Heriot-Watt University

Masters Thesis

Data Fusion for SLAM in Underwater Environment

Author:
Timothee Freville

Supervisor:
Dr. Yvan Petillot

A thesis submitted in fulfilment of the requirements for the degree of MSc.

in the

School of Engineering and Physical Sciences

October 2020

Declaration of Authorship

I, Timothee Freville, declare that this thesis titled, 'Data Fusion for SLAM in Underwater Environment' and the work presented in it is my own. I confirm that this work, submitted for assessment is expressed in my own words. Any use, made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: _____

Date: _____

" In Theory, theory and practice are the same. In practice, they are not..."

Albert Einstein

Abstract

Robots are everywhere: They facilitate comfort, convenience, technological advancement in our society, and can help create the future of it. Robotics assist humans to explore and understand inhospitable environments such as a seabed, a nuclear irradiated area and they can assist us in space exploration. In the case of deep underwater missions, which are dangerous for humans, due to pressure and current uncertainty, Underwater Robotics, used in Autonomous Underwater Vehicle (AUV) and Unmanned Underwater Vehicles (UUV) are deployed to avoid any risks to humans. The objective of this thesis was to implement a robust sonar-based SLAM algorithm for an underwater ROV and AUV. Seven prototypes were implemented to reach this objective, which included : Generation a Dead reckoning method by fusing the DVL and IMU data. The methodology involved segmentation, and creating a data buffer to extract meaningful information from the MSIS and store it. From the data collected, a map was built using Octomap and an ICP algorithm was coupled with Kalman Filter to perform re-localization. The entire implementation was done using ROS framework with UUV simulation as the environment. The results show the effectiveness of the SLAM presented as well as its limits.

Acknowledgements

I wish to express my deepest gratitude to Dr. Yvan Petillot and Jonatan Scharff Willners for their supervision, time, suggestions and advice.

I would like to thank my family and friends, who have helped and supported me during this year of study, - especially the Tinguys, who opened their home and extended their hospitality to me during the lockdown. A special thanks to Anna Freville, for her help in the proof-reading of this paper And Daria De Tinguy for her support.

And Finally I would like to thank my Mother for her unconditional love and support, without whom nothing would have been possible.

Contents

List of Figures

List of Tables

Abbreviations

ASV	**A**utonomous **S**urface **V**ehicle
AUV	**A**utonomous **U**nderwater **V**ehicle
DCKF	**C**entral **D**ifference **K**alman **F**ilter
DOF	**D**egree **O**f **F**reedom
DVL	**D**oppler **V**elocity **L**og
EKF	**E**xtended **K**alman **F**ilter
GT	**G**round **T**ruth
ICP	**I**terative **C**losest **P**oint
IMU	**I**nertia **M**easurment **U**nit
KF	**K**alman **F**ilter
MSCKF	**S**imultaneous **L**ocalization **A**nd **M**apping
MSIS	**M**echanically **S**canned **I**maging **S**onars
OKVIS	**O**pen **K**ey-based **V**isual **I**nertia **S**LAM
PF	**P**article **F**ilter
RMS	**R**oot **M**ean **S**quare
ROS	**R**obot **O**perating **S**ystem
ROV	**R**emotly **O**perated **V**ehicle
ROVIO	**RO**bust **V**isual **I**nertia **O**dometry
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SONAR	**SO**und **N**avigation **A**nd **R**anging
UKF	**U**nscented **K**alman **F**ilter
URDF	**U**nified **R**obot **D**escription **F**ormat
UUV	**U**nmanned **U**ndersea **V**ehicle

Dedicated to Larry Tesler…

Chapter 1

Introduction

1.1 Motivation

Robots are everywhere: They facilitate comfort, convenience, technological advancement in our society, and can help create the future of it. Autonomous robotics systems assist humans to explore and understand inhospitable environments such as a seabed, a nuclear-irradiated area and they can assist us in space exploration. In the case of, deep underwater missions, which are dangerous for humans, due to pressure and current uncertainty, Underwater Robots such as Autonomous Underwater Vehicle (AUV) and Unmanned Underwater Vehicle (UUV) are deployed to avoid any risks to humans. These robots can undertake certain tasks such as a search for a leak, fix a mechanical issue on an offshore oil platform, or explore a wrecked ship.

Indeed, the underwater environment brings constraints that require specific features, management and solutions. Firstly, the high attenuation of radio waves makes communication with the surface very hard [3] and hinders the possibility of GPS communication. Secondly, propagation delays, amplitude and phase fluctuation, the Doppler effect due to the motion makes position estimations and communications difficult [4]. Furthermore, underwater cameras, often have issues, when trying to find recognizable features due to poor visibility. Thus, communication with the surface is limited. A cable between the robot and the surface restricts the area to be explored and acoustic communications cannot transmit a significant amount of data. Nevertheless, an underwater robot needs to be able to explore unknown environments independently, without constraints, and be able to have a good interpretation of their surroundings. Therefore, Simultaneous Localization and Mapping (SLAM) is an indispensable key for AUVs/UUVs to perform tasks successfully. Furnishing a robot with an accurate map and position estimates enables it to work autonomously-thereby ensuring the success of a mission and the safety of divers.

Section 1.1 explains the purpose and importance of this particular SLAM process and how it can help UUVs solve the particular issue of navigating itself in an underwater environment. The next section 1.2 will detail the specifications and outline the framework of the project.

1.2 Project Description & Disclaimers

My original project proposal was " Multi-modal Data Fusion for Shared Sub-sea Maps", which was to be completed during a 6-month internship with SeeByte Ltd., Scotland. However, owing to the Cover-19 pandemic, I have had to re-orient my thesis to "Data Fusion for SLAM in an Underwater Environment'.

Recent approaches to robot localization in complex environments have been data intensive, for example, detailed LIDAR-derived maps have been used in self-driving cars. In the underwater domain, the ability to gather or exploit equivalent data is more limited, due to both-the difficulty of access and the characteristics of the sensors typically deployed. Most of the work done around offshore energy installations use remotely operated vehicles (ROVs) fitted with video cameras and sonars. The video data is used for close-range navigation of several metres as well as, to inspect a system or product. Sonars are used as a piloting navigation aid over longer ranges of distance, where the use of video is not practical. This project will evaluate simultaneous localisation and mapping (SLAM) algorithms for fusing sonar with Doppler Velocity Log (DVL) and Inertial measurement unit (IMU) to produce maps for autonomous underwater vehicles (AUV) navigation or an underwater ROV. By using a combination of real-world and simulated data, different SLAM variants will be evaluated by:

• Extracting range data from sonar images, DVL and IMU;

• Building depth maps and converting into a 2D reconstruction of a broader scene;

The resulting 2D maps and the localisation methods arrived at-from different algorithms-will be assessed in a simulation-based environment or by taking into account the known features in the context of real-world data.

1.2.1 Disclaimers

As you may know, The project has been achieved during the Covid-19 pandemic. Therefore, The project did not followed the project plan excepted. The university has closed in April and the robot and facilities where closed as well. The project passed from implementing for simulation and for real underwater robot to implementing a SLAM on simulation only. The methods used to do so changed as well, the project was supposed to use a Particle Filter, with

Hough transform and land-marking. But it end-up being a scan-matching process coupled with a Kalman Filter.

Thus, The chapter 3 represent how the project was supposed to be conducted. and the Chapter 4 is how the project was actually performed.

This section 1.2 has explained the specification and the results expected as well as defined the frame of the project. The next section 1.3 will define the objectives and the expectations of the thesis.

1.3 Objectives

The thesis will focus on implementing a robust SLAM algorithm for a ROV equipped with a sonar, IMU, and DVL. The aim is to produce a robust SLAM algorithm for further application, which will, firstly, involve data extractions from sensors such as a sonar and DVL; secondly, an effective sonar image reconstruction and feature-extraction and finally, the implementation of an accurate Particle Filter localisation algorithm.

The project will be conducted in two stages: The first stage would be to create a reliable simulation scenario using Ubuntu tools and a Robotic Operating System (ROS Kinetic). The simulated scenario will include an underwater environment and thrustable model based on the BlueRov2 model. Each step of various algorithms will be performed in simulation before conducting a real water test. The results of the SLAM approaches, in simulation vs in a real test will then be compared.

The second stage would involve implementing a full architecture for SLAM algorithm using ROS tools. The SLAM implementation will be split into the following sub-tasks: Segmentation, Data buffering, Feature extracting, landmarks and PF.

The next section 1.4 will define the resources allocated to complete the expectations of the project, the robot specification, the simulator used and the changes that had to be implemented to the original project due to circumstances related to Covid-19.

1.4 Resources and Constraints due to COVID

1.4.1 Access to Underwater Environments

It is paramount that the testing provide a robust proof of the effectiveness of several SLAM Approaches. Thus, access to an underwater laboratory with tank of water such as ORCA HUB will be necessary.

Owing to the lock-down measures that took place in March 2020 due to Covid-19, I was not able to conduct a real-test on my own in Edinburgh. If the situation remains, as it is, at the time of writing this paper - and the lock-down is possibly extended, my supervisor or the ORCA HUB team have agreed to perform test for me in the laboratory and send me the results and data post-test.

1.4.2 Ubuntu & Robot Operating System

The simulation and all the related processes will be done using Ubuntu tools and open source framework such as ROS Kinetic. The framework provided, will aid in simulating the underwater environment, using UUV simulation [5] or any other simulation.

ROS Kinetic will provide the ROS bag command file that enables all the data from the tests to be gathered from tests conducted, and converts it into a format that will be treated later.

The next section 1.5 outlines the professional, social and legal issues that may arise in the execution of the project.

1.5 Professional, Social & Legal Issues

1.5.1 Professional

According to the British Computer Society's Code of conduct I, and as a future engineer, I have to adhere, and I am proud of adhering to the following guidelines :

• Public interest: my work shall not interfere with public health, privacy and security, I shall respect the right of third parties and the project shall be conducted without discrimination to ensure the equal access to the benefits of IT.

• Professional competence and integrity: I shall not claim any level of competence that I do not have, and I shall only undertake the work within my range of competences. As I do not have all the competences required to do this thesis, I shall work to acquire them.

• Duty to relevant authority (Heriot-Watt University): I shall not disclose any information about my work to any unauthorized person(s) or organization(s), due to its novelty.

• Duty to the profession: I shall act with integrity and respect in my relationships and dealings with my colleagues, and I shall help and encourage them, whenever I can do so.

1.5.2 Social

• Owing to the rising number of Covid-19 cases in Europe and the UK, Heriot-Watt university was closed and this project had to be done, remotely.

1.5.3 Legal

The main legal issues that may be relevant are:

• Data protection: All information has been referenced to the respective authors and sources. Intellectual Property: All data pertaining to my project has been stored i.e. results and development are in overleaf as well as the algorithm is on GitHub and hence all the work done during this project is the property of Heriot Watt University.

Chapter 2

Literature Review

In chapter 1 the project, the framework of the implementation and the expected outcomes have been defined together with the required and available resources. Before starting the project and implementing a sonar-based SLAM, the thesis needs to define the terms of the project as well as review state of the art methods, previously used and currently in use for underwater SLAM. Hence, this chapter 2, is a literature review covering how robotics aids, such as AUVs/UUVs in localisation; the SLAM method and the sonar based SLAM.

First, let's review the functioning and the use of the robots that will be used for this project.

2.1 Autonomous Underwater Vehicle (AUV)

An Autonomous Underwater Vehicle (AUV) is a robot that moves autonomously in an underwater environment. In general, AUVs have a shape of torpedo to enhance their hydrodynamic performance and reduce their battery consumption. They can be small enough to be carried by a person or be several meters in length and heavy-weight. The lifetime of the battry-pack can last between a few hours to a few days. Most of the time, a trajectory is predetermined on the surface and the robot follows it. Once the mission or task is completed and the AUV emerges to the surface, data gathered during its submersion is retrieved for further investigation [6]. As mentioned in the chapter 1 the scope of communication to and from the robot is limited. Acoustic modems are the only way to communicate with the AUV and these do not allow for a high data-flow rate and induces a lot of noises. AUVs are propelled by magnetic motors that ensure the integrity of the electronic equipment inside the robot. Some AUVs can improve their motion using a sailing ballast system. This system permits the AUV to improve its momentum and enhance its hull resistance.

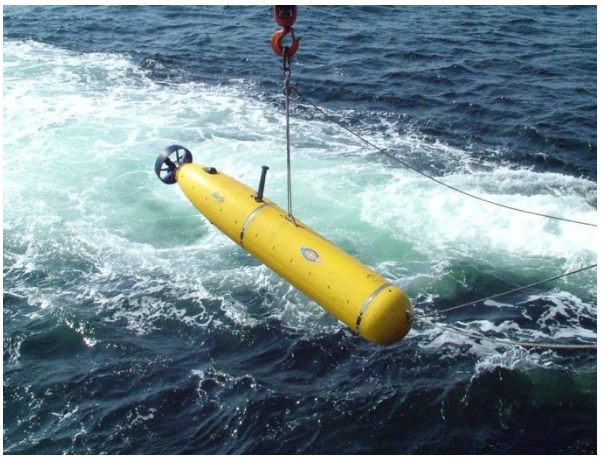


FIGURE 2.1: Picture taken of the Battlespace Preparation Autonomous Underwater Vehicle (BPAUV) by an employee of Bluefin Robotics Corporation during a US Navy exercise. [1]

2.2 Simultaneous Localization And Mapping

SLAM stands for "Simultaneous Localization and Mapping", and it consists of a process, wherein a robot or an autonomous vehicle build a map to localise itself. Robots can be classified into two categories:

- industrial robots that are tethered and achieve repetitive tasks in a known environment and
- autonomous robots that have to work in an unknown environment.

An industrial robot requires precision and robustness to perform a repetitive task and does not require a lot of sensors. In contrast, an autonomous robot requires more sensors, in order to reconstruct their surroundings, as a prior requirement, before performing tasks. As autonomous robots have to deal with an unknown environment (e.g. underwater cave, Nuclear Area), they need to perform SLAM to execute their mission and the tasks they have been assigned to do, underwater. In practice, these two problems can't be solved separately. A map is needed to determine where the robot is, and the location is needed to build a map according to the information that the sensors give. It's a chicken and egg situation. Both of these actions need the other to define themselves. As mentioned in*Chapter 1*, the underwater environment brings unique communication and sensor constraints such as high attenuation of electromagnetic waves, low visual accuracy, etc. Therefore, in the absence of external references (GPS, Wi-Fi) performing SLAM becomes a key technique for an autonomous robot to navigate and perform tasks in a new and unknown environment. [7] [3]

[1] R. B. Wyn et al,"Autonomous underwater vehicles (auvs): their past, present and future contributions tothe advancement of marine geoscience,"Marine Geology, vol. 352, no. C, pp. 451–468,2014

According to Gamini Dissanayake et al, "...a good understanding of the constraints is the key to implement our program and the most effective algorithm for SLAM depends much on the operating environment." [8] Hence, the type of SLAM to be implemented, relies on the environment that is being explored, and the possible features that environment may have and that need to be extracted. For example, if the robot has to deal with an indoor environment, SLAM will involve the detection of geometric features such as line, circle and square, by which the robot would be able to take measurements to locate itself. Therefore, a good definition of the environment is the key to choosing and implementing SLAM algorithms.

2.2.1 Approaches to SLAM

As discussed above, the type of SLAM to be used depends on the environment to measure. There are many ways for a robot to perform SLAM based on their sensors and the type of environment that needs to be explored. Through the years, the methods used for measurement and performing SLAM has become more and more powerful. While they all used the same probabilistic paradigm, they all built their SLAM approach differently. These approaches can be group into a family.

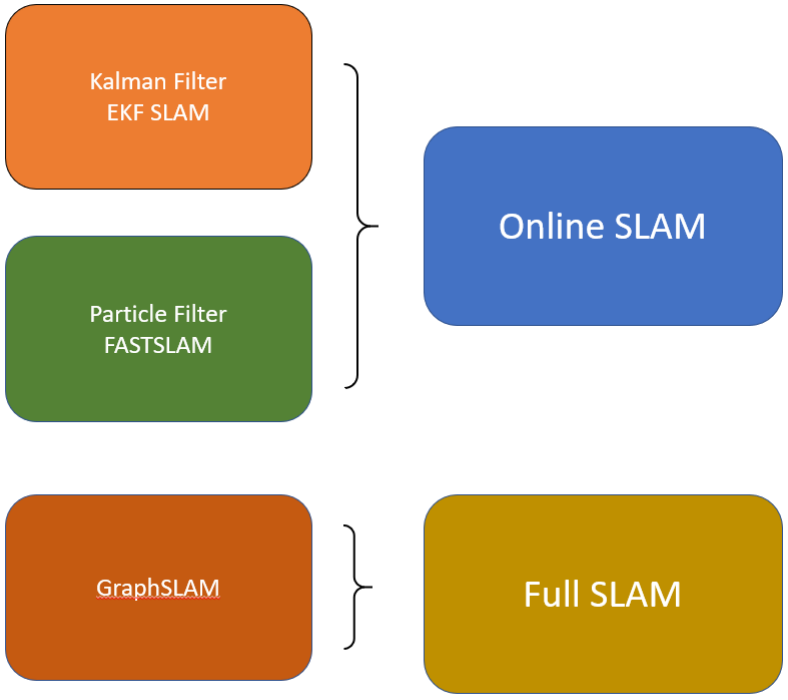


Figure 2.2: Family of SLAM

2.2.2 Online SLAM

The Online SLAM Family includes all the methods that use the Bayes rules and Markov Assumption as a framework for recursive state estimation. The next pose of the robot is estimated by the most recent pose. If every n pose relies on the $n-1$ and it is known that every transformation between $n-1$ and n includes a correction that reduces the uncertainty, then the map and the pose can be assumed to be accurate. This method describes linear and non-linear motion and observation model and uses Gaussian and other types of distribution to estimate position.

Bayes Filter:

Bayes Filter is a general approach for unknown function estimation for recursive probabilistic density. The Bayes filter is an algorithm used in computer science to estimate multiple pose estimations for a robot to know its orientation and position. It allows the robot to update its most probable position for a defined coordinate frame by correcting its belief via sensors.[9]

A Robot's pose estimation cannot rely only on its odometry. Every odometry induces a factor that generates drifting problems through time and assume the robot's position is further than where it actually is. As, perfect odometry in the real world doesn't exist, the Bayes filter is important. A robot needs to update its position using an external sensor to stop the uncertainty from growing. This recursive method can be represented by using a graph. Every x_k stands for a position and every z_k represents the observation at the state k.[10]

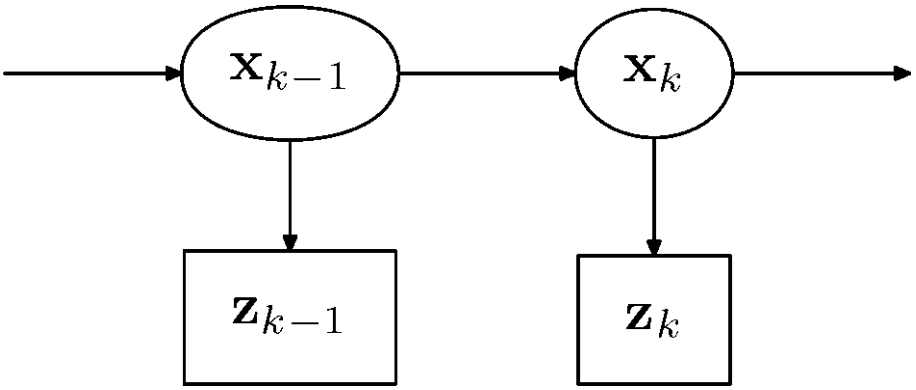


FIGURE 2.3: Representation of a Bayesian Network applied on a Hidden Markov model [2]

- **Motion Model**

[2] https://upload.wikimedia.org/wikipedia/commons/8/81/HMM$_K alman_F ilter_D erivation.svg$

$$P(x_t|x_{t-1}, u_t) \tag{2.1}$$

P = Probabilistic distribution
x_t = New pose
x_{t-1} = Old pose
u_t = control

- **Observation Model**

$$P(z_t|x_t) \tag{2.2}$$

P = Probabilistic distribution
x_t = New pose
z_t = Observation

- **Robot Localization**

$$P(x_{0:T}, m|z_{1:T}, u_{1:T}) \tag{2.3}$$

P = Probabilistic distribution
$x_{0:T}$ = motion model $t \in [0; T]$
m = given map
$z_{1:T}$ = Observation model $t \in [1; T]$
$u_{1:T}$ = Control $t \in [1; T]$

Next, by using the Bayes theorem, the distribution of the robot's localisation can be determined or found out.

$$P(A|B) = \frac{(P(B|A)P(A))}{(P(B))} \tag{2.4}$$

in our case:

$$P(x_{0:T}, m|z_{1:T}, u_{1:T}) = \frac{P(z_{1:T}, u_{1:T}|x_{0:T}, m)P(x_{0:T}, m)}{P(z_{1:T}, u_{1:T})} \tag{2.5}$$

The figure below explains the uses of the Bayes filter applied to SLAM problems.

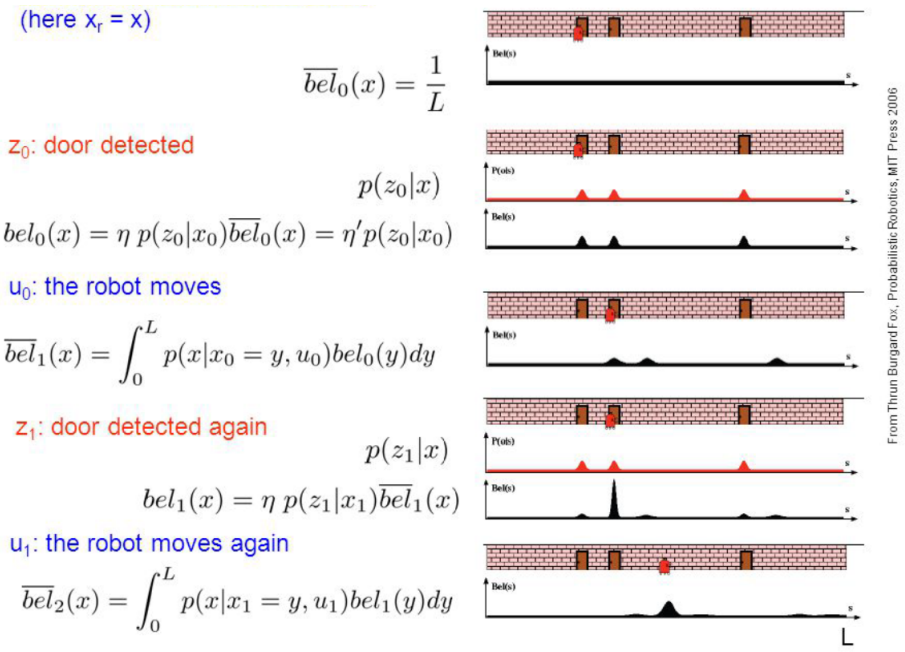


FIGURE 2.4: Bayes Filtering to solve SLAM problem from [3]

With the Bayes filter explained above, next the localisation methods using this approach can be defined.

2.2.2.1 Kalman Filter

In the Kalman filter(KF) every distribution is assumed to be Gaussian. It simplifies calculus but only works for linear models. The distribution is used to weigh the prediction and measure the Gaussian distribution, which can be summarized by using two values: the mean μ and the variance σ^2. The Gaussian represents the uncertainty of our position and measurement through a probabilistic density.[11]

[3] Thrun, S., Burgard, W. & Fox, D., 2006. Probabilistic robotics /, Cambridge, Mass.: MIT P.

