Gonnochenko, and D. Tsetserukou, "Comparison of modern open-source visual slam approaches," 2023.

[14] J. Steward, "Camera modeling: Exploring distortion and distortion models, part ii," Aug 2021. [Online]. Available: https://www.tangramvision.com/blog/camera-modeling-exploring-distortion-and-distortion-models-part-ii

[15] B. Minor and J. Steward, "Camera modeling: Exploring distortion and distortion models, part iii," Apr 2023. [Online]. Available: https://www.tangramvision.com/blog/camera-modeling-exploring-distortion-and-distortion-models-part-iii

[16] A. S. L. . E. Zürich, "Kalibr: visual-inertial calibration toolbox," Feb. 2023. [Online]. Available: https://github.com/ethz-asl/kalibr

[17] P. J. Pereira, "bluerov-ros-playground," Jan. 2018. [Online]. Available: https://github.com/patrickelectric/bluerov_ros_playground

[18] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "Uuv simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, 2016, pp. 1–8.