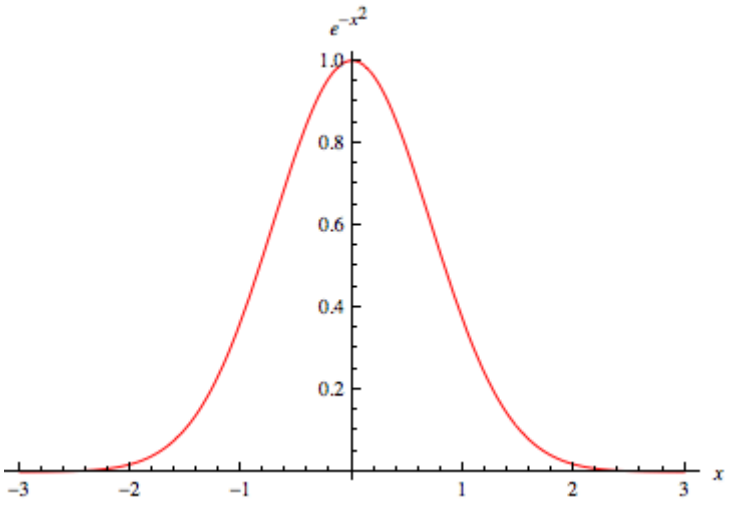


FIGURE 2.5: Gaussian Function taken from [4]

If both of the motion and measurement distributions are assumed to be Gaussian. Then, a new probability of the Robot localization is calculated.

Where the new mean value is:

$$\mu' = \frac{r^2\mu + \sigma^2 v}{r^2 + \sigma^2} \tag{2.6}$$

and the new variance is:

$$\sigma'^2 = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}} \tag{2.7}$$

Here is an example a Kalman filter applied to an airplane : The new density of probability is computed using the mean and the variance of the measurement model and the motion model. The new curves represent the current estimation:

[4] https://mathworld.wolfram.com/GaussianFunction.html

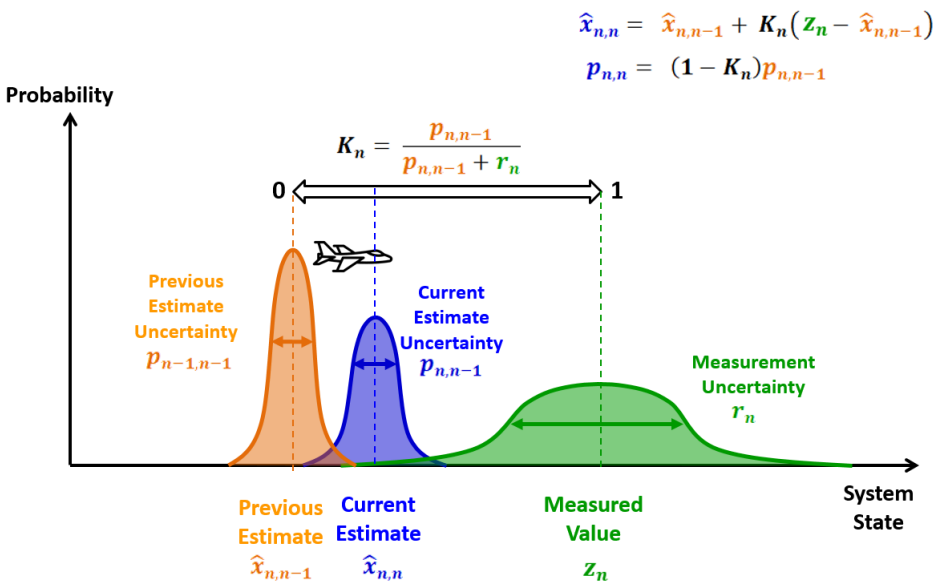


FIGURE 2.6: Explanation of the Kalman filter using the example of an airplane [5]

Extended Kalman Filter

As described in the previous section 2.2.2.1, the Kalman filter is limited to linear systems due to the Gaussian assumption. However, as most real systems are not linear, this filter is not used anymore. Instead, the Extended Kalman filter (EKF) is said to be one of the answers for non-linear models. In the case of EKF the observation and evolution can be differentiated. The purpose of this technique is to locally linearize the system using a Taylor expansion. The matrix defining the motion model will not be an identity matrix, as before, but instead, the Jacobean Matrix 2.7. "The EKF works well in practice for moderate non-linearities and large uncertainty leads to increased approximation error". [12]

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

FIGURE 2.7: Jacobian Matrix [6]

There are many Kalman filters such as Central Difference Kalman Filter (DCKF) which is described by C.Stachniss [10]) and the Unscented Kalman Filter (UKF) that uses a set of sigma points wherein each point is weighted and every new Gaussian is calculated using those points.

[5] https://www.kalmanfilter.net/kalman1d.html
[6] $https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant$

It makes this technique more accurate than the EKF. [13] However, for underwater environments, we can easily see that KF is limited compared to the non-linear one [14]. While there are many more Kalman Filters, only the relevant ones to this thesis will be mentioned, henceforth.

2.2.2.2 Particle Filter

The Particle filter (PF) does not need Gaussian approximation, and is an alternative to the Kalman filter. Therefore, the PF can compute any probabilistic distribution by using particle sampling. Every particle can be seen as a hypothetic pose and a weight representing a likelihood outcome.

$$X = \left\{ \langle x^{[i]}, w^{[i]} \rangle_{i=0,\dots,j} \right\} \tag{2.8}$$

x = the pose estimation
w = Weight of this position

The samples represent the posterior

$$P(x) = \sum_{i=0}^{j} w^{[i]} * \delta_{x^{[i]}}(x) \tag{2.9}$$

P = the density of probability
W = the Weight
δ = Dirac function

By using this method, the function is discretized using samples. Consequently, more particles in an area increase the probability of motion estimation in this area. Once the particles are sampled, a ratio between the proposal distribution and the target allows us to weight them. The 'Proposal curves' represent what our estimation is (Gaussian distribution in this case), while 'The Target' function is what our observation model will provide us with. [15]

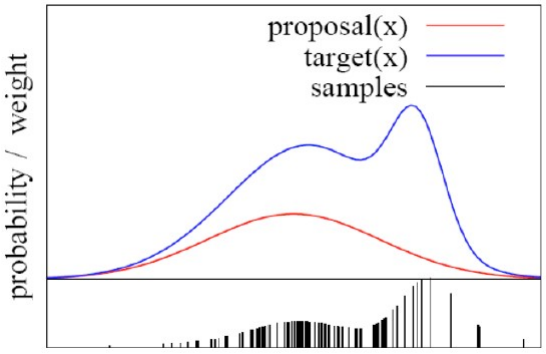


FIGURE 2.8: Weighted samples in the Particle filter [7]

After the smapling part, every particle is weighted according to the ratio of those two functions:

$$w_t^{[i]} = \frac{target(x_t^{[i]})}{proposal(x_t^{[i]})} \tag{2.10}$$

Then the prediction step is made by creating samples from the proposal shape while the Correction is made by weighting the samples with the ratio of the target and the proposal function. As seen in section 2.2.2.2, this representation allows usage of any type of function- and not only Gaussian, which gives this method an advantage over the KF and EKF. The next step is to re-sample new particles of a certain weight in order to refine and exploit and then repeat the process.

2.2.2.3 Grid map

The Occupancy Grid map is an important tool for Robotics. This method discretizes the world into cells of predetermine size. Every cell has a value attributed to it, which is: between -1 and 1. -1 is an unknown cell, 0 corresponds to a free cell and 1 means occupied. It allows the robot to divide the world into squared cells. The occupancy grid map can be built using the Bayes Filter. If a previous map exists, the prior map is compared with the sensors' data to build the posterior map.

[7] https://www.semanticscholar.org/paper/2.1-Introduction-to-Particle-Filters/5898f7a39309b48a6e46fcd661ec455ef4c446e2

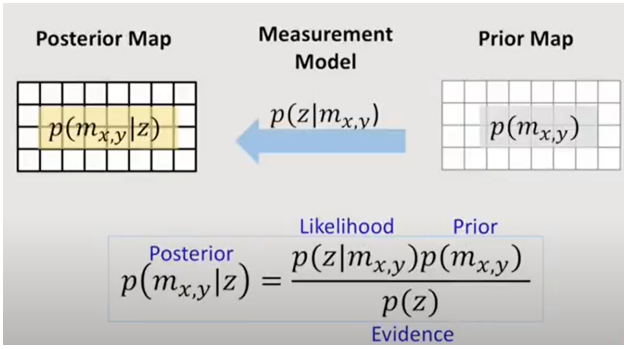


FIGURE 2.9: Occupancy Grid map representation [8]

The process of discretizing the world helps reduce the computational part of the algorithm. Indeed, the use of static state binary Bayes Filter per cell reduces the processing part of this approach. The drawback of this technique is that it requires a lot of data storage. The larger, the map becomes, a higher capacity of memory resources are required. If the pose of the robot is known, then mapping the area is a lot easier. [12]

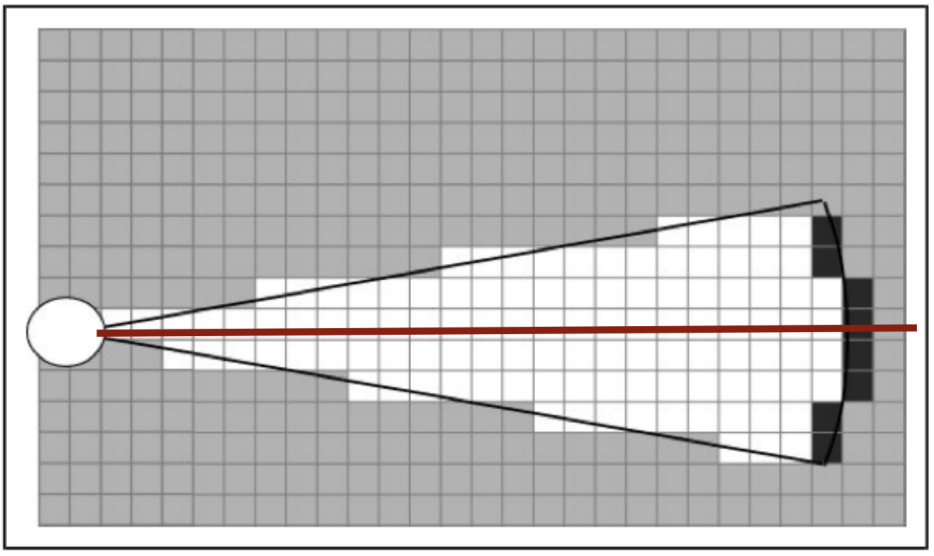


FIGURE 2.10: Grid map representation [9]

This grid-based approach relies, as the Kalman filter and the Particle Filter, on a probabilistic method. Cells are binary random variables. For a 2D map model the probability of a cell can be expressed as:

$P(cell_{(x_i,y_i)}) = 0$ for free cell

$P(cell_{(x_i,y_i)}) = 1$ Occupied

$P(cell_{(x_i,y_i)}) = 0.5$ no knowledge

[8] https://www.youtube.com/watch?v=Ko7SWZQIawM
[9] http://aslanfmh65.com/robotic-mapping/

Once the probability rule of the cells is established, some assumptions must be assumed to simplify the map equation:

- First assumption: the cells are independent of each other.
- Second assumption: the map is assumed to be static.

The map can be expressed as a product of every cell as follows:

$$P(m) = \prod_i P(m_i) \tag{2.11}$$

Estimation of the map according to the sensors z and the position x:

$$P(m_i|z_{1:t}, x_{1:t}) = \prod_i P(m_i|z_{1:t}, x_{1:t}) \tag{2.12}$$

Using the Bayes rule and Markov assumption, The equation is:

$$P(m_i|z_{1:t}, x_{1:t}) = \frac{P(m_i|z_t, x_t)P(z_t|x_t)P(m_i|z_{1:t-1}, x_{1:t-1})}{P(-m_i)P(z_t|z_{1:t-1}x_{1:t})} \tag{2.13}$$

And:

$$P(-m_i|z_{1:t}, x_{1:t}) = \frac{P(-m_i|z_t, x_t)P(z_t|x_t)P(-m_i|z_{1:t-1}, x_{1:t-1})}{P(m_i)P(z_t|z_{1:t-1}x_{1:t})} \tag{2.14}$$

By creating a ratio of these two probabilities:

$$\frac{P(m_i|z_{1:t}, x_{1:t})}{P(-m_i|z_{1:t}, x_{1:t})} = \frac{P(m_i|z_t, x_t)}{1 - P(m_i|z_t, x_t)} \frac{P(m_i|z_{1:t-1}, x_{1:t-1})}{1 - P(m_i|z_{1:t-1}, x_{1:t-1})} \frac{1 - P(m_i)}{P(m_i)} \tag{2.15}$$

- First term : Uses the Z
- Second term : is a recursive term
- Third term : Prior

Then, turn the ratio into probability:

$$\frac{p(x)}{1 - p(x)} = Y \tag{2.16}$$

...

$$p(x) = \frac{1}{1 - \frac{1}{Y}} \tag{2.17}$$

Using the logarithmic notation, the product is transformed into a sum:

$$l(m_i|z_{1:t}x_{1:t}) = l(m_i|z_t x_t) + l(m_i|z_{1:t-1}x_{1:t}) - l(m_i) \tag{2.18}$$

- First term : Inverse sensor model
- Second term : is a recursive term
- Third term : Prior

Then by creating a loop, the process can be repeated for every cell to build the map. However, this final formula does not prevent the drifting odometry and hence, to address this issue, a scan matching-such as Iterative Closest Point, Scan to Scan, Scan to Map or Feature-base has to also be considered [12].

Furthermore, the grid map approach is, most of the time, the first step of the SLAM and is be coupled with other methods such as Kalman filter [16] or Particle filter. Overall, this approach allows us to discretize the surrounding environment and make the computation part more efficient. As every SLAM technique must discretize the world before mapping, the grid map occupancy approach will be adopted in this thesis as a first step before mapping.

As seen in the section 2.2.2 onlineSLAM is a very powerful method of SLAM. These techniques used are known to reduce many errors except an incremental error. However, FullSLAM explained next is a solution to erase this remaining error.

2.2.3 FullSLAM

So far, this thesis has covered the Iterative Bayes approach of SLAM in the setion 2.2.2, wherein the new position is only computed by the observation and the pose of the previous instant. This method creates some issues such as the drifting of the odometry. Indeed, no odometry is perfect and the smallest error still brings an error pose estimation through time. For example, if the robot starts at x =0 and y = 0 and travels straight for 5 meters on the x-axis. The new odometry will be X = 4.95 Y=0. Iteratively, if the robot continues to travel, the error will continue to increase. Full SLAM was designed to make the covariance collapse using the loop closure effect, and to do this the SLAM needs to remember every previous position. The most common method

to do this is the Graph SLAM. The representation is created using a graphical representation of the nodes and links between the nodes. Every node is composed of pose at this instance and features extracted from the measurement. Every link is a spatial constraint that can be drawn from the odometry or visual extraction [17].

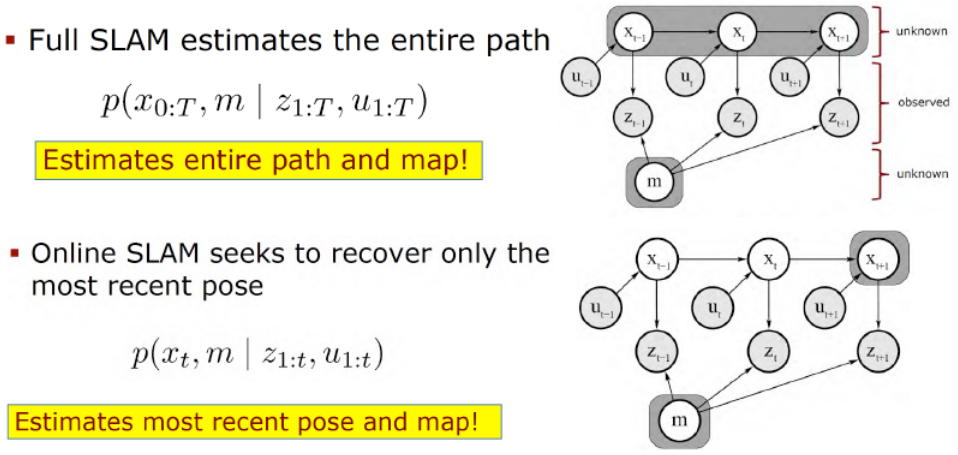


FIGURE 2.11: Full SLAM and Online SLAM

Now that all the SLAM techniques has been detail, the presentation of sonar sensor will be done in the next section 2.3.

2.3 Sonar Imaging Model

Sonar (Sound Navigation and Ranging) is a sensor that uses acoustic wave propagation in water to detect and locate objects, e.g. a seabed, another vessel , or obstacles. Sonars are generally attached to AUVs/UUVs to understand their surroundings.

The sonar used is a Mechanical Scanning Imaging Sonar (MSIS). A MSIS scans with acoustic signals in a horizontal 2-dimensional place, by mechanically rotating the transducer at a pre-set angular increment. It outputs the measured range and the intensity of the signal back-scattered from the environment.

A sonar system generates an acoustic wave in the form of an arch, that travels without an echo until it reaches an underwater object. The first echo to return indicates that the wave has reached the object or the ground. For instance, once the wave touches the seabed, it will be reflected. The angle of the seabed creates a low-intensity echo indicating that the echo has reached the floor level [18]. Once the echo hits a more significant obstacle the wave reflected by it, will be sensed by the sonar, by an electric signal with an certain amplitude and time of arrival. Using these two values, the distance from this obstacle can be interpret, and eventually its type of surface or its angle. These values are computed based on the speed for underwater acoustic wave 1.500m/s and the amplitude of the returned signal measured in mV [19].

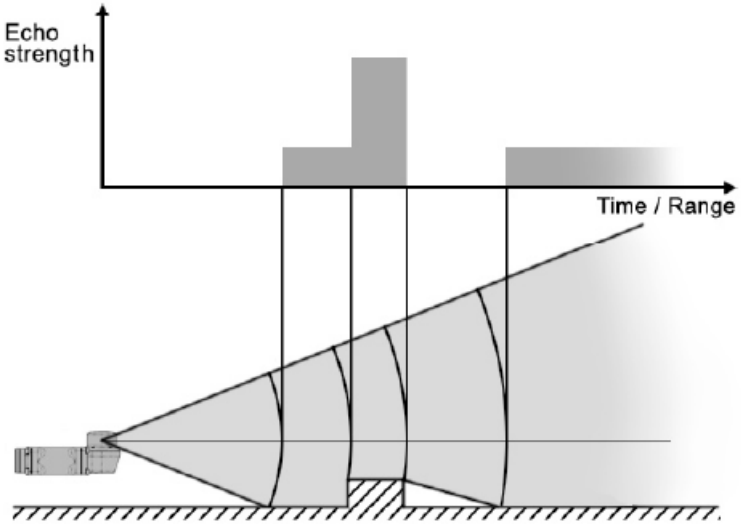


FIGURE 2.12: Sonar principle [10]

Resolution

By modulating the frequency of the wave, the sonar modifies its observation angle. The lower the frequency is, the wider the angle will be, resulting in a poorer quality image. Generally, an AUV uses this type of scanning in shallow waters and larger areas. In the case of a high frequency signal $> 200Hz$, the sonar will give a higher image resolution on a thinner angle (10° to 20°). As this type of scanning provides a more precise imagining, it is often used in instances where more clear imaging is required.

Acoustic Shadow

A phenomenon, known as acoustic shadowing is visible when using sonar. It is characterized by a drop in electric intensity, after an obstacle detection. The absence of intensity means that this zone remains undiscovered. As explained in David Ribas et al [19] the length of a shadow can inform us about the material features of the object encountered, which is essential for programming a SLAM algorithm.

Uncertainty in Azimuth

In Sonar scanning the returned echo is a 2D signal, which does not give us any information about the vertical positioning of the obstacle. This lack of information makes it harder to map obstacles in 3D mapping. The positive side of this effect is that it helps optimise the sensor to only detect an object. [19].

[10] D. Ribas,Underwater SLAM for Structured Environments Using an Imaging Sonar by DavidRibas, Pere Ridao, Jos e Neira.STAR (Series : Springer-Verlag), Berlin, Heidelberg: SpringerBerlin Heidelberg, 1st ed. 2010. ed., 2010.

The best example of this limitation is shown in the *figure 2.13* when two objects are detected at the same radial distance from the sonar, and they both create the same echo. This results in the same sonar image coming from two different objects at two different positions.

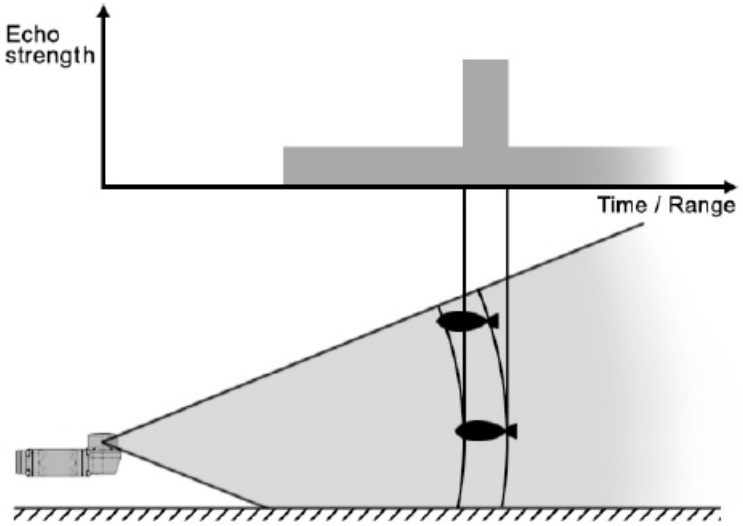


FIGURE 2.13: Sonar localizing two object at the same radial distance [11]

Reflection

Another very common effect, mostly visible in water tanks, is the phantom reflection of echoes. If a wall is wide enough in an enclosed area, the acoustic pulse created will have some phantom reflections, which, in turn will generate a weaker electric signal measurement after the main reflection. [19]

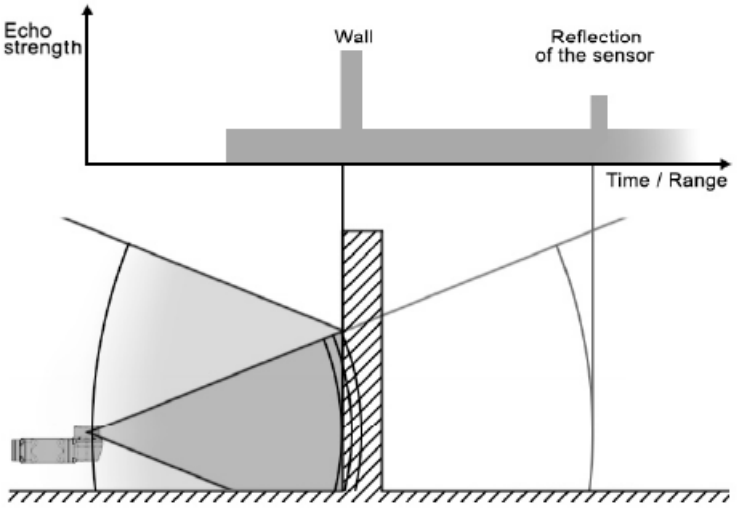


FIGURE 2.14: Phantom echo reflection of a sonar [12]

[11] D. Ribas,Underwater SLAM for Structured Environments Using an Imaging Sonar by DavidRibas, Pere Ridao, Jos e Neira.STAR (Series : Springer-Verlag), Berlin, Heidelberg: SpringerBerlin Heidelberg, 1st ed. 2010. ed., 2010.

The sonar image system has been presented. Sonar-based SLAM techniques are reviewed in the next section.

2.4 Underwater SLAM technique

2.4.1 Sonar-based

The Sonar-based approach relies on the Doppler Velocity Log (DVL), Inertia Measurement Unit (IMU) and Sonar. The DVL and IMU estimate the position and the orientation of the robot, while the sonar does the mapping and feature extraction for further reconstruction. It's a very common technique used in underwater robotics because it does not require any equipment such as a Camera or Acoustic communication. Even though, there may be a lack of visual accuracy underwater, the robot can perform SLAM. One of the problems with this approach is that it requires feature extraction which makes it less efficient for an unstructured environment.

The sonar-based SLAM uses the Sonar to get observation from the surroundings. The sonar image is then treated and used to recognized a structured environment to provide landmarks to KFs and PF for SLAM.

2.4.1.1 Extended Kalman Filter Approach

To apply the Kalman Filter to a Sonar based SLAM, the G and H matrices are defined with our robot's hardware. G is the Motion model. The more DOF the robot has, the bigger the matrix will be. The H Matrix represents our observation. The observation is computed using the sonar. The algorithm will draw landmarks that will be used as reference for our observation model. Then the implementation will compute the new covariance and the new pose estimation using the following algorithm:

Algorithm 1 Extended_Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

$\overline{\mu}_t = g(u_t, \mu_{t-1})$
$\overline{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
$K_t = \overline{\Sigma}_t \, H_t^T \, (H_t \, \overline{\Sigma}_t \, H_t^T + Q_t)^{-1}$
$\mu_t = \overline{\mu}_t + K_t \, (z_t - h(\overline{\mu}_t))$
$\Sigma_t = (I - K_t \, H_t) \, \overline{\Sigma}_t$
return μ_t, Σ_t

[12]D. Ribas,Underwater SLAM for Structured Environments Using an Imaging Sonar by DavidRibas, Pere Ridao, Jos e Neira.STAR (Series : Springer-Verlag), Berlin, Heidelberg: SpringerBerlin Heidelberg, 1st ed. 2010. ed., 2010.

For sonar-based methods the difference between each type of Kalman filters becomes less important according to various studies (method of Extended Kalman filters for sonar-based [20]; the study on the use of integrated Unscented Kalman filter [21] and the paper on the use of Unscented Kalman filter [22]).

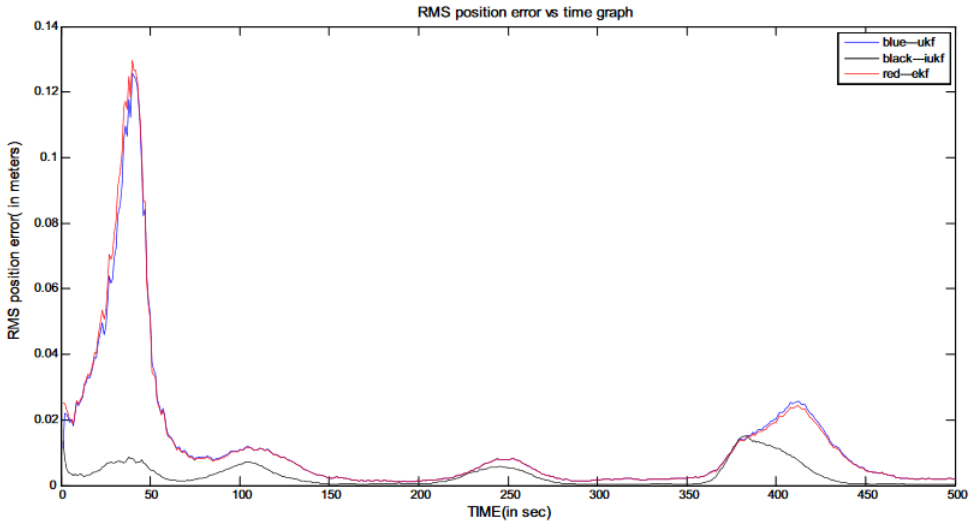


FIGURE 2.15: Root Mean Square error for different type of Kalman Filter

The first graph (figure 2.13) represents the RMS of the errors of different Kalman filters through time. It shows that the difference between UKF and EKF is less significant. It also shows us the efficiency of the IUKF over those two methods. Another point to take into account is the rising of the RMS at the beginning of IUKF and UKF method. The Kalman filter is the most common approach to solve data fusion and SLAM for a system. Its advantages are that it's very simple to implement and has low computational needs.

2.4.1.2 Particle Filter Approach

For the sonar-based PF, the approach is slightly different. The Matrix of the motion model can be assumed to be Gaussian or take any form. Then, the PF samples the distribution of the observation model. Next, the pose estimate is computed using a weighted function between the motion model and the samples drawn from the observation model using the following algorithm.

According to Bo He et al (2012) [23] the Particle filter has some advantages over the Kalman filter. First of these, is that it does not require an initial pose estimation, which allows us to not have a steep rise of covariance at the very beginning of the SLAM. Another advantage of the PF over the EKF is that the SLAM can recover from a wrong convergence. The total accuracy

Algorithm 2 Particle_Filter(χ_{t-1}, u_t, z_t)

$\bar{\chi}_t = \chi_t = \emptyset$
for $m = 1$ *to* M **do**
  sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
  $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
end for
for $m = 1$ *to* M **do**
  draw i *with probability* $\propto w_t^{[i]}$
  add $x_t^{[i]}$ *to* χ_t
end for
return χ_t

of the PF is much higher than EKF. [24] However, the Kalman filter takes less computational resources, while the PF suffers from a sample impoverishment problem [25].

The debate could be continue on the pros and cons of these two techniques and see which one is the most suitable for our application but, as they both rely on the same probabilistic Bayes approach, they are not necessarily incompatible with each other. Nowadays, most scientific papers on sonar-based Underwater SLAM show that both techniques can be used on the same architecture. Both of these methods have their pros and cons and the best SLAM algorithm is the one that takes advantage of both.

Indeed, the sonar-based approach can have several crossovers between these two methods, as Maurelli et al described in their paper [26]. They used the particle filter approach for the initialisation of SLAM, and once, the estimate pose was accurate enough, it was then switched to the EKF approach, and then the PF was recalled after a certain amount of time to check on the convergence. Another very common method was to use the particle filter with Kalman filter on every particle [27] or use it to retrieve data from the sonar by using PF, and then using the EKF to fuse data [24]. The approach is usually chosen based on several constraints of a project , such as: what type of environment is to be explored, the computational and memory resources that its required and what is available to be used.

2.4.2 Visual Inertia based

Visual-inertia-based SLAM is becoming more and more popular in robotics because of its affordable Camera (when compared to any sensors), its robustness, and its wide range of applications such as autonomous car, AUV and drone flight [28]. For AUV, Visual-inertia-based SLAM uses the DVL, the IMU for the inertia and the Camera for the visual part. Indeed, this area of research has received significant interest from the scientific community, in the last decade. The inertia-based SLAM proposes a powerful alternative to GPS. However, visual SLAM does have

	KF	PF
Computational effort Complexity	m^2 m: Features	N*log(m)
Assumed Distribution	Gaussian	Pose : Arbitrary Motion : Gaussian
Linearisation	KF all linear EKF One Linearisation	Motion do not need linearisation
Landmarks Handled	Hundreads	Thousand
Flexibility	++	+++
Large Scale	+	+++
Require Initial pose Estimation	Yes	No
Mulitmodal	Yes	No

TABLE 2.1: Recapitulating comparison between PF and KF [1]

some issues due to the changing illumination and motion blurring [29]. The techniques can be divided in two groups: Filtering-based and Optimisation-based.

Optimization-based methods have shown to achieve better localization accuracy and better memory use, whereas, the filtering-based is known to have an advantage in computational resources [28].

2.4.2.1 Filtering-based

The filtering-based approach is the online SLAM approach that was discussed in the section 2.2.2. The most common approach is the EKF and PF. The robot has its motion model and measurement model and uses the feature position to correct its pose estimation. The new estimation is made using only the previous one, which does not allow our model to perform loop closure. The most well-known approaches in Visual-inertial SLAM, which use this method are MSCKF and ROVIO. They are both described respectively in Bloesch et al. (2015) [30] and Sun et al. (2018) [31].

	Rovio	MSCKF	SVO-MSF
CPU Ressources	100%	150%	65%
Memory Ressources	10%	19%	20%
Process Time[ms]	40	50	15
RMS[m]	0.3	0.6	0.7

TABLE 2.2: Effectiveness of open-source Filtering based VI-SLAM [2]

2.4.2.2 Optimization-based

The Optimization-based method uses the Graph SLAM approach which stores the measurement and motion in a graph structure. Every node of the graph represents the robot's position and measurement at an instant and every node is linked to another with a spatial constraint. The constraints can be drawn from the odometry and the visual features. Once the graph draws the optimization part will perform loop closure and make the covariance collapse. Famous methods are OKVIS and Vins-Mono. More details on these methods are covered by Li, P et al [32] and Qin, T et al [33].

	OKVIS	**VINS-Mono**	**SVOGTSAM**
CPU Ressources	175%	175%	165%
Memory Ressources	15%	17%	25%
Process Time[ms]	20	100	30
RMS[m]	0.25	0.2	0.1

TABLE 2.3: Effectiveness of open-source optimization based VI-SLAM [2]

2.4.2.3 Visual Inertia for Underwater SLAM

As seen in the beginning of the section, the Visual inertia SLAM requires cheaper sensors to perform SLAM and it becomes an interesting option for small robots that cannot bring DVL inboard. Even though, the DVL is a large instrument, and it is very useful for localisation, its price and size can be a issue when building a robot. Consequently, VI-SLAM appears to be a solution to this problem. The Pre-processing part of the IMU and the camera can substitute the DVL and enable the Robot to construct a map without a pose initialization. In terms of the type of open source VI-SLAM to be used, the choice is large and it depends mostly on the type of environment to be explored, the resolution of the overall map and also the hardware on board. The table below represents the accuracy and effectiveness of some open source VI-SLAM for underwater environment [34].

	Monocular + IMU						Stereo + IMU		
	msckf	okvis	rovio	svo	vinsmono	vinsmonolc	okvis	svo	stereomsckf
Bus/Out (53m)	× ×	1.17 100%	0.7 100%	0.19 19%	2.39 100%	2.06 100%	0.78 100%	2.68 100%	0.61 100%
Cave (97m)	2.42 18%	0.92 100%	0.76 85%	0.79 31%	3.38 100%	1.48 100%	0.98 100%	1.39 52%	0.66 100%
Aqua2Lake (55m)	× ×	1.07 100%	0.45 100%	× ×	0.50 100%	0.50 100%	0.51 97%	1.42 90%	0.51 84%
Bus/Out Cave Aqua2Lake									
Bus/In DPV Aqua2Reef									

FIGURE 2.16: Effectiveness of different open-source VI-SLAM methods for underwater environment : the first row represents the RMS on the overall trajectory. The second row represents the percentage of the trajectory tracking. The evaluation was made in an wrecked bus, an underwater cave, and a lake. The last two multi-colored rows represents the effectiveness of every type of VI-SLAM. Among those values the first three rows of color stand for the global performance with the sonar enable and the last three with sonar disable [34]

.

Most Underwater VI-SLAM tries to integrate the Sonar to its algorithms. The visual accuracy of the water is subject to change. Therefore, adding the sonar as option for SLAM helps the Robot to understand its surrounding, regardless of the clarity of the water. Furthermore, the sonar provides information about the distance between the camera and what is being seen. For example Rahman, Sharmin et al. proposed a VI-SLAM using OKVIS structure while fusing sonar data [35] and their extension to integrate the depth sensor and loop closure [36].

Chapter 3

Project Plan

The purpose of the project will be to implement a robust 2D SLAM algorithm for an underwater ROV. Enabling the Robot to perform SLAM gives the robot the possibility to build a consistent map with a good approximation of its localization for any further tasks. To perform SLAM, the fusion of the IMU and DVL for a dead reckoning position will be achieved and subsequently, sonar imaging will be used for re-localisation based on observations. Chapter 3 will outline the robot model to be used, the sensors, the architecture and the project plan to meet the project goals stated in chapter 1.

Note: Most of the proposed prototypes mentioned in this chapter could not be implemented exactly as designed, due to circumstances arising from Covid-19. However, wherever possible, an alternative plan and methodology has been provided and executed.

3.1 Robot Model

The BlueRov2 is an underwater ROV comprising of 6 propellers, which allows it to move within 6 degrees of freedom. An IMU, a depth sensor and a camera are on board. The BLueROV2 uses open source software and is documented with a well-explained data sheet for the electronics and sensor management. A rospackage is also available on ROS Kinetic, which will serve as the main framework for robot development.

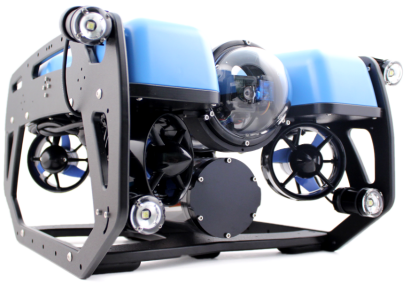


FIGURE 3.1: Picture of the BlueRov2 [1]

3.1.1 Sensors

The micron sonar will be used in this project, with a DVL and IMU sensors (specifications provided below).

3.1.1.1 Sonar

The Sonar was not on the datasheet of the official documentation for the BlueRov2 and it was installed in March 2020 by ORCA HUB. The current Sonar is a Tritech Micron Sonar; Operating CHIRP centered on 700kHz with a beamwidth of 35 degree vertical and 3 degree horizontal; a scan sector of 360 degrees, a voltage between 12V and 48V and a RS485 (twisted pair), RS232 communication protocols.

FIGURE 3.2: Picture of the Micron Sonar [2]

[1] https://bluerobotics.com/store/rov/bluerov2/
[2] https://www.tritech.co.uk/product/small-rov-mechanical-sector-scanning-sonar-tritech-micron

3.1.1.2 DVL

The Doppler Velocity Log (DVL) is an instrument that can compute the three orthogonal velocity components of the robot. The DVL is reputed for its accuracy in tracking measurements and its independent data reference-based system. The DVL uses the Doppler effect to compute velocity in three dimension. The sensor creates three co-planar microwaves in three different directions. The acceleration of the vehicle will deform the waves during its transfer according to the Doppler effect. This wave deformation is used to inform the Robot about its movement. [37]

3.1.1.3 IMU

The IMU is composed of 3 elements:

- 3 DOF Gyroscope
- 3 DOF Accelrometers
- 3 DOF Magnetometer

The IMU sensor gives us: the Roll, Pitch and Yaw components of the robot. The IMU is important for Drones and ROVs that evolved with 6 DOF in an environment.

3.2 Architecture

In terms of concept, the project plans to use ROS Kinetic on Ubuntu 16.04 as tools throughout the duration of the project. Implementation and tests will be achieved using UUV simulation and a plugin of UUV to simulate the sonar data, and eventually, a Rosbag of the real test performed underwater environment by my supervisor and the ORCA HUB members.

One of the project's goals will be to develop a robust Particle Filter. To do so, a good data extraction and process will be needed. DVL and IMU will give us a good pose estimation while the sonar image will be processed to extract features and provide the particle filter with landmarks. To perform the SLAM a Particle Filter will be implemented using the following *figure 3.3*.

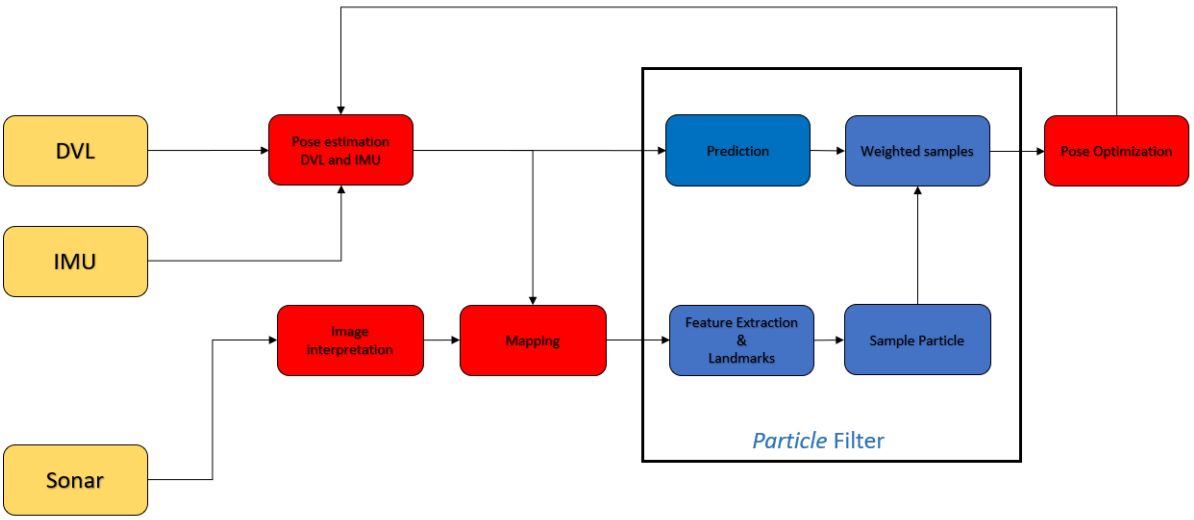


FIGURE 3.3: Architecture of the SLAM Algorithm: yellow boxes represent the sensors. red boxes for the work achieved during the project. Blue boxes for the Particle filter implementation

3.2.1 Plan & List of Tasks

The project plan was split in a list of tasks and deliverables for different stages, which acted as milestones for the project and each milestone will be described in Chapter 4, under the sub-sections of the corresponding prototype implemented.

Prototypes	Projected Time(days)
P1 - Environment and Robot Model set-up	4
P2 - IMU and DVL fusion	3
P3 - Segmentation & Databuffer	8
P4 - Mapping with Octomap	2
P5 - Hough transform & Feature extractions	10
P6 - Landmarks & Particle Filter	16
P7 - SLAM	15

TABLE 3.1: Projected Gantt Chart

P1 - Setting Environment

The project will kick-start with the selection of the robot model and the setup of the sonar simulation and ROS environment: Robot Model, Workspace, world simulation. Here, every sensor and thruster needs to be checked and ensured that they work in relation to their topics. This part is an important part of the set-up for the project. A good setup will save time and provide more time to focus on dealing with any issues inherent to ROS or Ubuntu.

• Deliverable: Working Environment
• Implementation: See Prototype 4.1

P2 - IMU and DVL fusion

The next important step is to create a good and reliable odometry topic with the rightful transform between IMU and the base frame as well as the DVL with the base frame. For now every sensor is assumed to be at the coordinate (0,0,0) of the base link of the robot. The sensors position will have to be changed according to the real robot sensor position. Once this done the project will have a reliable odometry.

• Deliverable: Dead Reckoning working
• Implementation: See Prototype 4.2

P3 - Segmentation & Data-buffer

This main task is the first stage of the sonar implementation. The segmentation will filter the data to keep only useful information of the surroundings. The Data-buffer is there to keep the old values of the sonar in a memory for future mapping and feature extraction. The implementation of a solution to distortion, induced by movement, is also a part of this stage. By the end of this task, the project would have a reliable sonar image with less noise and without discontinuity.

• Deliverable: only meaningful information on the sonar image and no Discontinuity in the sonar image
• Implementation: See Prototype 4.3

P4 - Mapping with Octomap

Once the previous stage is achieved, the sonar information and odometry can be used to map the surroundings and have a initial map for the SLAM implementation.

• Deliverable: Posterior Map
• Implementation: See Prototype 4.4

P5 - Hough transform & Feature extractions

With the initial map and the sonar image, the program can use different image processing algorithms for feature extraction such as SURF or Hough Transform. With this part done the robot will be able to extract features from its environment. Making it possible to move on to the creation of Landmarks. Deliverable: understanding and characterizing of the environment for future recognition

P6 - Landmarks & Particle Filter

With the presence of features to enhance the understanding of the surroundings, the next step will be to create landmarks that will help us for the sampling and re-sampling of particles. The Particle filter will be implemented during this period, enabling the robot to correct its odometry and map. Deliverable: Finalization of the environment recognition process and ensuring Localization algorithm is working

P7 - SLAM

The final stage is the SLAM algorithm. The purpose will be to ensure every step, incorporate is working together to achieve SLAM. This stage involved modifying the previous steps to create a robust SLAM implementation, enabling the robot to map and locate itself without external reference.

• Deliverable: SLAM algorithm working, implementation on Github and Report.
• Implementation: See Prototype 4.7

3.3 Testing and recording results

To test our SLAM approaches a final test will be performed on the robot; and the simulation will be using relevant indicators of the effectiveness of the robot architecture. The support team and I will be performing tests with several algorithm on real and simulated scenarios as per the following process:

The test is planned to be conducted in three different environments, as follows:

- UUV simulation
- ORCA Tank
- Lake (*Subject to quarantine measures of the pandemic*)

In all these above mentioned environments, the robot shall perform the navigation on a predetermined path with a different algorithm to ensure the test validity, as follows:

- Dead Reckoning
- Our method
- Gmapping (*or another robust SLAM method approved by the community*)

However, some of the testing scenarios, e.g. the lake was not available, due to Covid-19 constraints.. IF the above testings scenarios were successfully implemented, the result part would have had three different tested case scenarios with 3 different methods. The dead reckoning would have allowed us to see the differences with and without our SLAM package. The following criteria would have been used to compare those 3 methods.

- Total Accuracy: RMS [m]
- Processing consuming: [%]
- Lost convergency [%]

In the end, the robot model was tested in one out of three environments and the results used to generate SLAM. Only the third prototype could have been tested with a rosbag of the real sonar.

Chapter 4

Methodology & Results

In this chapter, the methodology used as outlined in Chapter 3, the 7 prototypes developed and tested, and the algorithms arrived at during the time allotted for the project will be described. Each prototype is explained in three parts outlining i.e. theory, implementation and results, wherever relevant, and each completed prototype represents a milestone reached in the project, which also provides crucial data required for the final SLAM.

4.1 Prototype 1: Robot Model and Sonar Simulation

As mentioned in chapter 3.1, the robot model, a BlueRov2 with kinematic model with 6 DOF: x,y,z and roll,pitch,yaw will be used to perform SLAM, in order to generate the 2D map and re-localization. By pre-determining this, we can exclude 3 of the 6 DOFs of the equation to be used in SLAM.

4.1.1 Kinematic Model

The robot model is similar to the differential drive robot, where kinematic modelling is concerned. As the BlueRov2 does not have wheels but instead has thrusters, the kinematic equations will differ. The kinematic model will not be thoroughly detailed here as we will not be using the standard equations to predict the motion or perform dead reckoning. Instead, these functions will be performed using other sensors such as DVL and IMU.

Our kinematic model can be seen as a differential robot with 3 degree of freedom, as follows :

Let the world frame reference be $\{X_{world}, Y_{world}\}$ and the robot frame reference be $\{X, Y\}$. In the Cartesian coordinate system, the robot position is called P $[x,y,\theta]$.

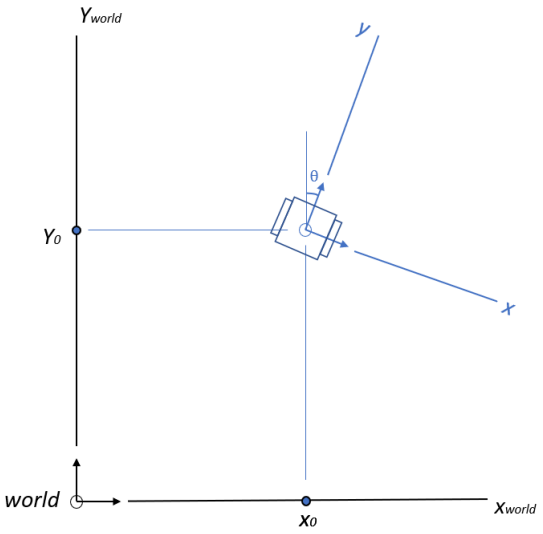


FIGURE 4.1: Robot frame in the world frame reference

The relationship between the world frame and the robot, can be written using this matrix:

$$T = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \end{pmatrix}$$

(4.1)

Now that the kinematic model of the robot has been defined, this will enable the SLAM to define the map and the Kalman filter to be used. Next, we need to equip the BlueRov2 with the missing sensors, as given below.

4.1.2 Implementation

Implementation of Prototype 1 involved setting up the robot model, workspace and world simulation, as mentioned in Chapter 3, and trouble-shooting some tasks, that did not go as planned. As the BlueRov2, available on ROS Hydro is outdated, a different ROV robot model from the UUV simulation website, namely Desistek Saga had to be used. Its sensors will need to be edited to make it function in the same way as an BlueRov2.

To ensure the substitute robot model meets the performance expectations of a BlueROV2, the DVL will need to be added at just the right position , and the FOV and the number of points of the forward sonar will need to be changed as well.

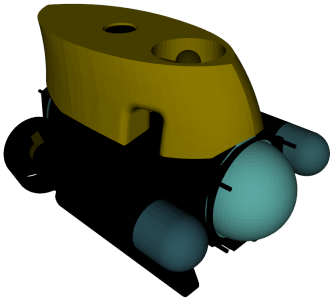


FIGURE 4.2: Robot model of the Desistek Saga

In order to execute the above mentioned modifications, the URDF of the robot's sensor was changed, as follows:

- the DVL is added
  file : *sonar_snippets.xacro*
  path : *desistek_saga/desistek_saga_description/urdf*

```
  ¡!-- DVL  --¿
¡xacro:defaultdvlmacro
  namespace="$-namespace"
  parentlink="$-namespace/baselink"
  inertialreferenceframe="$-inertialreferenceframe"¿
  ¡origin xyz="0 0 0" rpy="0 $-0.5*pi 0"/¿
¡/xacro:defaultdvlmacro¿
```

- Then, the following sonar was added, to the sonar description of the UUV simulation.
  file : *sonar_snippets.xacro*
  path : *uuv_simulator/uuv_sensor_plugins/uuv_sensor_ros_plugins/urdf*

```
  ¡xacro:macro name="micronsonar" params="namespace parentlink *or
      ¡xacro:multibeamsonar
        namespace="$-namespace"
        suffix=""
        parentlink="$-parentlink"
        topic="sonar"
        mass="0.02"
```

```
        updaterate="15"
        samples="396"
        fov="6.3"
        rangemin="0.3"
        rangemax="75"
        rangestddev="0.027"
        mesh=""¿
        ¡inertia ixx="0.00001" ixy="0.0" ixz="0.0" iyy="0.00001" i
        ¡xacro:insertblock name="origin" /¿
        ¡visual¿
          ¡geometry¿
            ¡mesh filename="file://$(find uuvsensorrosplugins)/mes
          ¡/geometry¿
        ¡/visual¿
      ¡/xacro:multibeamsonar¿
    ¡/xacro:macro¿
```

- Finally, the previous sonar called forward multibeam900 in the Desistek Saga was erased and replaced with the modified sonar snippet "*micronsonar*". And the new one was added, as follows
  file : *sonar_snippets.xacro*
  path : *desistek_saga/desistek_saga_description/urdf*

```
¡xacro:micronsonar namespace="$-namespace" parentlink="$-namespa
  ¡origin xyz="0.0 0 -0.1" rpy="0 0 0"/¿
¡/xacro:micronsonar¿
```

The robot model prototype is now set-up except for the sonar, which still remains a laser-scan with 360 degrees vision, and which needs to be converted into a pointcloud, to create a mechanical rotation of a real sonar.

Next, the environment was set up in the simulation and the sensors were tested to manage and resolve any errors that cropped up.

4.1.3 MSIS in UUV Simulation

The sonar in UUV Simulation sends outs Laserscan messages. Hence, at each step and every time, values of the whole surrounding are retrieved in a Laserscan format. To change this,

the mechanical movement of the sonar needs to be simulated first before formatting the laser. Making the change, in this sequence is much easier to do, as the Laserscan format has an implicit rotation which will be hard to handle, if done prior.

A Python file that subscribes to the sonar topic (and that publishes values that simulate a mechanical sonar) was created. This had to be done carefully, as the slot in the array will define the orientation of each value. To manage this information, the original array, which was under a tuple format was converted into a list. Once the list was filled, one value at a time was taken and the other deleted to simulate the rotation.

The main problem encountered during this part was the time synchronisation: the Topic was always late compared to the original Laserscan. This error occurred because values of the original Laserscan were stored, before performing the loop. That meant that the values in output were not from the latest scan. To solve this issue, the code subscribed at every iteration to the original Laser scan to have the latest value. Then, a converter node that redirected the Laserscan messages was generated, by using the same header but changing the format of its value into pointcloud.

4.1.4 Sonar Simulation

To locate a sonar simulation during the time allocated was an important but challenging part of the project. In UUV simulation, the sonar works as a laser. In the previous section **??**, the sonar was adapted to make it resemble (as close as possible to) a MSIS sonar. However, to validate the SLAM, more realistic data from the sonar data was needed.

Using a laserscan as a sonar, without any further modifications cannot be applied in a simulation as in reality, the sonar scan has the beamwidth to consider. As the beamwidth is hard to estimate the position of a object in a larger area. Hence, at some angles, some errors in scanning may occur with obstacles appearing bigger than they should be.

Another aspect that needed to be taken into consideration is that a Laserscan-type message is measured at the speed of light. The approximation provided by the UUV simulator is not accurate enough to consider in a project that is simulating the robot model working dynamically. In fact, in UUV simulation the motion of the robot does not induce distortion in data. To solve this problem, different sonar simulators such as, A [38] and B: [5] were tried. For simulator A, several attempts were made to contact the owner, and while he was helpful in resolving some issues, other errors of compiling persisted. Simulator B was a UUV simulation plugin that worked perfectly with the help of Jonathan S. Willner (a PhD student in Robotics).

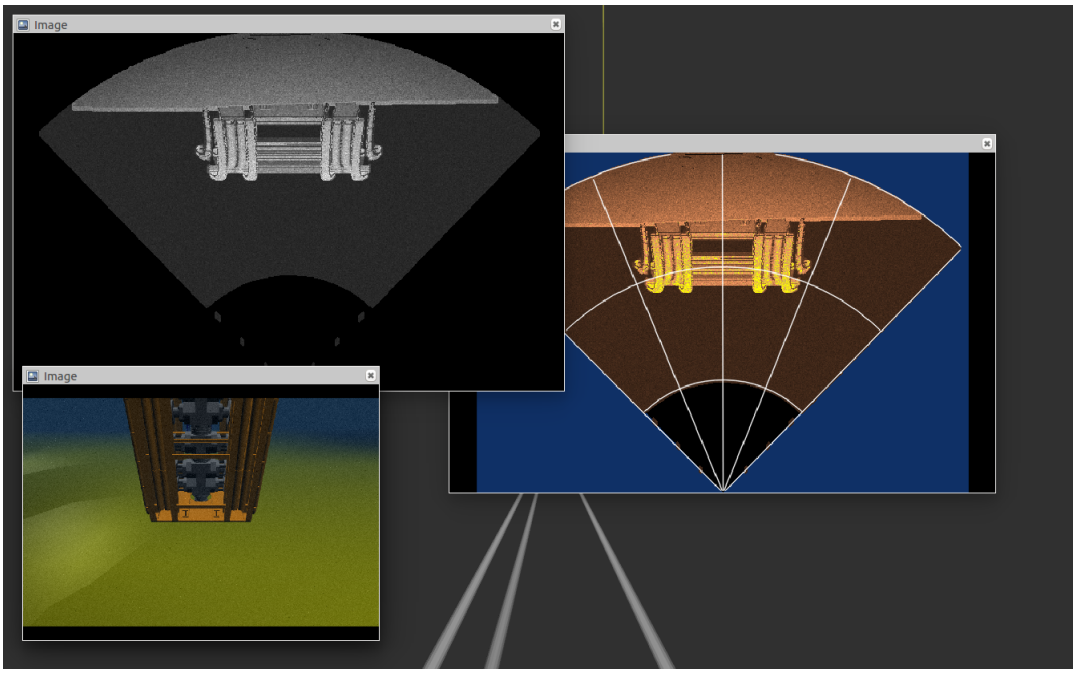


FIGURE 4.3: Images taken from Rviz once the sonar simulator launched

The simulated sonar (B) provided a more realistic image than could be expected from a sonar, better than the laserscan message. The resulting images of the simulation launched with ROS are shown in the figure 4.3, where one can see the differences. At the top left corner, a black and white sonar image is depicted next to a coloured image but with radial lines of the same sonar image. The latter provides a better image and understanding. The picture at the bottom left corner is the actual camera of the robot.

ROS and openCV were used to obtain the images from these cameras and image processing was used to retrieve decipherable shapes and forms.

A threshold colour was used to retrieve the most important pixels, and then ”Gaussian Blur, Erosion, Canny, Dilatation” effects were applied on the image. The aim was to obtain a significant shape of the surroundings, as can be seen in Figure 4.4 after threshold and Figure 4.5 after treatment.

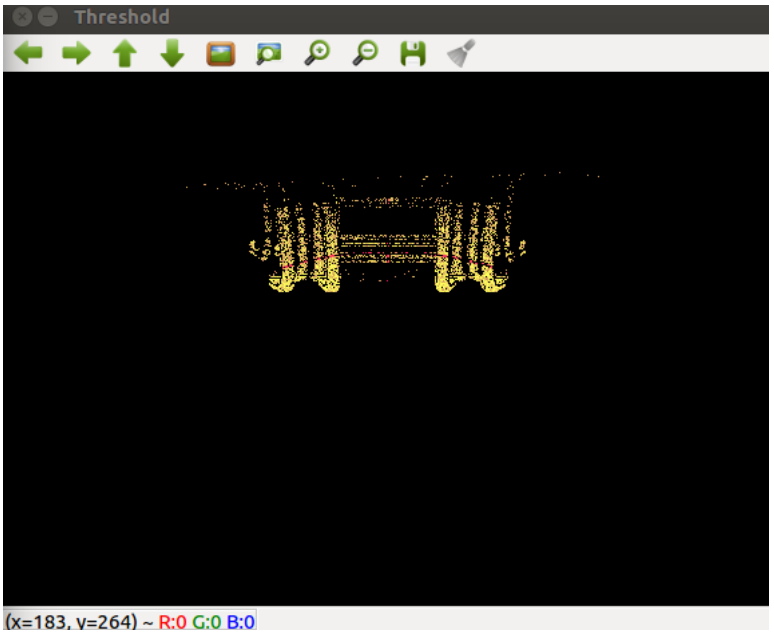


FIGURE 4.4: Sonar image after threshold

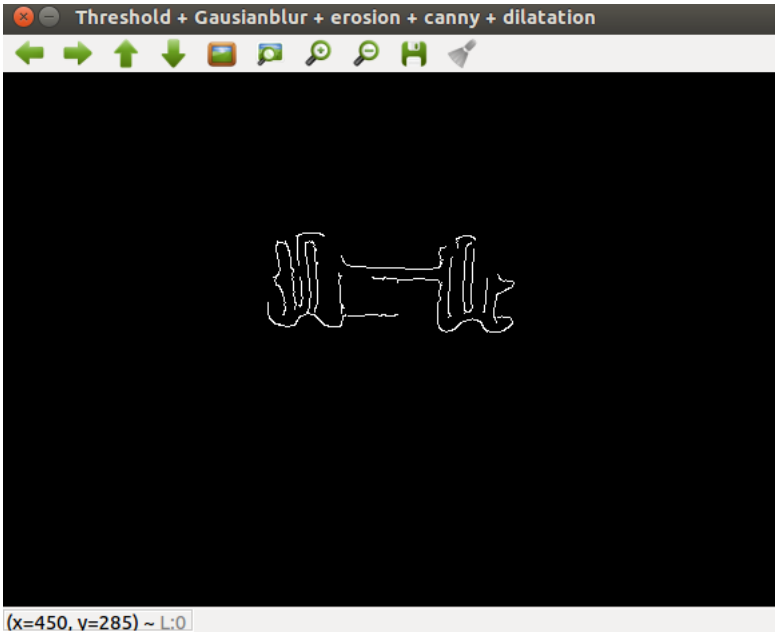


FIGURE 4.5: Sonar image after treatment

The results of the simulated sonar(B) and the image processing were convincing and optimistic. However, when the same function of the sonar was reproduced on the robot model by changing the parameters to make it work in MSIS, it was not successful. The sonar image presented in figure 4.3 is a merge of laserscan messages and multiple image cameras that reconstructed the structure from above. The process was too complicated to import into another model. Furthermore, these issues made the MSIS simulation impossible to create. After due discussion

with my supervisor, it was decided to put the sonar simulation aside and focus on the SLAM process, instead.

The environment is setup. The project was done using UUV simulator and the Desistek Saga model. Once the robot and environment define a localisation method can be created using the IMU and DVL as presented in the Chapter 3.

4.2 Prototype 2 : Dead Reckoning with IMU and DVL

In order to localise a robot, Dead Reckoning is used, which is performed by two sensors - DVL, which estimates velocities and IMU, which estimates orientation.

4.2.1 Theoretical functioning

An assumption was made, as follows: that the robot can only rotate around the Z axis, so that it will remain parallel to the water surface. The following formulae were used to estimate its position:

$$\hat{X}_t = \hat{X}_{t-1} - X_{dvl;t} * dt * cos(\theta) + Y_{dvl;t} * dt * sin(\theta) \tag{4.2}$$

$$\hat{Y}_t = \hat{Y}_{t-1} - X_{dvl;t} * dt * sin(\theta) + Y_{dvl;t} * dt * cos(\theta) \tag{4.3}$$

$$\hat{Z}_t = \hat{Z}_{t-1} - Z_{dvl;t} * dt \tag{4.4}$$

Where $\hat{X}_t$, $\hat{Y}_t$, $\hat{Z}_t$ are the estimated position at time t, respectively along x, y and z axis. $\hat{X}_{dvl;t}$, $\hat{Y}_{dvl;t}$ and $\hat{Z}_{dvl;t}$ are the measurement from the DVL along each axis. θ corresponds to the yaw and dt is the time difference between two DVL measurement.

The problem observed while using the above formulas, was that when the DVL is not positioned at the inertial center point of the robot, the estimation is not correct, as seen in Figure 4.6. Subsequently, it was found that by using the following formulas, and by knowing e_x and e_y the distances between the DVL and the inertial center point of the robot in x and y axis, the trajectory corresponded.

$$\hat{X}_t = \hat{X}_{t-1} - [X_{dvl;t} - e_x \dot{*} \theta] * dt * cos(\theta) + [Y_{dvl;t} - e_y \dot{*} \theta] * dt * sin(\theta) \tag{4.5}$$

$$\hat{Y}_t = \hat{Y}_{t-1} - [X_{dvl;t} - e_x \dot{*} \theta] * dt * sin(\theta) + [Y_{dvl;t} - e_y \dot{*} \theta] * dt * cos(\theta) \tag{4.6}$$

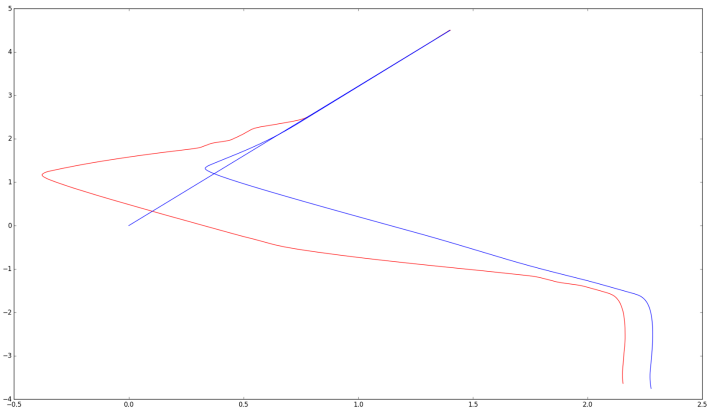


FIGURE 4.6: The blue trajectory corresponds to the real one, whereas the red one corresponds to the trajectory estimated with a DVL positioned at a position shifted from 0.75m in the Y direction.

4.2.2 Implementation

The most efficient way to achieve the equations explained previously, was to create a node that could be launched at the beginning of the simulation. The node, when launched subscribed to the DVL and the IMU and computed equations, and then published a Topic called $/odom$ which serves as Dead reckoning. Furthermore, another node that publishes a transformed frame of the current odometry, was created. The tf frame could then be used, for the SLAM and the sonar buffer, at a later stage.

By implementing this, an allowance was made for the robot to have an odometry topic, which provides a make-believe position for the robot in a certain position and then permits a possible re-localization of the robot using a SLAM. With the implementation almost done, the addition of some noise to simulate the BlueRov2 had to be done to complete it.

4.2.3 Gaussian Noise on measurement

The dead reckoning topic that was done by the DVL and IMU data-fusion is now working. The retrieved data can be considered as perfect, as there is no error added from the simulator for now. However, in reality, the DVL and the IMU have problems of drifting measurements. Therefore, a way to simulate this noise in the UUV simulation is required

Gaussian noise can be added to the measurement by auditing a line of code in the URDF description of the sensors with the UUV simulator. To emulate real noise, it is important to know first the typical errors that are generated by these sensors. After a discussion with my supervisor, it was agreed that the DVL usually has an average error of 2mm/s. Hence, a Gaussian

noise with a mean value = 0.2 and a variance = 0.2 was to the relevant lines in the URDF of the DVL. However, the Gaussian noise remains an approximation.

The IMU has an error of 3 ° max. This will be simulated with a Gaussian noise with a mean value = 0 and a variance = 0,0523599, which correspond to 3 ° in radiant domain.

4.2.4 Results

To approve the system, two different exercises were conducted and the results recorded. The robot was moved during 250s-350s while recording its position provided by the dead reckoning and the ground truth at the frequency of 2Hz. From these values, the error of every co-ordinate will be retrieved and the RMS error of the two scenarios will be computed. The aim here is, to show the difference between the system when operating in the total absence of noise vs when operating with Gaussian noise (defined earlier and added in the DVL, and the IMU).

4.2.4.1 Perfect sensors

During this test, the noise of the sensor was totally absent. Hence, a recording of each, the 'no-noise' and the 'noisy' scenario, was made and the results mapped in the graphs provided below to demonstrate the differences between both scenarios. The curves shown in the graph were made using Matlab plot functions. Only the position plot and the RMS is being discussed here, as it is sufficient to draw a conclusion with it. However, the CSV files of every test conducted are available on the Github.

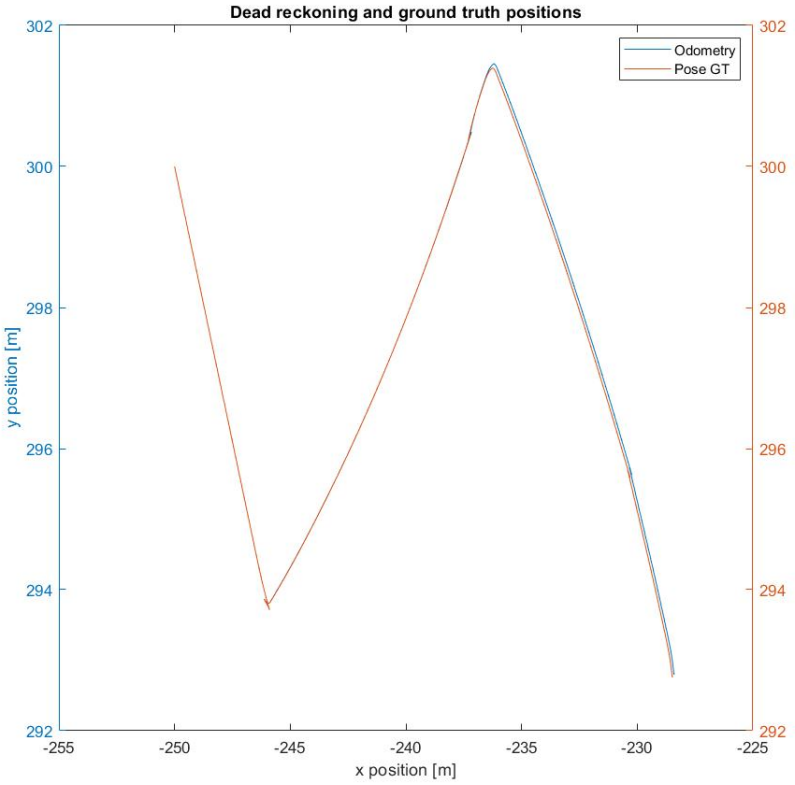


FIGURE 4.7: Dead Reckoning and ground truth position during the ”no noise” scenario

Figure 4.7 shows the path followed by the dead reckoning and the ground truth(GT). The dead reckoning is in blue and the GT pose in orange. The graph shows that these curves are exactly juxtaposed at the beginning of the simulation at the position [-250 ; 300]. However, the longer the simulation lasts, the wider the drifting grows. However the error must come from the equations used or the kinematic model which is simplified for the SLAM.

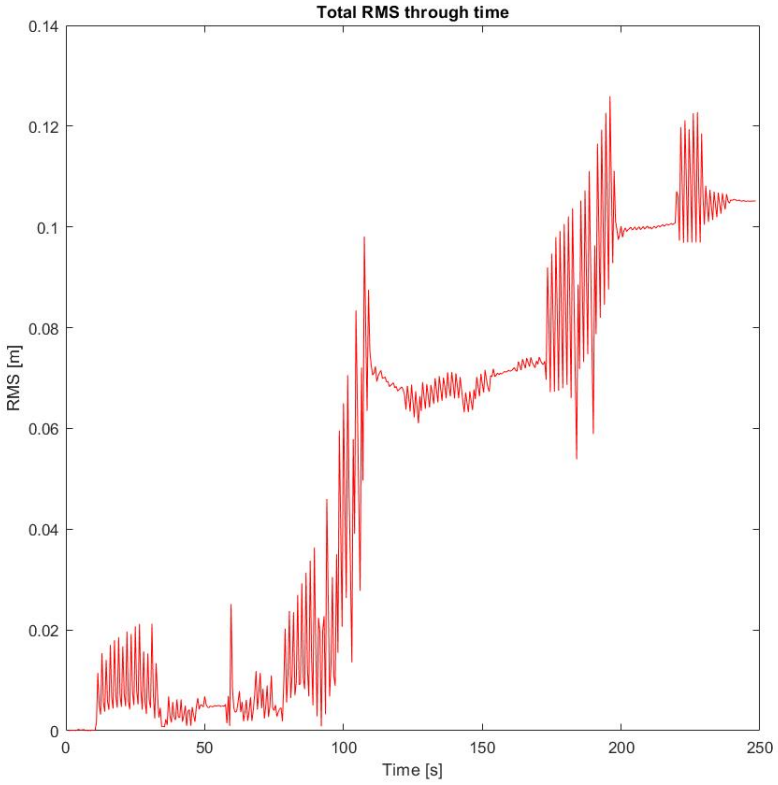


FIGURE 4.8: Evolution of the RMS during the "no noise" scenario

Figure 4.8 represents the RMS error of the system. The rise of the RMS error can be explained by the time rate of the sensors. Between two measurements of the same sensors, a certain amount of time passes, and during this amount of time, the measurement is not aligned to the ground truth. This explains every high peak in the RMS error. However the average of the RMS error is about 0.08 which is low but not perfect.

4.2.4.2 Typical noise added

As mentioned previously, a Gaussian noise was added to the DVL and the IMU. The subsequent test performed informed us of the error emulated by UUV simulator and its validity. The following graphics were drawn from the noisy measurement. The noise implementation of this scenario is described in the *implementation* part of this prototype.

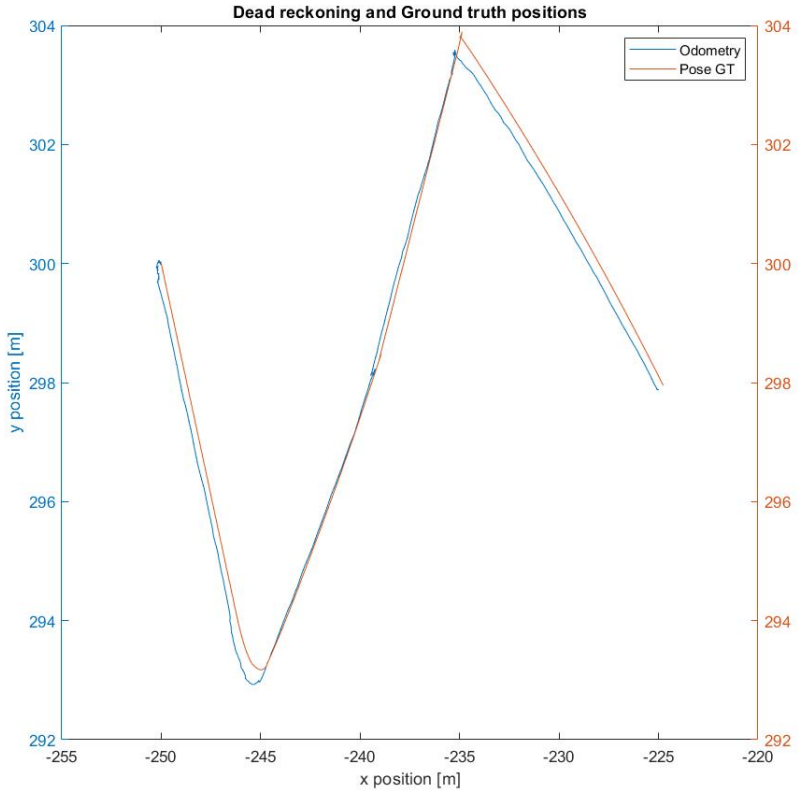


FIGURE 4.9: Dead Reckoning and ground truth position during the ”noisy” scenario

Figure 4.9 shows the path followed by the dead reckoning and the ground truth. The dead reckoning is in blue, while the pose GT is in orange. The path drawn by the Dead Reckoning is slightly drifted compared to the previous graph. The noise being Gaussian can increase the error or reduce it. However, the error reduction is only a local phenomena that happens due to the coincidence of distance between the noise and the real position.

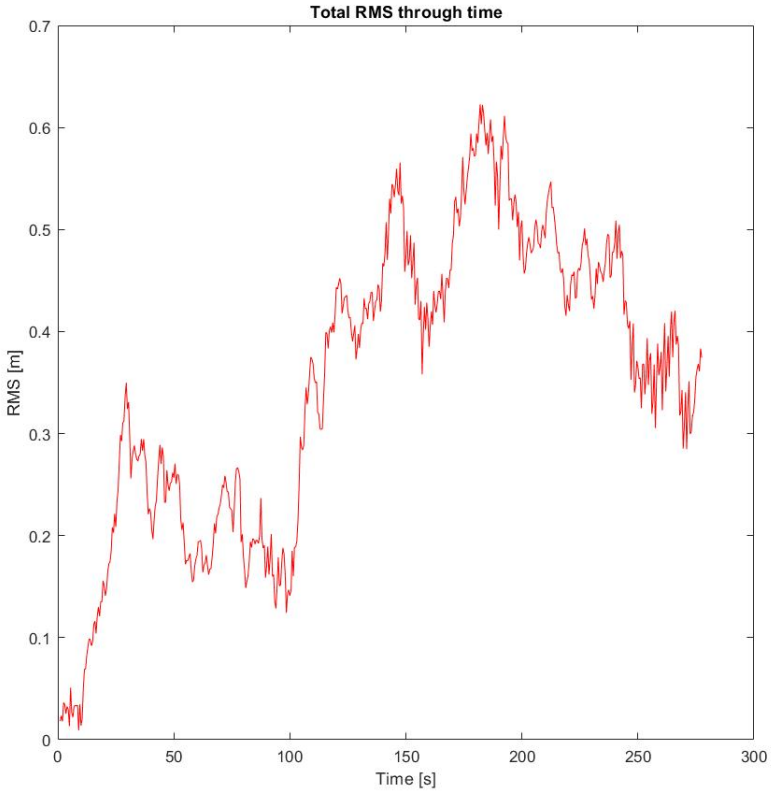


FIGURE 4.10: Evolution of the RMS during the ”noisy” scenario

The figure 4.10 represents the RMS error of the system. The RMS is a good metric as it allows us to see (in more detail) the average error of the system through time. Compared to the previous one, the biggest error reached is at 0.6m, which is approximately 7.5 times bigger than the ”no noise” scenario. The graph proves that the drifting of the sensors has a general tendency to grow even if it can decrease locally sometimes.

The results given by these scenarios allowed us to affirm that the Gaussian noise has a significant impact on the measurement and can be approximated to real measurements achieved on the real robot.

The robot model has now been defined and was implemented with dead reckoning topics, drifting through time, enabled. Before implementing the localization process, the method by which we will get and treat data from the sonar needs to be defined.

4.3 Prototype 3 : Segmentation & data buffer

In this section, segmentation and the data buffer used before the localization will be presented. As mentioned in Chapter 3, P3, the aim here is to have a reliable and continuous sonar image. The image sonar is, not only, one of the best solutions for an underwater environment reckoning, but it is also well known for its noise measurement capabilities. However, before processing and registering any data the information needs to be filtered.

4.3.1 Threshold & Filter

While, the sonar image is well-known for its noise capabilities, we cannot use this initial raw data from the sonar. The raw data needs to be filtered by techniques such as Threshold and Filtering outlined in "Underwater SLAM for Structured Environments Using an Imaging Sonar" by David Ribas et al [19]. First, the intensity of the points retrieved need to be thresholded. Therefore, every point with an intensity below a certain value was considered as unusable. Then, the highest intensity of every line of the 360-degree scan was taken to ensure that every radial line would be empty or have only one value remaining.

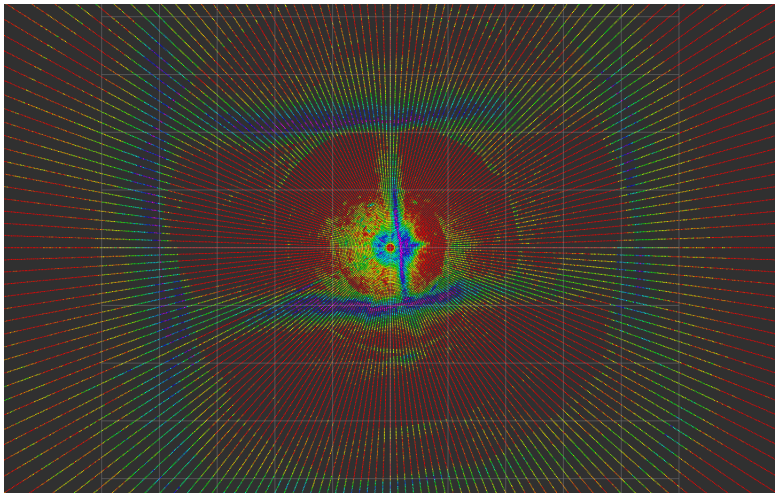


FIGURE 4.11: Image raw of the sonar

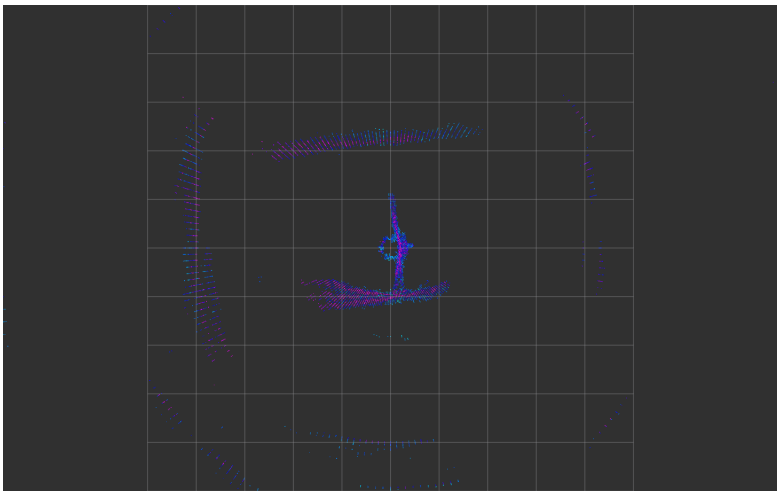


FIGURE 4.12: Sonar image with the threshold

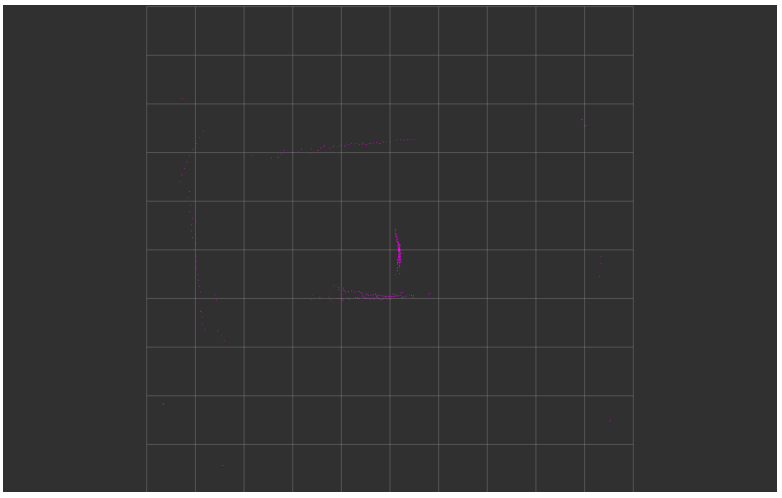


FIGURE 4.13: Sonar image max value

Calibrating the value of the threshold is an important task. This value will determine what will be taken into account or not. The value used here was determined empirically with the sonar test conducted in the ORCA water tank, and which can be changed for a real scenario, if and when needed. What could also be improved here is to create an algorithm that changes the threshold value according to the environment. With the threshold and filtering part completed, the data buffer implementation is next.

4.3.2 Data Buffer: Implementation

The Data buffer is a class that is called for at the beginning of the simulation. The data buffer will not deal with the distortion because of the absence of accurate sonar simulator. There are two buffer object :1) the mapping node and 2) the buffer node and they possess different features which will be explain here :

Scan: A full scan takes 12 seconds. Hence, the buffer is required to retrieve all the data provided by the sonar during this amount of time.

Clear: This function is used to clear the scan from the points taken. It can also be used after the re-localization to empty the buffer and reprocess the ICP with new data.

Update: Is a function that takes input from pointclouds(PC) and adds this data to the ones that are already in the buffer. This feature is essential for mapping buffer to allow the map to grow and to add new points to be matched later.

Mapping and the buffer are defined and called in their own python files and then communicated by messaging to the main program. The message received is a Boolean value but the mapping and the buffer nodes, each use it in a different way.

Mapping node
$True$: initialize the map
$False$: update the map

Buffer node:
$True$: Process the scan
$False$: Empty the scan

Initially, an attempt to call buffers in the main program was made to have full control. However, the main node could not handle two buffers at a time and hence the second one called was slower compared to the previous one. Hence, the reason why they are independent.

4.3.2.1 Data treatment

Prior to sending the data into the ICP, a different treatment on the current retrieved values was used. First, the duplication was removed: every point that was at a distance $<$ 0.1m from another point. Then, the points that remained were sampled to generate a new pointcloud with 2x less points in it. The idea is if the number of points are limited, the easier the ICP process is. The fewer and more accurate the points, the faster the convergence takes place.

4.4 Prototype 4 : Mapping

The mapping part is a crucial part of the SLAM. It enables the robot to autonomously localise itself by constructing a consistent 2D map, as it moves. In real-life and in the UUV simulation,

mapping has to factor in many aspects and adjustments in order to enable the robot model to generate its own navigation path, and perform localization using the prior map as reference with a feature-based or a scan-matching method of localisation.

Localization architecture had to be changed several times during the project to make it compatible with mapping system. The idea of creating a map manually was discussed. However, as the localisation architecture took quite a lot of time, the project could not proceed without this compromising the timeline and deadline for the SLAM project. Consequentially, and as mentioned in chapter 3, P4, the focus was the deliverable, which was to provide a Posterior Map in 2D.

4.4.1 Octomap

Octomap is a library that generates both a 2D and 3D occupancy grid. A ROS package was implemented to allow its use with the middleware.

The Octomap library uses octree implementations which allows the world to be fragmented into different sizes of voxels according to resolution. Octomap can generate a 2D and a 3D map without prior assumptions or position inputs, compared to other mappers such as gmapping. It can discretize the world into 2 types of boxes: the free and the occupied ones.

Underwater robots have to deal with large areas, that can be covered by Octomap. Therefore, the Octomap feature that provides the flexibility to set the resolution is very useful and appreciated. Furthermore, the map generated by this library is well-compressed, which is another important feature, that helps not only when one needs to embed the map, but is useful in our case, where the computational power of the processor of an ROV is quite restricted.

Octomap is multi-modal, it enables map creation using different sensors. It uses the 3D Pointcloud approach developed in these papers[39][40] The use of different sensors could help to add the cameras in the SLAM process in the future. Pointcloud approach is interesting for scaling large areas, but does not provide a model for unknown and free areas. Also, pointcloud requires a sensor treatment to remove noisy data before mapping.

4.4.2 Mapping Architecture & Process

The Octomap server node is very simple to use. It only demands three inputs to be initialized, which are :

- ID frame that will be used as reference for the map.

- Resolution of the map [m]
- The Max size of the map [m]

As this project has a large environment to map, the resolution and size of the map needs to be fixed with appropriate values. Hence, the resolution was set between 1m and 2m, as well as the max size at 1Km.

These values are easily changeable, and a user can find and change them in the main launch of the package. To ensure the validity of the mapping, a frame coordinate will be anchored at the position of the robot at the instant that the user wants to initialize the map. To do so, a frame is created at the beginning of the simulation, and will follow the odometry. But once the map initialization order has been sent, it will need to stop taking the odometry into account and remain at the same position in space.

The frame spawned after the initialization map command was tried, but it created errors in the frame. ROS prefers the frame to exist from the beginning of the simulation and can be anchored afterwards as shown in the figure 4.14, instead of making it spawn in the middle of the simulation.

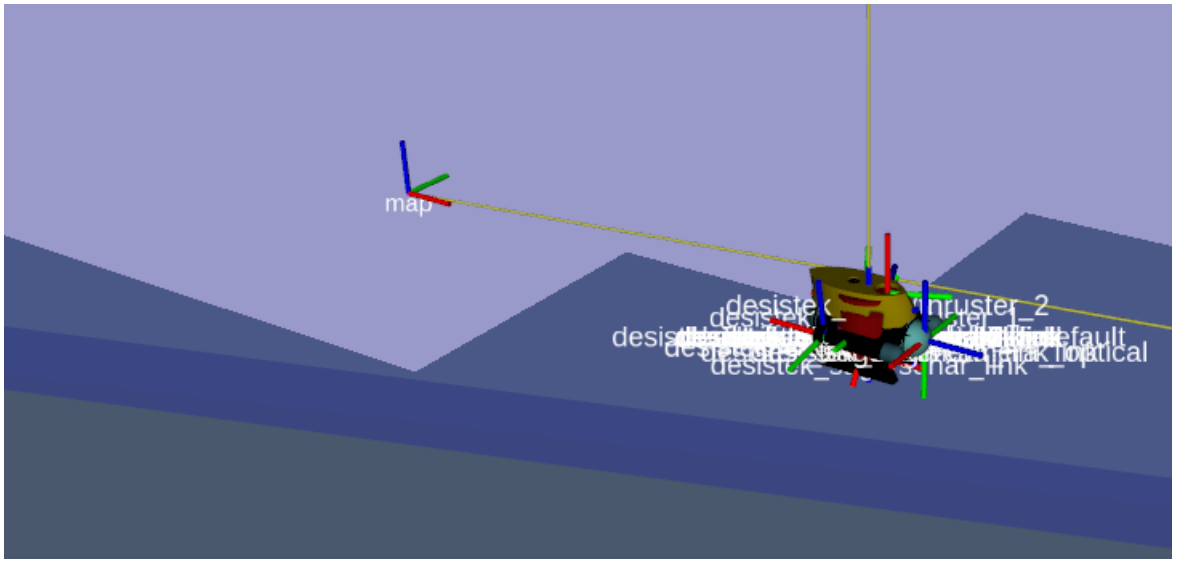


FIGURE 4.14: Map frame anchored after initialization

After the frame ID had been created, and the other parameters were set, the map using the pointclouds of the sonar could be generated using Octomap.

The first scan's frame id is assigned as "map", and the output from pointcloud can be directly added to the map.After the first map is achieved, the program needs to update the map, and here the frame referencing, plays a very important part. The next scan will also have "map" for its frame reference, and so forth.

For some reason, the ICP had issues dealing with points that have different frame ids. The process explained took place in the $tf\_map.py$ file in the $init\_robot$ folder.

4.4.3 Results

Unfortunately, owing to the Covid-19 situation, both these methods had to be evaluated separately: - The Filter test was done using a Rosbag record, in April 2020 in the water tank, and - The data buffer using UUV simulation and Gazebo was done in June 2020. The ideal solution would have been to test the mapping directly on the robot submerged in the water tank.

4.4.3.1 Micron Sonar

To ensure that the mapping package would work with the real robot, it had to be tested with the Micron Sonar. A rosbag was used during the tests, in the situation described previously in figure 4.17. As the sonar was not added at this point to the BlueROV2, we held it under the water by its end, and so consequently, the tests could only occur in static.
During the test with the micron sonar, the output of the sonar was sent inside a pointcloud message. The message was declined several times, in order to understand and treat it. The first modification was to transform the intensity into a z axis coordinate. This method provided a better understanding of the functionality of the sonar.

Another facet of the micron sonar test, was as shown in figure 4.15, where the echo was high, as the water tank was relatively small. Intensity of the echo was used as a main reference to threshold and filter most of the important points.

An assumption was made- that the points in the results from the scan were the ones with the highest probability to find an obstacle. And the output of the filter was then used for Octomap, which produced the map in figure 4.16. It can be noticed that the scan is not perfect and some outliers remain after the filtering. The process could be improved later to reject the aberration points.

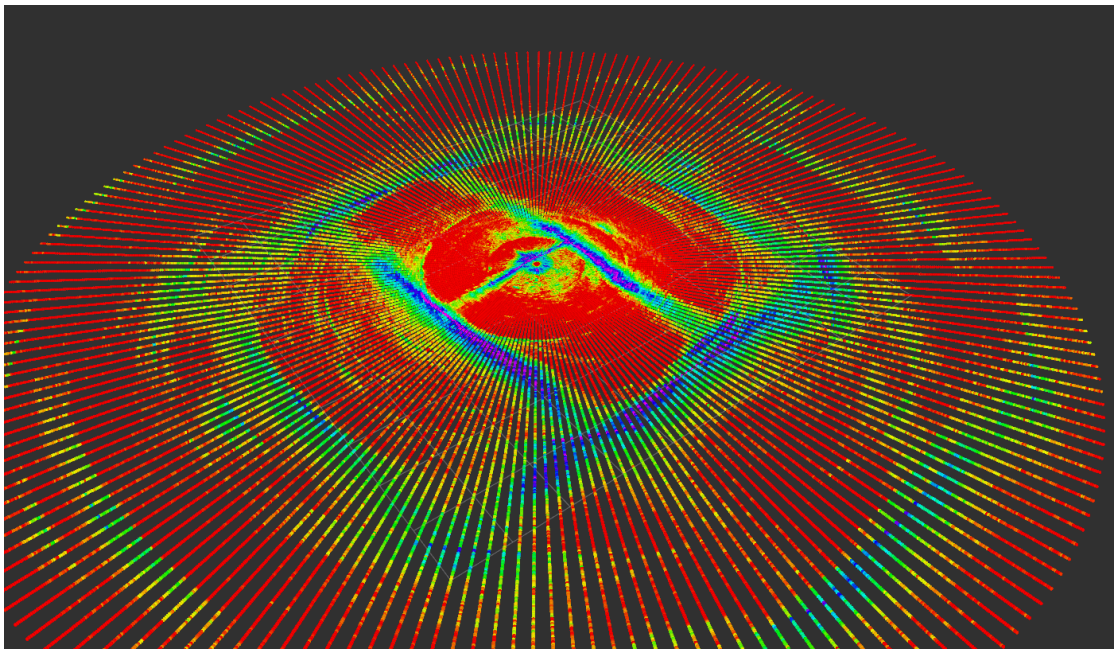


FIGURE 4.15: Output of the sonar in the water tank.

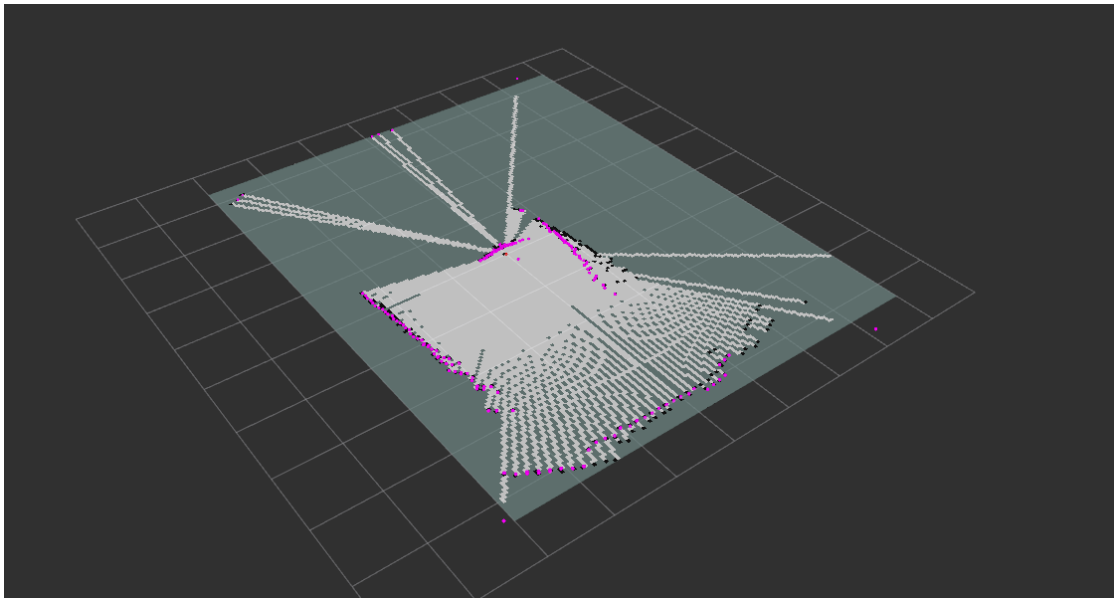


FIGURE 4.16: Result of octomap on the water tank with the micron sonar.

4.4.3.2 UUV simulation

In UUV simulation the case was much simpler than the rosbag scenario. The sonar becomes a Laserscan message which is completely different from the sonar image.

To validate the SLAM, a search was done for a compatible sonar plugin to be used in the UUV simulator but the search was unsuccessful.

The figure 4.17 shows the end-result of the data buffer process. However, the following steps were executed to reach the given result. The data buffer first retrieved data from the sonar and stored it during a predetermined period of time to allow the sonar to complete a full circle. Once this was done, the buffer used different functions to decrease the number of points such as, removing duplicated points and sampling. The results were achieved using these two aforementioned functions.

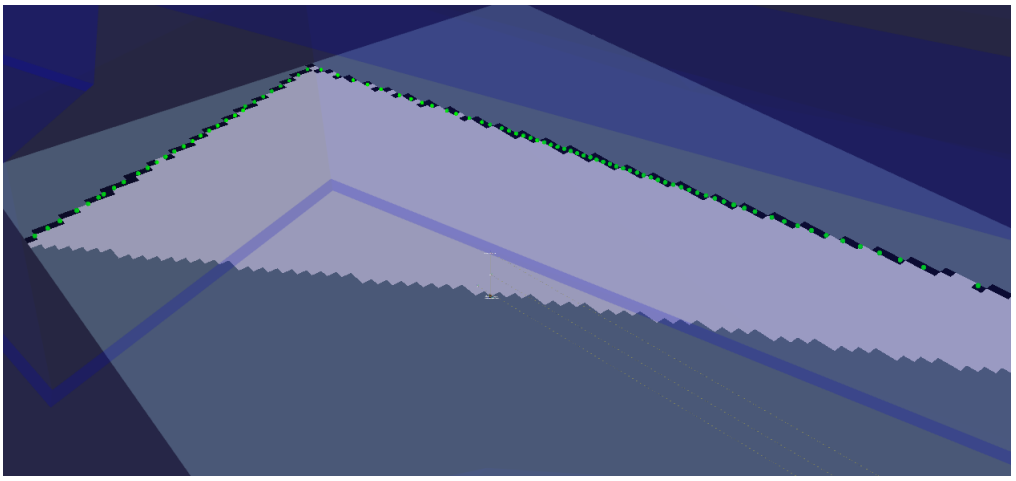


FIGURE 4.17: Picture of the map after initialization

It was noted that, the complicated part of this process was not so much the initialization of the map but making the map grow as per the buffers and the localisation data that was provided. To allow the map to extend and grow, a localisation method was required. This would help the robot to execute the re-localization process as well as extend the map accordingly. Unfortunately, this still remains a challenge, as mentioned in section 4.1.4

4.5 Prototype 5 : Scan-matching and ICP

As mentioned in Chapter 3, this prototype was supposed to entail testing feature-extraction such as SURF or Hough Transform. However, the following alternative was pursued.

There are multiple ways to perform SLAM. However, the two common methods that will be considered, here are 1) the Feature-based SLAM and 2) the Scan-matching based SLAM.

Feature-based SLAM consists of generating landmarks from interesting points in the environment, and those serve as an anchor in the surroundings, that the robot finds itself and thereby helps to re-localize itself.

Scan-matching based SLAM consists of overlapping multiple scans of the surroundings, taken at different instances of the robot's positioning and provides a self-correction of new position belief to the robot via its sensors.

Both methods use sensors to locate and re-locate the robot: While, the first one relies on specific landmarks acting as anchors, the second one uses all the data sets stored in the buffer to help the robot navigate its environment.

4.5.1 Scan-Matching

After discussion with my supervisor, the decision to utilise the scan-matching method and not the feature-based SLAM was taken. This is because, in the latter, the distortion and noise makes the generation of landmarks very unstable and in most of the cases the landmarks will need to be manually created [26]. Further, in order to focus on the more important parts of the SLAM process, instead of spending time creating landmarks, the feature-based method was put aside.

The most common method for scan-matching is the Iterative Closest Point (ICP). This method consists of iteratively finding the best Rotation and Translation of a two-point set that minimizes error. At the end, if the algorithm is converged to the "best solution", the program can retrieve the rotation and translation between the 2 scans.

As mentioned in the prototype 4.2, the project already has a position belief via IMU and DVL sensor. The idea is to use this belief as an initial guess for the ICP program and thus retrieve another position belief computed by the sonar, itself, this time.

While, the dead reckoning drifts through time, the ICP is always correct when it converges. However, the fusion of these position beliefs gives a good approximation of the ground truth position of the robot. The fusion of these beliefs can be achieved using EKF, UKF or PF methods.

4.5.2 Iterative Closest Points

The ICP method consists of overlapping two sets of points that correspond to two separate and partial views of the same object. Used mainly in domains that require 3D reconstruction, such as computer vision and Scan-matching for SLAM algorithms [41], ICP has fast evolved and widely used.

To explain, the calculus part of the general algorithm presented above: Say, there are two sets of points with unknown configurations:

Let's Q and P be theses two set of points:

$$Q = \{q_1, ..., q_N\} P = \{p_1, ..., p_M\} \tag{4.7}$$

Algorithm 3 Iterative_Closest_Points(q, p)

$error = inf$
while error Decreased & error $>$ Treshold **do**
$DataAssociation$
$Compute\ the\ center\ of\ mass(p, q)$
$Compute\ Rotation\ R\ and\ Translation\ T\ via\ SVD$
$Apply\ R\ \&\ T\ to\ the\ source\ points$
$error = E(R, T)$
end while
return R, T

$with\ correspondence : C = \{(i, j)\}$

The final aim is to find the Rotation R and Translation T that minimize the following error function:

$$E(R,T) = \sum_{(i,j)\in C} \|q_i - Rp_j - T\| \tag{4.8}$$

To do so, the algorithm needs to proceed to the data association stage. The data association will help the associate points of the two set of points between them.

4.5.2.1 Data Associations

Data association is another important part of this prototype 5. If the data association is known, the problem is half solved. However, some computational processes need to be done before, in order to have the best data associations. There are many ways to refine this process. For example, a weighting system could be considered, where every point is given a weight that will help with data association. Another thing that could be implemented is a rejecting outlier.

Then, the ICP will compute the center of mass. The center of mass will allow the ICP to find the translation between both set of points.

4.5.2.2 Center of mass

The center of mass of both set points has to be computed using the following formulas.

$$\mu_Q = \frac{1}{|C|}\sum q_i \ \ \mu_P = \frac{1}{|C|}\sum p_i \tag{4.9}$$

Then, every point is subtracted by the center of mass of its set of points. By doing this the set of points normalize around the same origin.

$$Q' = \{q_i - \mu_Q\} = \{q'_i\} \tag{4.10}$$

$$P' = \{p_i - \mu_P\} = \{p'_j\} \tag{4.11}$$

Error can be re-written using the new set of points:

$$E'(R) = \|[q'_1, ...q'_n] - R[p'_1, ...p'_n]\|_F^2 \tag{4.12}$$

This problem is called Orthogonal prosecute problem and can be solved using the Singular Decomposition Value.

4.5.2.3 Singular Value Decomposition

Firstly, the Cross-Covariance matrix needs to be computed:

$$W = \sum_{(i,j)\in C} q'_i {p'_j}^T \tag{4.13}$$

Then the SDV is used to decompose this matrix:

$$W = UDV^T \tag{4.14}$$

where U and V are two 3x3 matrices corresponding to the rotation matrice.

D is a diagonal matrice. If the rank(W) = 3 then it exist an unique configuration of R and T that minimize the error given by:

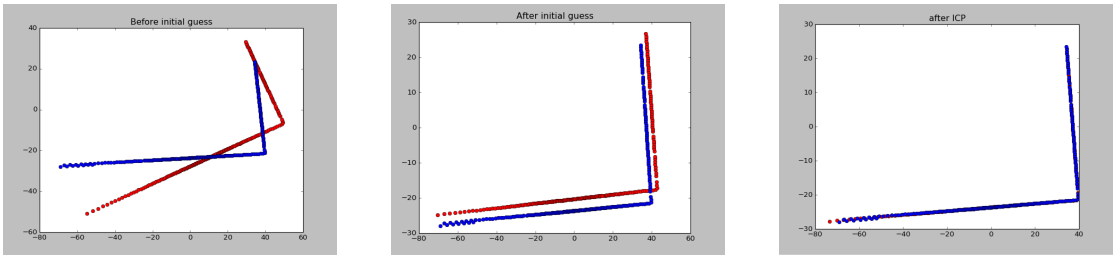
$$R = UV^T \tag{4.15}$$

$$T = \mu_Q - R\mu_P \tag{4.16}$$

Finally, the source point set is transposed with the new Rotation and Translation and recompute the new error function:

$$p_j \leftarrow R(p_j - \mu_P) + \mu_Q \tag{4.17}$$

In figure 4.18 an example of the ICP working on the *Desisteck saga* can be seen. First, a scan of the area was performed, then the robot moves and finally computes another scan. The algorithm can be split into 3 different steps: Before the initial Guess; After the initial guess and After the ICP. The initial guess is achieved by calculating the differences in the odometer of the two scans.



(A) scans before the initial guess (B) scans after initial guess (C) scans after ICP

FIGURE 4.18: Step of the ICP algorithm

4.5.2.4 Implementation

The mathematical formula and process of the ICP has been explained in the previous section. The ICP was taken from this website [42] and the ICP had to be studied to understand how it works before its useful features e.g how to plot the curves and how they could be implemented in Desistek Saga could be implemented. The ICP is a Python class that takes as arguments two set of points under the homogeneous format: nx3 (homogeneous) with n the number of points. Points are under the homogeneous format to easy the matrix multiplication in the ICP. The ICP also took the initial guess under a 3x3 matrix format that corresponds to the transformation matrix from the azimuth reference.

$$P_{source} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \dots & \dots & 1 \\ x_n & y_n & 1 \end{pmatrix} P_{target} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \dots & \dots & 1 \\ x_n & y_n & 1 \end{pmatrix}$$

$$\mathrm{T}_{initial} = \begin{pmatrix} Cos(\theta) & Sin(\theta) & x \\ -Sin(\theta) & Cos(\theta) & y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.18}$$

The output of the class will be the new Transformation T to pass from the source reference to the target and the Error associate E.

4.5.2.5 Limitations

The theoretical proof of the convergence of the ICP algorithm (and it's capacity to slowly converge toward a local minimum) was given by Besl & McKay. Its evidence relies on the hypothesis that the number of corresponding points is constant. This hypothesis remains complicated to prove for a real scenario because of the complexity of corresponding points between them. [43].

Another difficulty comes from the locality of the solution found. The transformation returned by the algorithm highly depends on the initial guess of the two sets of points. Therefore, a good initial guess is required to ensure the convergence of the ICP [44].

4.6 Prototype 6 : Re-Localization & Kalman filter

As stated in chapter 3, this prototype was to involve the creation of landmarks and the implementation of the particle filter. However, as explained in prototype 5, landmarks and the feature-extraction was put aside. Instead, an alternative re-localisation methodology was followed.

The output of the ICP gives the transformation between two sets of points, nevertheless these transformations are done using the map frame coordinates. The process needs to transform this matrix into world frame coordinates before sending it to the KF.

4.6.1 Frame Transformation

During the frame transformation process, points are transformed from one frame to another. To do so the SLAM process will use the matrix of rotation and translation. The matrix format is a very powerful tool, very efficient in terms of processing time because of its format. The motion model has only 3 degree of freedom (x,y,yaw). The problem can be modelled, as follows:

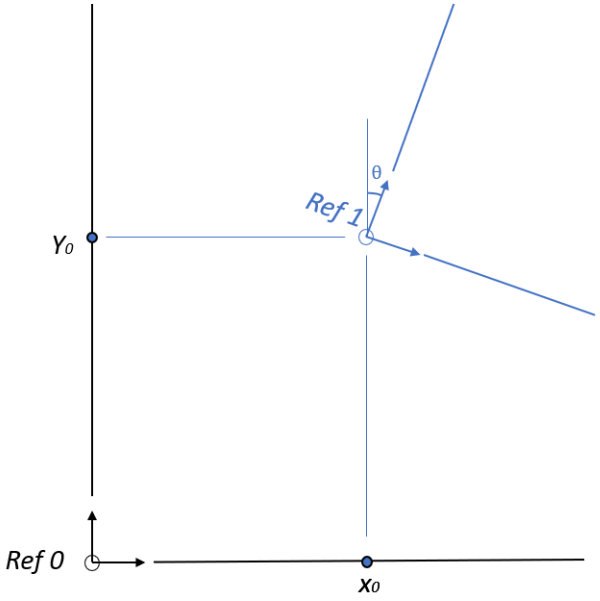


FIGURE 4.19: Frame transformation example

Therefore, the only rotation that is possible is on the z axis, and this rotation matrix is used:

$$R = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.19}$$

as well as this translation matrix:

$$t = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.20}$$

Finally, for matrix transform, we have:

$$T = t * R \tag{4.21}$$

$$T = \begin{pmatrix} cos(\theta) & sin(\theta) & x_0 \\ sin(\theta) & cos(\theta) & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.22}$$

The transformation matrix has been computed by the program and this allows for it to pass a coordinate from one point to another using a homogeneous format. Let P_{ref_1} be a point from

the $ref1$ coordinate frame and transform it into P_{ref_0} the same point in the $ref0$ coordinate frame. The equation between these two points will be :

$$P_{ref_0} = T * P_{ref_1} \tag{4.23}$$

$$\begin{pmatrix} x_{P0} \\ y_{P0} \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\theta) & sin(\theta) & x_0 \\ sin(\theta) & cos(\theta) & y_0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_{P1} \\ y_{P1} \\ 1 \end{pmatrix} \tag{4.24}$$

4.6.2 Kalman Filter

As mentioned in the *chapter 2*, the purpose of the Kalman filter is to compute and correct the belief with the observation. The simplicity of the Kalman filter is the Gaussian Assumption: both, the belief and the observation are assumed to be shaped in a Gaussian distribution with the mean value μ the center of the coordinate and an arbitrary variance σ that will represent the uncertainty. The following Matrices formats were chosen for the project:

Algorithm 4 Kalman_Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

$\overline{\mu}_t = A_t \mu_t + B_t u_{t-1}$
$\overline{\Sigma}_t = A_t \, \Sigma_{t-1} \, A_t^T + R_t$
$K_t = \overline{\Sigma}_t \, C_t^T \, (C_t \, \overline{\Sigma}_t \, C_t^T + Q_t)^{-1}$
$\mu_t = \overline{\mu}_t + K_t \, (z_t - C_t \overline{\mu}_t)$
$\Sigma_t = (I - K_t \, C_t) \, \overline{\Sigma}_t$
return μ_t, Σ_t

4.6.2.1 Motion model

$$x_t = A_t \mu_{t-1} + B_t u_t \tag{4.25}$$

$A_t =$ Matrix that describes how the state evolves.
$B_t =$ Matrix that describes how the control changes.
$u_t =$ The motion at the time t
$\mu_{t-1} =$ The previous mean value

For the kinematic model the matrix was very simple because the DVL already gives the belief under an (x,y,θ) form. Therefore our Matrix was a diagonal matrices 3x3:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mu_{xt-1} \\ \mu_{yt-1} \\ \mu_{\theta t-1} \end{pmatrix} \tag{4.26}$$

4.6.2.2 Observation model

$$z_t = C_t x_t \tag{4.27}$$

$C_t =$ Matrix that describes how to map a state to an observation.
$x_t =$ The observed motion at the time t

The ICP returns a Rotation and a translation between the first scan and the second one. It implies that the output of the ICP is not in the right frame reference. A transformation has to be processed to replace the position in the world frame reference and not the map.

$$x_{world} = T * x_{map} \tag{4.28}$$

$$\begin{pmatrix} x_{world} \\ y_{world} \\ 1 \end{pmatrix} = \begin{pmatrix} cos(-\theta) & sin(-\theta) & -x_0 \\ sin(-\theta) & cos(-\theta) & -y_0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_{map} \\ y_{map} \\ 1 \end{pmatrix} \tag{4.29}$$

This equation only works for the x and y axis. θ is completely independent from this process. However $theta$ has to be converted as well. To do so, the following equation will be used :

$$\theta_{world} = \theta_{map} - \alpha \tag{4.30}$$

α : is the angle of the odometry during the scan

By doing this:

$$\begin{pmatrix} z_{xt} \\ z_{yt} \\ z_{\theta t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{world} \\ y_{world} \\ \theta_{world} \end{pmatrix} \tag{4.31}$$

The Kalman Filter code can be found and changed in the $KF.py$ file in the ICP Folder of the Github.

4.6.3 Results

The Re-localization method plays an important part in the SLAM process. To test the Re-localisation that was created, two case scenarios were considered: In the first one, a Gaussian noise (mentioned previously) was added and its value doubled to make the divergence easier. After the test, it was noticed that while the effectiveness of the Re-localisation was evident, the noise was too unstable to really gauge any effect. Hence, to approve the localisation, a second scenario was created, wherein the error was made to drift linearly.

4.6.3.1 Gaussian noise

The first set to be performed was the Gaussian noise test to check on the effectiveness of the localisation process. The Gaussian noise is quite far from what a DVL and IMU error is, but still interesting to perform.

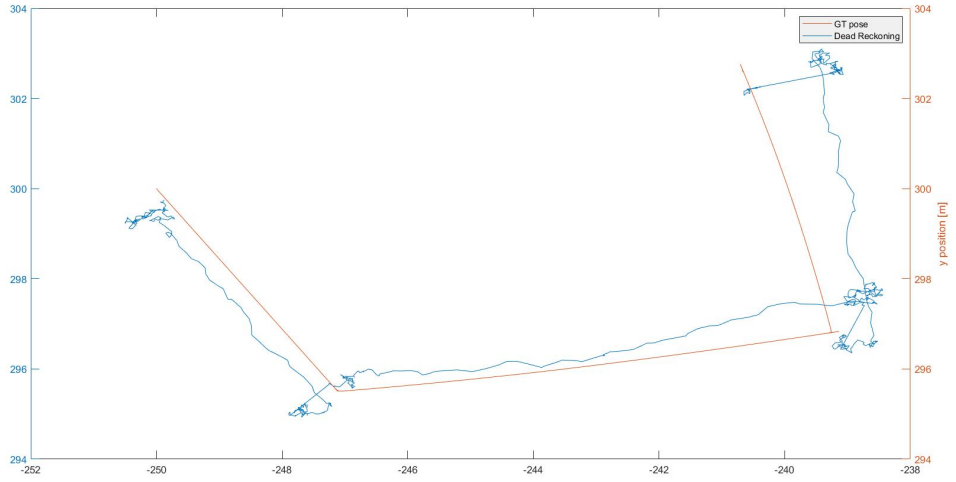


FIGURE 4.20: Dead reckoning and ground truth position during the test

The figure 4.20 represents the dead reckoning and the Ground truth during the process. The Gaussian noise has the particularity of not increasing the error. The error can grow, as much as it can reduce, which makes this improvement provided by the re-localisation complicated to extract. Hence, three re-localizations had to be completed during this test. They are represented by a straight line but are hardly visible owing to the uncertainty of the error.

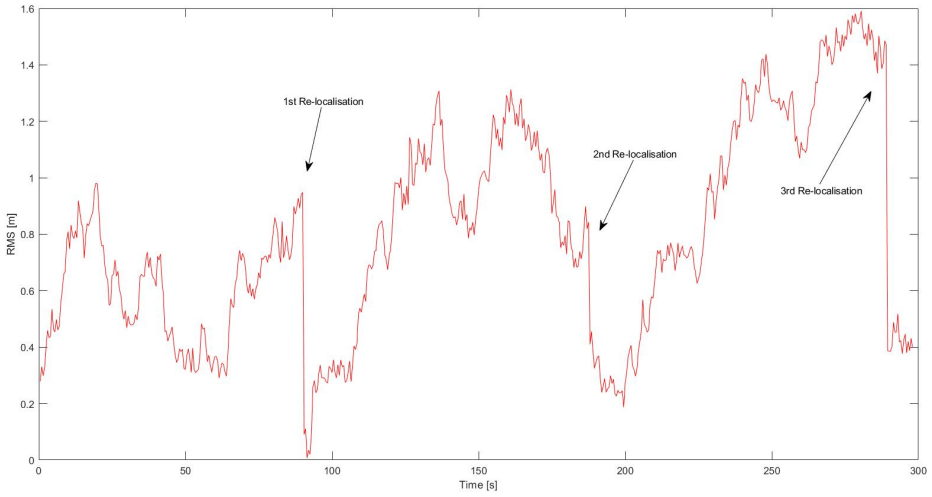


FIGURE 4.21: RMS evolution through time

The figure 4.21 represents the RMS error through time. The 3 re-localizations are easily readable on this graph (as indicated with black arrows). It can be noticed that the RMS is maintained upto 0.4m after re-localisation.

The error that occurs after re-localization has to be handled very carefully as the error has the tendency to rapidly change in a very short period of time. The time between the beginning of the scan and end of the Kalman Filter allowed the Dead reckoning to move from place to place, making the error after the KF remaining uncertain. The best way to see the efficiency of the localization is to induce a incremental drifting.

4.6.3.2 Linear noise

In this scenario an incremental noise was added in the DVL sensor for four re-localisation were processed. The error was set at 0,0625m/s on the x and y position. It was interesting to see how the localization method behaves with two types of errors and this process demonstrated a clearer picture of the drop of RMS after the KF.

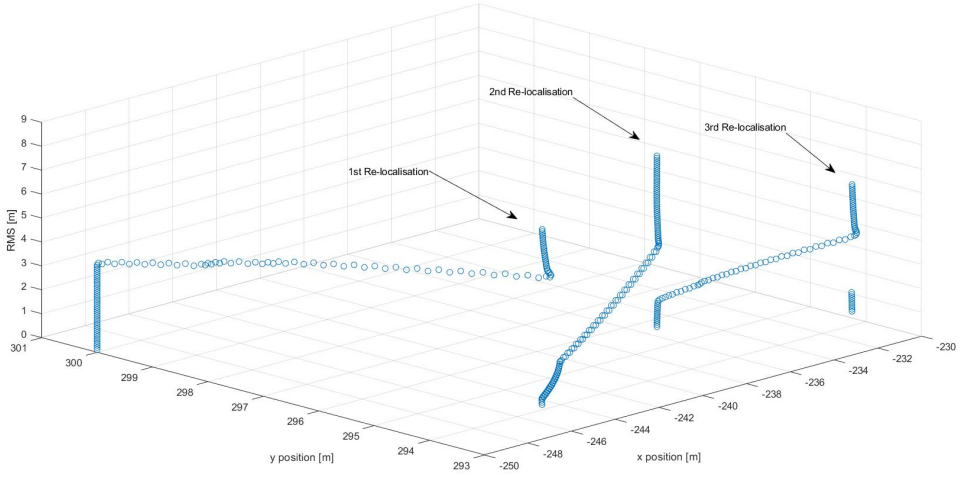


FIGURE 4.22: RMS evolution through the path draw by the robot

The figure 4.22 represents the evolution of the RMS according to the ground truth position of the robot. While, the classical graph of the ground truth and the dead reckoning was not clear enough to be visible, this representation was found to be clearer. The column of RMS is due to the rotation of the robot: when the robot is rotating the DLV continues to increase but the robot's position doesn't change. Once the position changes, the RMS continues and this rise is due to the wait for the robot to be stable before scanning plus the time to perform the ICP and KF.

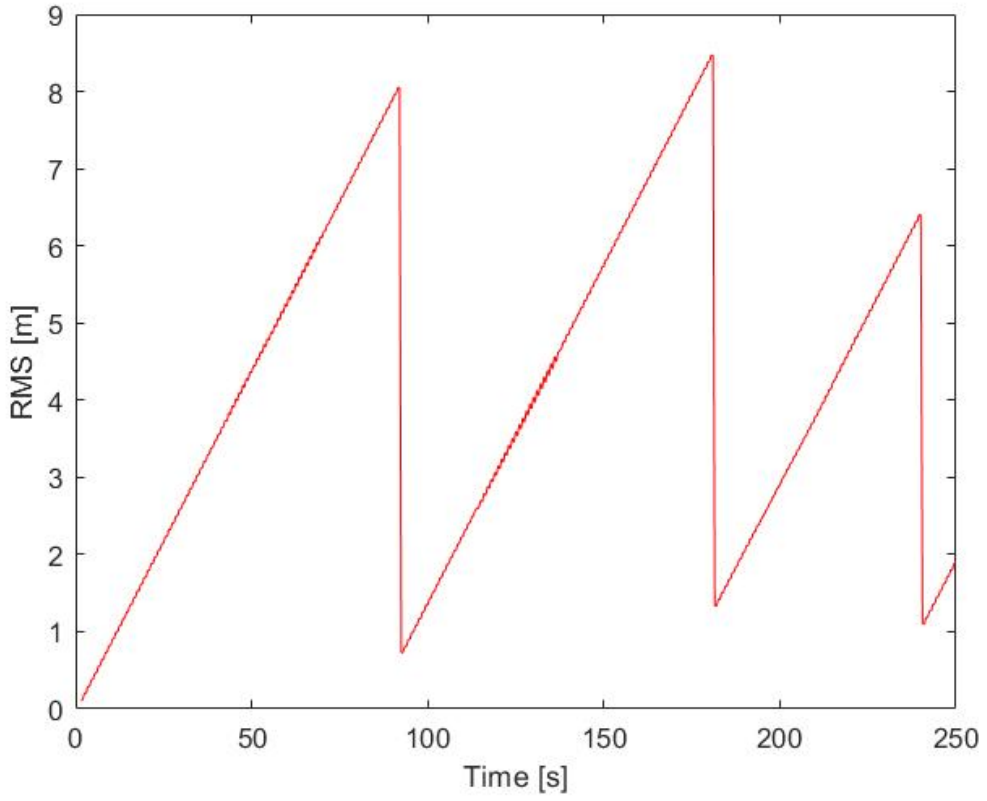


FIGURE 4.23: RMS evolution through time

The figure 4.23 represent the RMS of the system. In practice the RMS is reduced to approximately 0.5 meter. However, the time taken to process the ICP and the KF and the error had grown again, making the re-localization method not effective as it was.

4.6.3.3 Limitation

The Kalman Filter is limited to linear systems and most physical systems are not linear. Thus, the Filter remains optimal only on a short linear range described in the equations. The non-linearity can come from the observation model as well as the motion model. However, the reason why the KF presented gives good results, is linked to the fact that the the motion and observation model are close enough in behaviour where linear motion is concerned.

UKF and EKF could be an interesting improvement for the project in terms of localisation.

These 6 prototypes which were each implemented and tested in turn, were incorporated into a SLAM architecture, as they became available. The 7th and final prototype explains how a globol and coherent architecture was organised to ensure a smooth functioning of the SLAM process.

4.7 Prototype 7 : Simultaneous Localization and Mapping

Producing a sonar-based SLAM for an underwater ROV was the ultimate goal of this project and thesis. In this prototype 7, all the deliverables of the prototypes 1-6 come together in a common architecture to create a consistent and coherent map, that can pinpoint an accurate position during a AUV/UUV's robot subsea exploration.

This prototype 7 took more than one third of the time allotted to the project in Chapter 3, P7, which was projected to be 15 days. Once a prototype was completed and the deliverable for that respective prototype met, it was integrated into the SLAM architecture. This proved to be a useful process as it allowed for flexibility to change prototypes, when and if needed to make them compatible with the global SLAM process. However, it was not smooth sailing all the time. The structure of the architecture had to be changed many times. One time, in the middle of the project, it had to be completely re-written from scratch with a new and less ambitious goal. (As the ideal objectives mentioned in chapter 3, could not be done practically, owing to the constraints in place due to the pandemic). Ultimately, the SLAM is the heart of the project and the simplicity of utilization and the processing time was at the forefront from its conception to its implementation.

4.7.1 Global Architecture

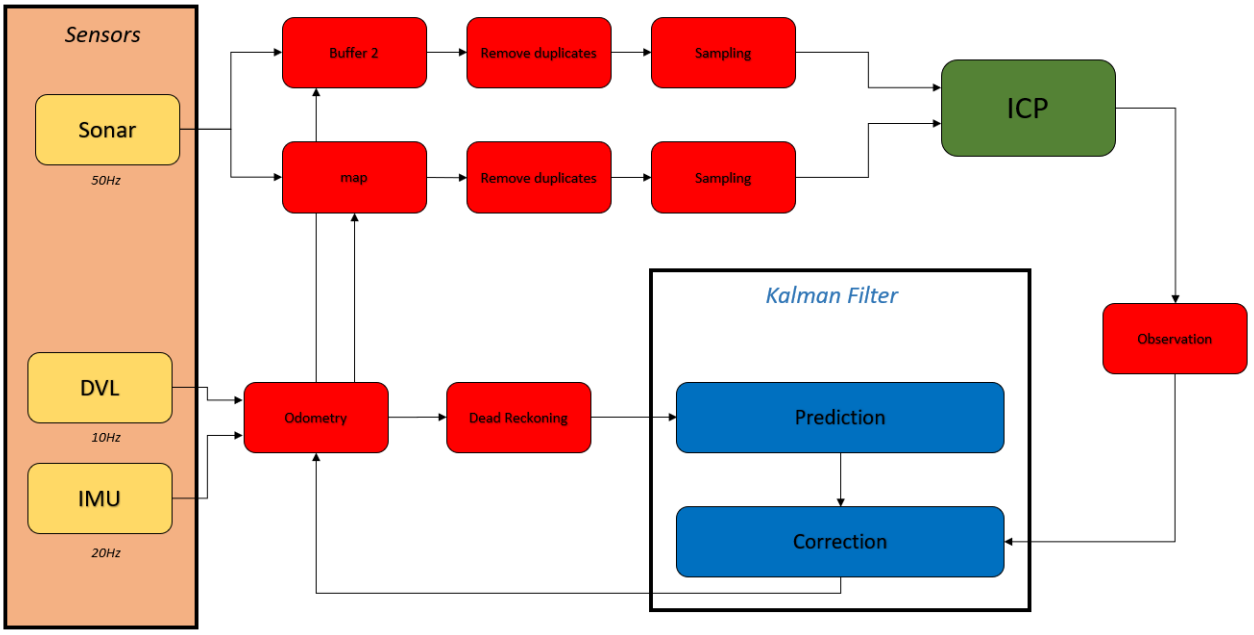


Figure 4.24: Architecture of the SLAM algorithm

In figure 4.24, an overview of the architecture of the SLAM process, is provided. The DVL and IMU generate the Dead reckoning that will serve as reference for the map and future scans. Once the map is initialised, the sonar scan will be treated by the buffer and compared to the map via the ICP. The corrected position will be provided by the ICP and compared to the believed position using a KF. The new position computed will correct the Dead reckoning.

Let's define in more detail the architecture functioning thought time.

4.7.1.1 Localization Process through time

figure 4.24 represents the working localization architecture through time-from the retrieving of data to the newly computed odometry.

In the following figure 4.25 the three important states of the robot are depicted: Start Simulation, SLAM initialization and First scan. For every state, data is stored and used for different functions. Boxes are either annotated with data with the frame reference in parenthesis or a function.

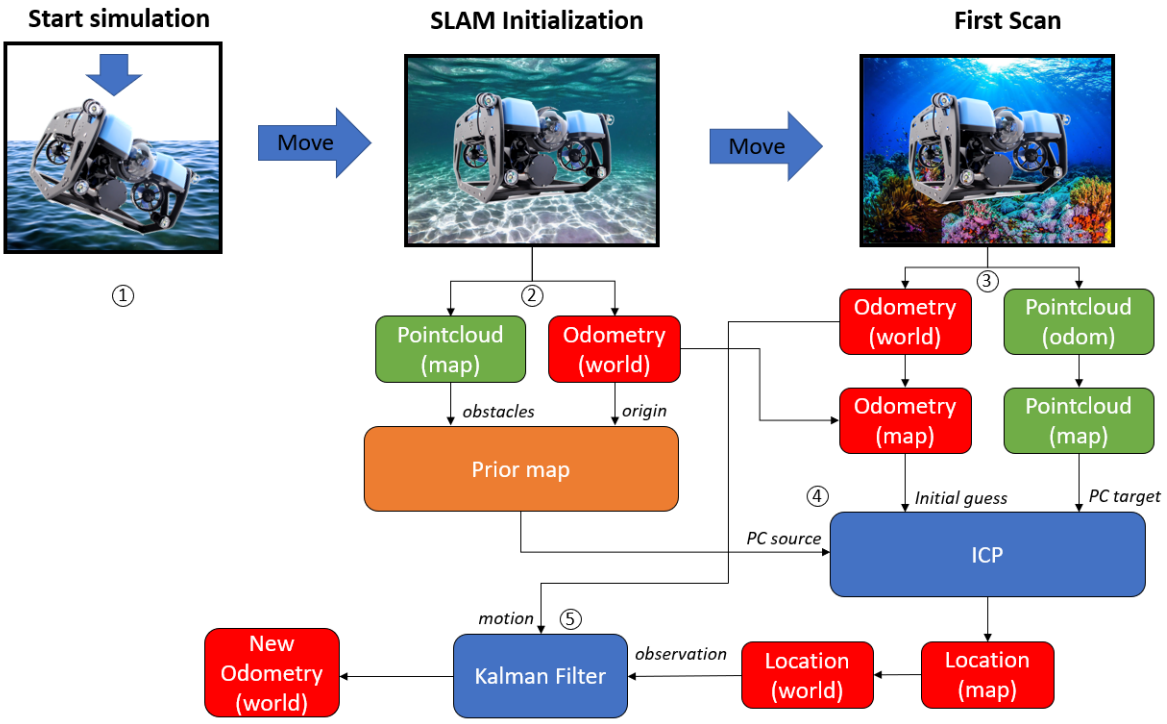


FIGURE 4.25: Architecture of the localisation through time

[Colour key: Red boxes are odometry messages, Green boxes are Pointcloud messages, orange boxes- the octomap node and blue one function used during the localization method.]

The localization method is detailed below, step by step, using the numbers that are referenced in the figure:

- 1 : The first step is the launching. The odometry is initialized and ”odometry” and ”map” Frames ID are defined and follows the vehicle dead reckoning.
- 2 : The SLAM initialization begins with the generation of the first map. The current odometry is gathered to be set as the origin of the map and the pointcloud retrieved is to detect obstacle of the map. Once the initialization is done, the map is constructed and defined. The robot needs to move to perform the re-localization and update the map.
- 3 : Once the robot moves the current odometry and the pointcould with the odometry as reference are retrieved. Then the origin of the map and the new odometry are used to compute the position of the robot. This new approximated position is used to transform the pointcloud from the odometry reference to a map reference.
- 4 : Having collected, all this information, the ICP process can begin its computational process, as follows: The odometry with a map frame as a reference will be the initial guess; The scan under pointcloud format with map frame reference as target and the map obstacle pointcloud as source; Once the ICP is done, if the error is too high, the program will ask if the results are suitable. If yes, then the ICP will provide the transformation matrix between the origin of the map of the true position of the robot in the map. Afterwards, the location will be added to the origin of the map to form the location in the world reference.
- 5 : Both the location and the odometry obtained during the scan will be used as an observation model and an motion model respectively in the Kalman Filter. The computed new position will be sent to the dead reckoning node that will drop the error by adding an offset to the current odometry.

4.7.1.2 Mapping Process through time

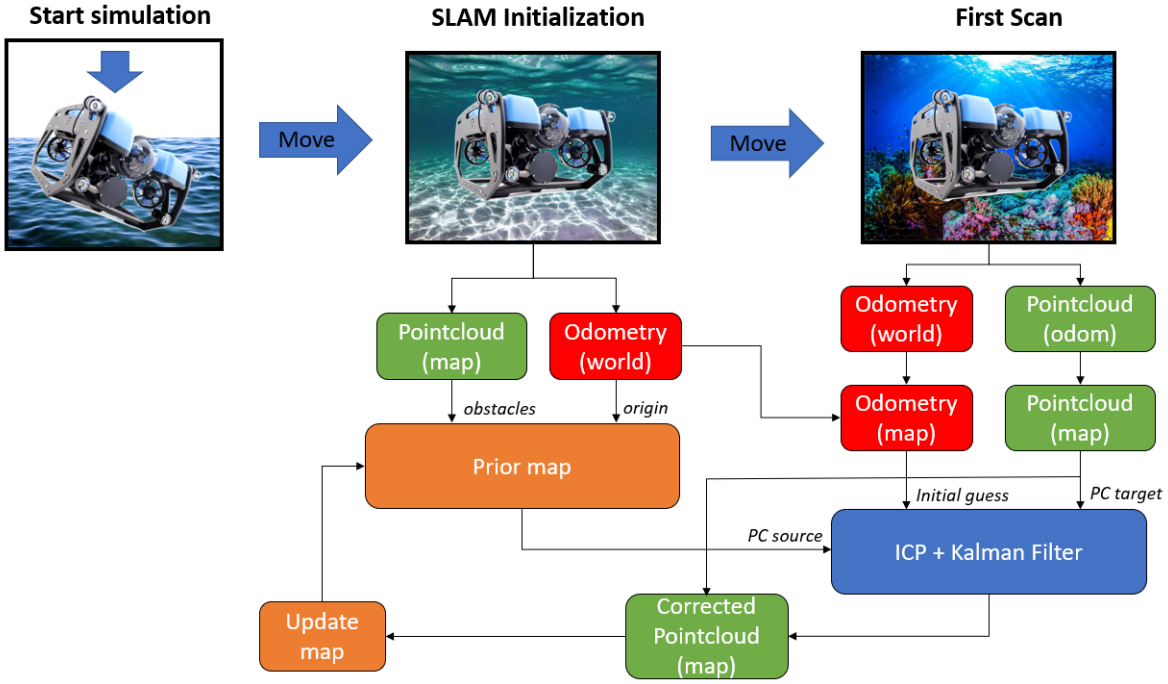


FIGURE 4.26: Architecture of the mapping through time

This process for the mapping is slightly different from the localization process. The process differs only after the ICP and Kalman filter. The re-localization was used to change the reference of the pointcloud from the scan and add it to the current map allowing it to be updated.

4.7.2 Implementation

The implementation of the global architecture using ROS components (nodes & topics) is detailed here.

The project was done using python for it simplicity of implementation and effectiveness. As said before the architecture changed many times, but only the latest version will be discussed, here.

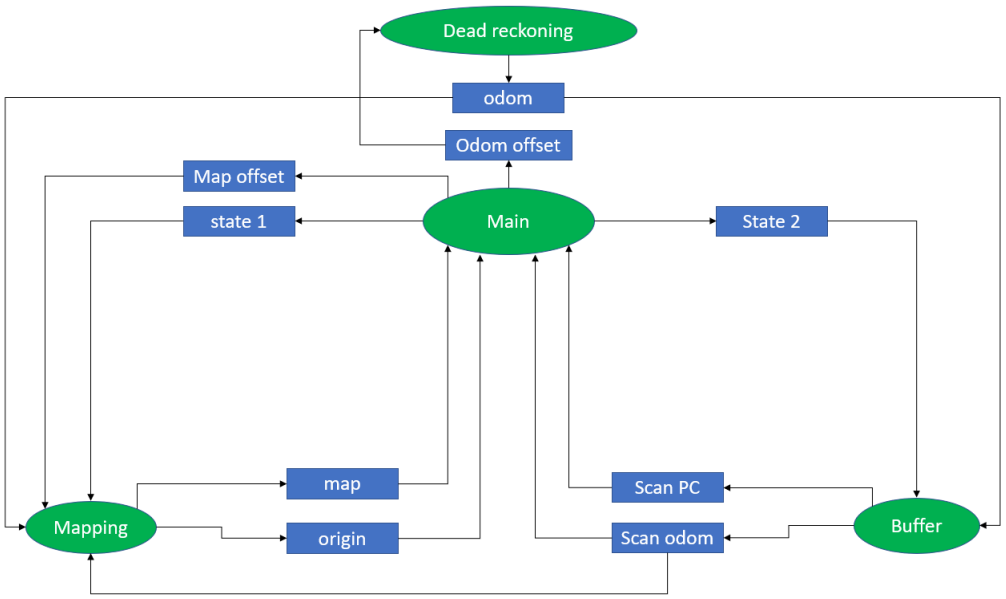


FIGURE 4.27: ROS Architecture : nodes and topics

Figure 4.27 represents the SLAM process through ROS components. The green-fill ovals represent the nodes and the blue-fill, the rectangles, the topics or the process was organised to have only 4 nodes, with each having a particular 'publishing function': the mapping nodes publish the topics, map, and origin- which are the pointcloud of the map and the origin in the world frame reference. the buffer node publishes in the topics "scan PC" and "scan odom" - which are the pointcloud of the scan after treatment and the odometry values of the associate position. The main node controls the mapping and buffer nodes by publishing in the topics "state 1" and "state 2" which command the node to : scan, clear the buffer or update the map. The main program contains the ICP by which it can correct the odometry using the topic "odom offset" and gives the transformation of the "scan PC" to update the map.

4.7.3 Results

The results have been achieved using a Gaussian noise as presented in the section 4.2.3. Two re-localisations were performed during the test. The aim was to show the effectiveness of the architecture built by performing localisation and mapping. The following graph 4.28 and 4.29 were done in UUV simulation in a maze created for the SLAM and the video of this test is available on Youtube

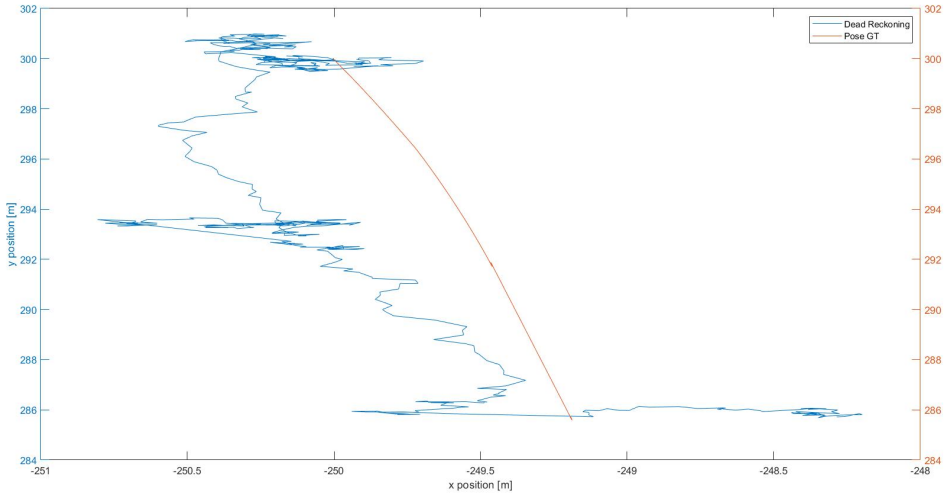


FIGURE 4.28: Dead Reckoning and pose GT paths through the process

Figure 4.28 shows the path drawn by the GT pose in orange and the dead reckoning in blue. This graph demonstrates the efficiency of the SLAM process. Two re-localisation can be seen in the graph. They are easily recognizable by the straight line that end-up close to the GT pose.

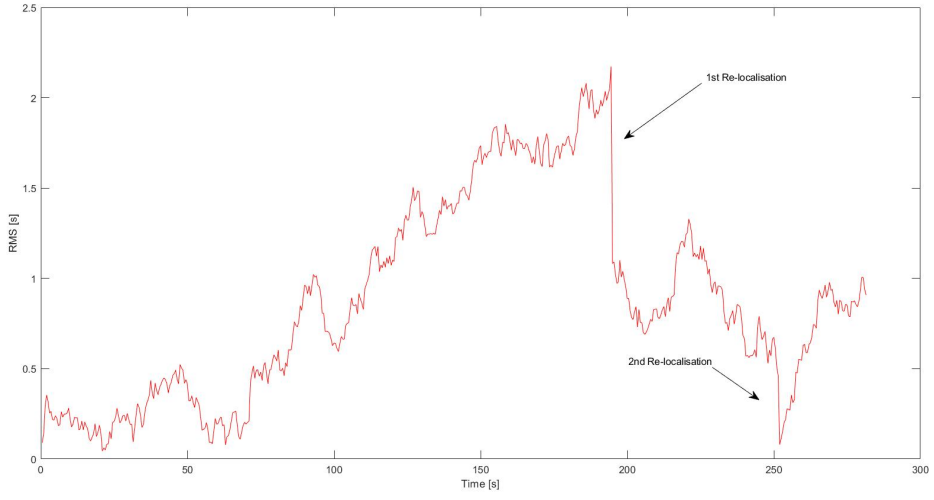


FIGURE 4.29: Total RMS error of the system through time

The figure 4.29 provides more convincing proof. It shows the RMS error through time. The two re-localisation are shown in the graph. The first one drops the RMS error from 2,2m to 1m and the second one from 1.2m to 0.2m.

The architecture provides good results. However, the SLAM algorithm is not perfect and suffers from technical problems.

4.7.4 Limitation

The SLAM architecture had some issue inherent to the technique used. The problem is only visible when performing several scans. The map construction has a slight drift through time, which makes the error grow in the re-localisation. The effect could come from the ICP. The data association doesn't reject outliers. Thus, the ICP associates every points from the map to the scan. If during the process, the robot explores a new area, the ICP will try to overlap the new points with the old ones, making the update of the map drifts. This drift is correlated with the number of new points in the scan of the surroundings and the distance of these points with the rest of the map. There exists two solutions to these issues. These two functions are independent and can be implemented in parallel, enhancing the update of the map as well as the re-localisation:

- The ICP must be updated with a "reject outlier" function before the data association. Allowing the ICP to overlap both scans without taking into account the new points.
- Raise the number of scans performed during the process. By doing this, the SLAM will have less new points to consider. This could be solved by implementing the dynamic mode in the SLAM. The re-localisation process would be called at every 360 of the scan.

The SLAM performance and architecture have been presented in this section 4.7, concluding this chapter 4. The next chapter 5 will be the discussion on different aspects of the project, its limitation, critique and suggestions for future improvement.

Chapter 5

Discussion

The objective of the thesis was to implement a robust sonar-based SLAM. As it is a practical subject, the thesis itself is the answer to the question:"How to fuse data to achieved a sonar-based SLAM". The project has demonstrated, how every part of the SLAM was formulated through the 7 prototypes and how the deliverables of each were incorporated into a global architecture to implement it. The results provided show the effectiveness of the process. The RMS error of all scenarios was maintained under 0.5m of error while providing a consistent 2D map.

Nevertheless, the scenarios were tested under the UUV simulation with all the implications that it entailed. The kinematic model of the Desistek Saga was a closer fit to a linear model, the error on the sensor measurements were assumed to be Gaussian and the sonar image was provided by a laserscan message without any distortion. Thus, all the results presented in chapter 4, alongside the prototypes is the proof that the SLAM works at least for the simulation. The thesis cannot be generalised to a real case scenario.

Transposition of the implementation on the real robot and further investigation of the algorithm and localisation method has be considered to generalize the SLAM to underwater vehicles. The project has to be implemented on a real robot, and the prototypes adapted to a real case scenario. The segmentation must be tested to ensure the validity of the data taken by the sonar. The Kalman filter and the linear assumption made, have high chances to not work for real case scenario. Therefore, a UKF or EKF must be considered to reinforce the localisation method and provide a non linear estimation. Finally, the ICP method could be improved using a Particle Swarm Optimization for a faster convergence.

Chapter 6

Conclusion

The thesis project was to implement a robust sonar-based SLAM with an accurate re-localisation method, while providing a consistent 2D map. The thesis is a contribution to underwater SLAM domain, in general, by providing a concrete explanation and an implementation example of the SLAM process. The work is also a contribution to OrcaHUB, who permitted their BlueROV2 to perform SLAM, which can be used for testing future underwater scenarios. The entire project code, curves and video explanation are available on the GitHub for further investigation.

The robot was able to perform both re-localisation tasks and provide 2D mapping in an underwater simulation environment provided by UUV simulator. The localisation was achieved using the fusion of the DVL and IMU sensors. The re-localisation was rendered possible with the use of the Iterative closest point algorithm coupled with a Kalman Filter. The robot was able to compare the scans with the map built before it moved and update its position. The map was able to construct itself thanks to octomap server. The map could, as well, update itself thanks to the ICP. The SLAM process was embedded in a global architecture allowing the SLAM to be performed and was well orchestrated.

The performance of every prototype has been discussed in their own section as well as in the discussion chapter 5. The results shows the effectiveness of the every prototype and the SLAM algorithm shows the limits of the model presented. Therefore, a significant amount of work has to be considered to improve the model and the algorithms used and open the SLAM presented to Real-case scenario.

Prototypes	Time(days)
P1 - Setting Enviroment	10
P2 - IMU and DVL fusion	3
P3 - Segmentation & Databuffer	8
P4 - Mapping with Octomap	10
P5 - Scan-Matching & ICP	15
P6 - Re-Localisation & Kalman Filter	15
P7 - SLAM	15

TABLE 6.1: Actual Gantt Chart of the Thesis

The figure 6.1 represent approximately the actual time send doing every prototypes.

6.1 Future work

As shown earlier, the objective of the work presented was to work on real ROV. To do so, the project will need a portage on the real robot. Every prototype would have to be performed and changed according to the real robot's functioning. Outlined, below are some of the ways this project can be further adapted:

The Dead reckoning will have to fuse data from the DVL and IMU with specific positions, different from the origins of the robot. The IMU and DVL were assumed to be at the origin of the robot (0,0,0). The change of position of these sensors would change the equation discussed in the section 4.2.1.

The ICP will have to be adapted to reject aberration for better re-localisation and update mapping performances. As said, in the section 4.2 the improvement of the data association can be multiple. The project could include a weight of the points associated as well as implement an outlier rejection function. By doing this, we ensure the map updates with a better accuracy inducing a better re-localisation.

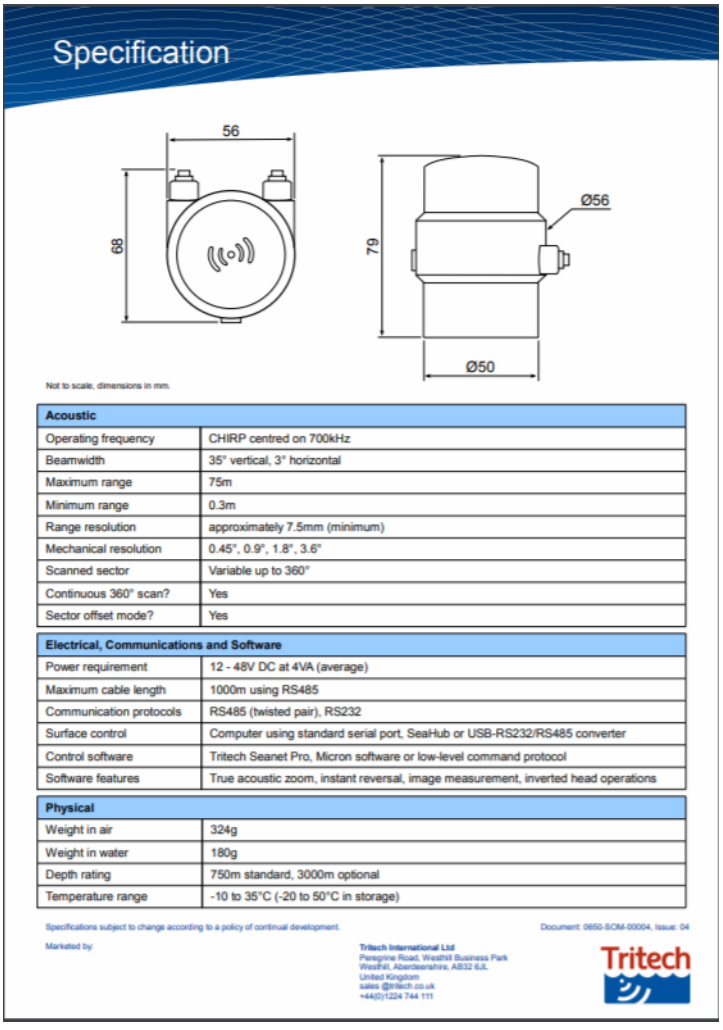
The Kalman Filter could be upgraded to a EKF or a UKF. The KF works in practice for the simulation because the noise measurement is Gaussian and the motion model is near a to linear behavior. In real case scenario, the noise will not be Gaussian and the motion model approximated to a linear one would not work efficiently. Thus, the implementation of a superior version of the Kalman filter is required.

A dynamical mode for the SLAM couldn't have been implemented because of the lack of time. The dynamic mode of the SLAM should help the performance, as well. This mode is very important as the new SLAM requires human intervention to launch the map and process the

scans. The robot in the current SLAM must remain at the same position while performing the scan. Whereas, the dynamic mode would allow the person controlling the robot to be free of these constraints. The buffer could save the position of the robot during the scan and reconstruct the scan afterwards. The robot could explore freely and process the ICP by itself, correcting the position and updating the map without intervention.

Appendix A

Sonar Datasheet



Specification

Not to scale, dimensions in mm.

Acoustic	
Operating frequency	CHIRP centred on 700kHz
Beamwidth	35° vertical, 3° horizontal
Maximum range	75m
Minimum range	0.3m
Range resolution	approximately 7.5mm (minimum)
Mechanical resolution	0.45°, 0.9°, 1.8°, 3.6°
Scanned sector	Variable up to 360°
Continuous 360° scan?	Yes
Sector offset mode?	Yes

Electrical, Communications and Software	
Power requirement	12 - 48V DC at 4VA (average)
Maximum cable length	1000m using RS485
Communication protocols	RS485 (twisted pair), RS232
Surface control	Computer using standard serial port, SeaHub or USB-RS232/RS485 converter
Control software	Tritech Seanet Pro, Micron software or low-level command protocol
Software features	True acoustic zoom, instant reversal, image measurement, inverted head operations

Physical	
Weight in air	324g
Weight in water	180g
Depth rating	750m standard, 3000m optional
Temperature range	-10 to 35°C (-20 to 50°C in storage)

Specifications subject to change according to a policy of continual development.

Document: 0650-SOM-00004, Issue: 04

Marketed by:

Tritech International Ltd
Peregrine Road, Westhill Business Park
Westhill, Aberdeenshire, AB32 6JL
United Kingdom
sales@tritech.co.uk
+44(0)1224 744 111

Tritech

FIGURE A.1: Micron Sonar Datasheet

Appendix B

Gantt Chart

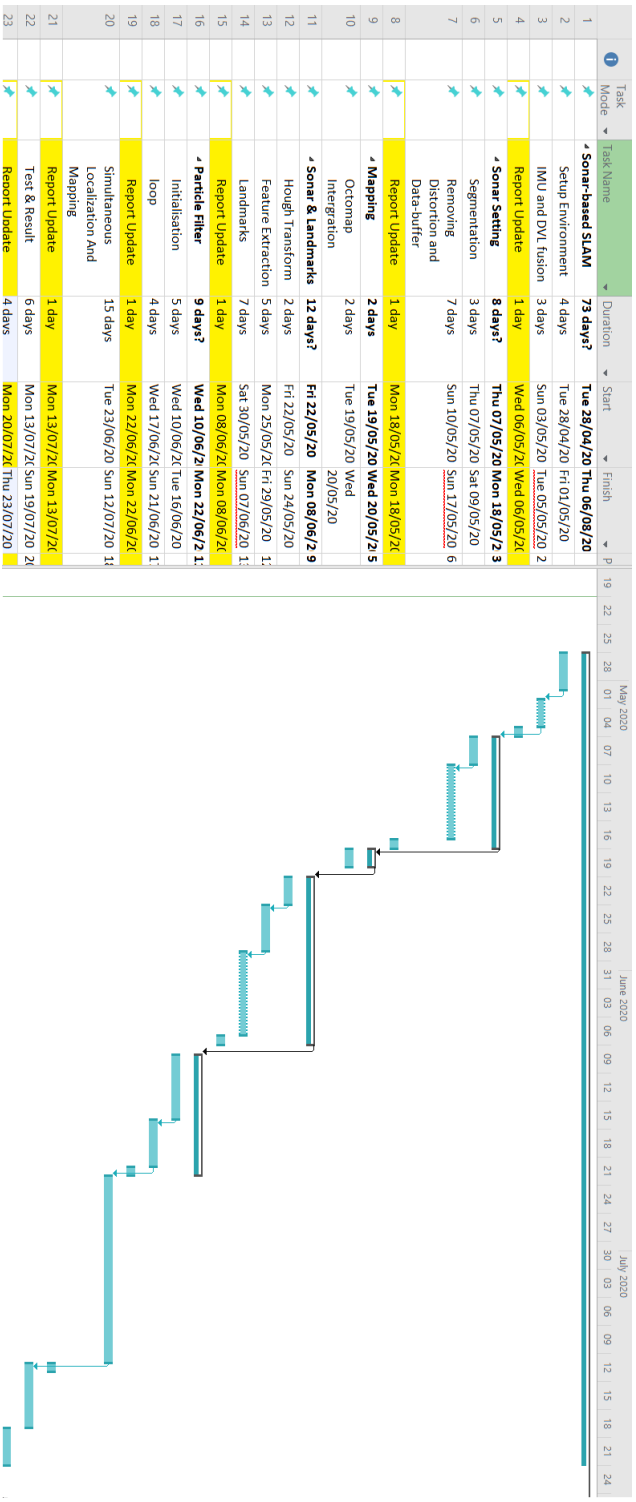


FIGURE B.1: Gantt Chart of the Project

Appendix C

Github Commits

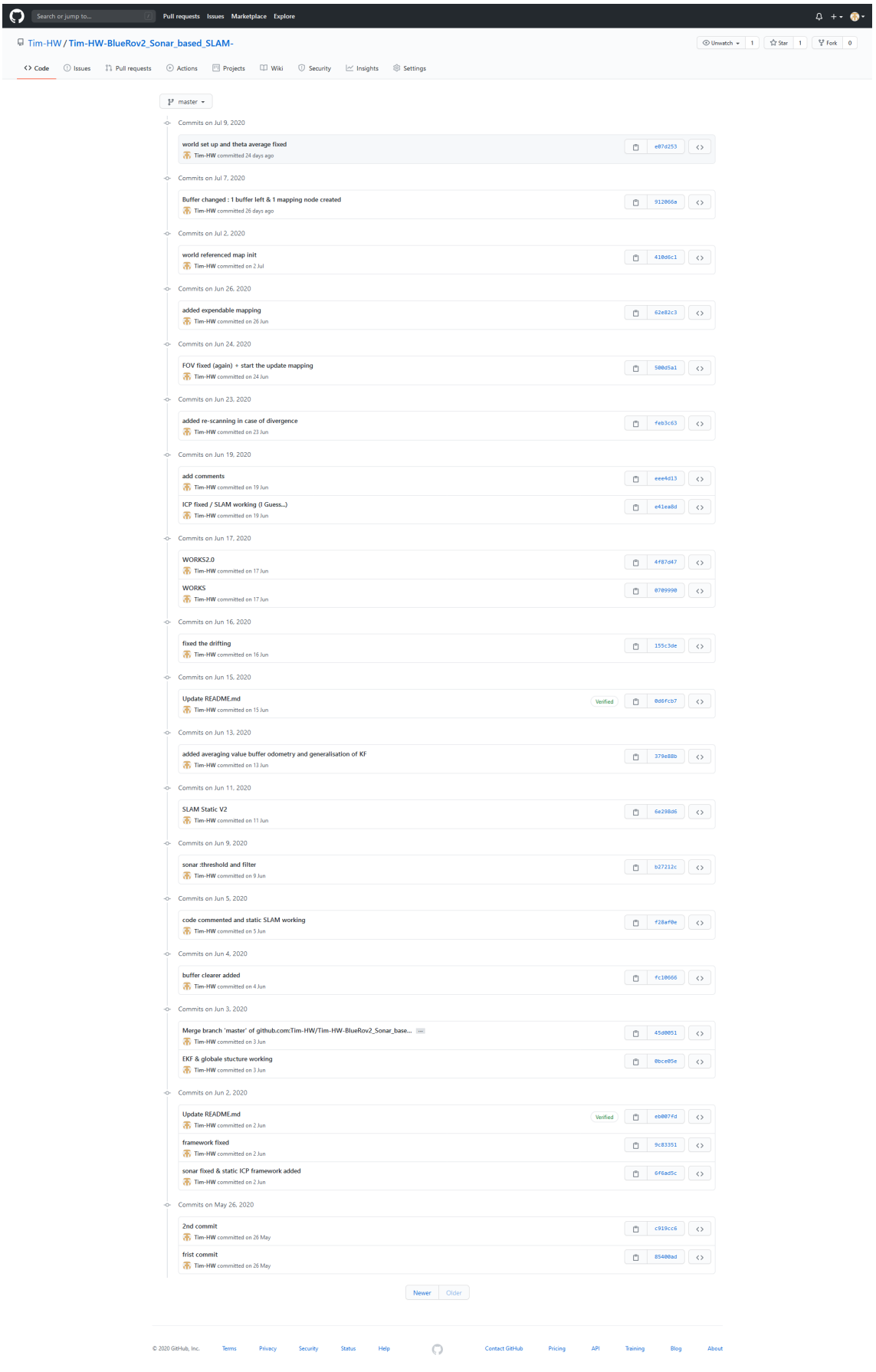


FIGURE C.1: Github Commits (1/2)

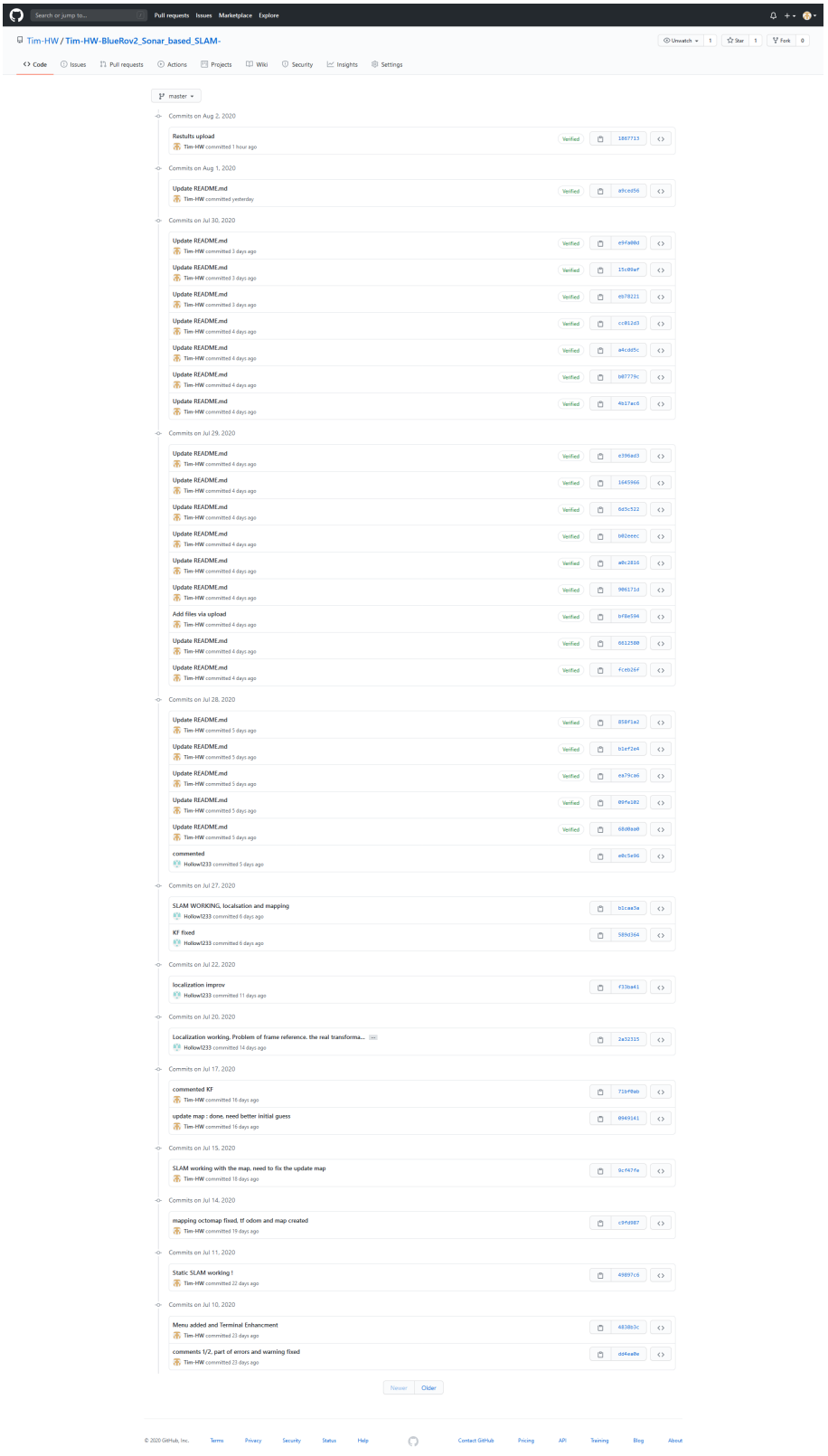


FIGURE C.2: Github Commits (2/2)
[note that both github account belongs to me. The second one appears when I came back in France. my SSH key logged my by default into this account]

Bibliography

[1] F. Hidalgo and T. Braunl, "Review of underwater slam techniques," in *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, pp. 306–311, IEEE, 2015.

[2] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," 05 2018.

[3] A. Vasilijevic, D. Nad, and N. Miskovic, "Autonomous surface vehicles as positioning and communications satellites for the marine operational environment - step toward internet of underwater things," in *2018 IEEE 8th International Conference on Underwater System Technology: Theory and Applications (USYS)*, pp. 1–5, IEEE, 2018.

[4] X. Cheng, H. Shu, Q. Liang, and D. Hung-Chang Du, "Silent positioning in underwater acoustic sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 3, pp. 1756–1766, 2008.

[5] 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*, IEEE, sep 2016.

[6] H. Bohm, V. Jensen, and N. Johnston, *Build your own underwater robot and other wet projects*, vol. 9594. Westcoast Words, 1997.

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

[8] G. Dissanayake, "Special issue on localization and mapping in challenging environments," *Robotics and Autonomous Systems*, vol. 97, pp. 16–17, 2017.

[9] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[10] C. Stachniss, "A short introduction to the bayes filter and related models." http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam03-bayes-filter-short.pdf.

[11] "Fundamentals of kalman filtering: A practical approach.(book by paul zarchan, howard musoff)(brief article)," *Mechanical Engineering-CIME*, vol. 123, no. 8, p. 83, 2001.

[12] S. Thrun, *Probabilistic robotics*. Intelligent robotics and autonomous agents, Cambridge, Mass.: MIT P., 2006.

[13] C. Stachniss, "Unscented kalman filter." http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf.pdf.

[14] R. Thiago Silva Da Rosa, G. B. Zaffari, P. J. Dias de Oliveira Evald, P. L. Jorge Drews, and S. Silva Da Costa Botelho, "Towards comparison of kalman filter methods for localisation in underwater environments," in *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, vol. 2017-, pp. 1–6, IEEE, 2017.

[15] C. Stachniss, "Introduction of particle filter." http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf.pdf.

[16] E. Hernández, P. Ridao, A. Mallios, and M. Carreras, "Occupancy grid mapping in an underwater structured environment," *IFAC Proceedings Volumes*, vol. 42, no. 18, pp. 286–291, 2009.

[17] C. Stachniss, "Graph-based slam with landmarks." http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam17-ls-landmarks.pdf.

[18] S. ABC, "What is sonar." https://www.scienceabc.com/innovation/what-is-sonar-definition-active-passive-uses-examples.html.

[19] D. Ribas, *Underwater SLAM for Structured Environments Using an Imaging Sonar by David Ribas, Pere Ridao, José Neira*. STAR (Series : Springer-Verlag), Berlin, Heidelberg: Springer Berlin Heidelberg, 1st ed. 2010. ed., 2010.

[20] J. P. Fula, B. M. Ferreira, and A. J. Oliveira, "Auv self-localization in structured environments using a scanning sonar and an extended kalman filter," in *OCEANS 2018 MTS/IEEE Charleston*, pp. 1–6, IEEE, 2018.

[21] D. Kumar, S. Rao, and K. Raju, "Integrated unscented kalman filter for underwater passive target tracking with towed array measurements," *Optik - International Journal for Light and Electron Optics*, vol. 127, no. 5, pp. 2840–2847, 2016.

[22] B. Allotta, A. Caiti, L. Chisci, R. Costanzi, F. D. Corato], C. Fantacci, D. Fenucci, E. Meli, and A. Ridolfi, "An unscented kalman filter based navigation algorithm for autonomous underwater vehicles," *Mechatronics*, vol. 39, pp. 185 – 195, 2016.

[23] B. He, Y. Liang, X. Feng, R. Nian, T. Yan, M. Li, and S. Zhang, "Auv slam and experiments using a mechanical scanning forward-looking sonar," *Sensors*, vol. 12, no. 7, pp. 9386–9410, 2012.

[24] F. Maurelli, A. Mallios, D. Ribas, P. Ridao, and Y. Petillot, "Particle filter based auv localization using imaging sonar," *IFAC Proceedings Volumes*, vol. 42, no. 18, pp. 52–57, 2009.

[25] T. Li, T. P. Sattar, and S. Sun, "Deterministic resampling: Unbiased sampling to avoid sample impoverishment in particle filters," *Signal Processing*, vol. 92, no. 7, pp. 1637–1645, 2012.

[26] F. Maurelli, Y. Petillot, A. Mallios, P. Ridao, and S. Krupinski, "Sonar-based auv localization using an improved particle filter approach," pp. 1 – 9, 06 2009.

[27] D. Forouher, J. Hartmann, M. Litza, and E. Maehle, "Sonar-based fastslam in an underwater environment using walls as features," in *2011 15th International Conference on Advanced Robotics (ICAR)*, pp. 588–593, June 2011.

[28] C. Chen, H. Zhu, M. Li, and S. You, "A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives," *Robotics*, vol. 7, no. 3, p. 45, 2018.

[29] M. Quan, S. Piao, M. Tan, and S.-S. Huang, "Accurate monocular visual-inertial slam using a map-assisted ekf approach," 2017.

[30] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2015-, pp. 298–304, IEEE, 2015.

[31] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[32] P. Li, T. Qin, B. Hu, F. Zhu, and S. Shen, "Monocular visual-inertial state estimation for mobile augmented reality," in *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 11–21, IEEE, 2017.

[33] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[34] B. Joshi, S. Rahman, M. Kalaitzakis, B. Cain, J. Johnson, M. Xanthidis, N. Karapetyan, A. Hernandez, A. Q. Li, N. Vitzilaios, and I. Rekleitis, "Experimental comparison of open source visual-inertial-based state estimation algorithms in the underwater domain," 2019.

[35] S. Rahman, A. Li, and I. Rekleitis, "Sonar visual inertial slam of underwater structures," pp. 5190–5196, Institute of Electrical and Electronics Engineers Inc., 2018.

[36] S. Rahman, A. Q. Li, and I. Rekleitis, "An underwater slam system using sonar, visual, inertial, and depth sensor," 2018.

[37] W. R. Fried, "History of doppler radar navigation," *Navigation*, vol. 40, no. 2, pp. 121–136, 1993.

[38] R. Cerqueira, *A multi-device sonar simulator for real-time underwater applications*. PhD thesis, Federal University of Bahia, 2019.

[39] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[40] B. O. A. Haugaløkken, S. S. Sandøy, I. Schjølberg, J. A. Alfredsen, and I. B. Utne, "Probabilistic localization and mapping of flexible underwater structures using octomap ; probabilistic localization and mapping of flexible underwater structures using octomap," Institute of Electrical and Electronics Engineers (IEEE), 2018.

[41] E. Recherche, E. Automatique, S. Antipolis, and Z. Zhang, "Iterative point matching for registration of free-form curves," *Int. J. Comput. Vision*, vol. 13, 07 1992.

[42] 1988kramer, "Lidar odometry with icp." http://andrewjkramer.net/tag/tutorial/.

[43] P. Besl and N. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, 04 1992.

[44] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," pp. 145–152, 02 2001.