NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ALESSIA JENNIFER FORTES OFFSAS

BSc in Electrical and Computer Engineering

SHIP MULTIMODEL 3D RECONSTRUCTION AND CORROSION DETECTION

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

July, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SHIP MULTIMODEL 3D RECONSTRUCTION AND CORROSION DETECTION

ALESSIA JENNIFER FORTES OFFSAS

BSc in Electrical and Computer Engineering

Adviser: José António Barata de Oliveira
Full Professor, NOVA University Lisbon

Co-adviser: Francisco Antero Cardoso Marques
Researcher Engineer, NOVA University Lisbon

Examination Committee

Chair: Luís Filipe dos Santos Gomes
Full Professor, FCT-UNL

Adviser: José António Barata de Oliveira
Full Professor, FCT-UNL

Member: João Almeida das Rosas
Full Professor, FCT-UNL

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
July, 2023

Ship Multimodel 3D Reconstruction and Corrosion Detection

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.15) [1].

Acknowledgements

First I would like to thank my advisor José Barata, for giving me the opportunity to join the RICS group and always be available to help me with anything. I would also like to thank my co-advisor Francisco Marques, who accompanied me in this journey from the start and even with all his busy life and work, could always find a little time to guide me in the making of this dissertation.

One colleague from NOVA Lisbon University that also deserves a big thank you is Afonso Alves, as he has helped me a lot in the writing of this dissertation by giving me ideas and useful tips, and for accompanying me since the beginning of our Electrical and Computer Engineering course.

I want to give the biggest thank you to my family, specially my mother as she moved heaven and earth to get me where I am today and to make my life as easy as possible. I would not be able to achieve none of this without her and for that I thank her from the bottom of my heart.

Also, I want to thank my boyfriend for his endless patience, for always being there no matter what and for giving me the strength to achieve the impossible.

Last but not least, I want to give a general thank you to everyone that directly or indirectly help me throughout my education, every contribution, even as little as it could be, helped me achieve the position I'm in today.

Thank you!

Abstract

3D reconstruction has been an area of increased interest due to the current higher demand in applications, such as virtual realities, 3D mapping, medical imaging, and many others. Although, there are still many problems associated with reconstructing a real-life object, such as capturing occluded zones, noise, and processing time. Furthermore, as deep learning technologies advance, there has been a growing interest in using such methods to replace human-driven tasks, namely corrosion inspection, as it decreases the risk of injury of the inspector, it is more efficient due to less time taken, and is cost-saving.

This dissertation proposes a method for reconstructing a 3D model of ships using aerial RGB images and terrestrial RGB-D images, along with a system capable of detecting the corroded parts of the ship and highlighting them in the model. Using two different sensors in two different ground planes mitigates some of the occlusion problems and increases the final model's accuracy. The current dissertation also aims to pick the methods that have the best trade-off between accuracy and computational speed. The final model can be advantageous for corrosion inspectors, as they will have the model of the ship, as well as the corroded zones which, with that information, can choose the steps to take next without the need to manually inspect the ship or even be in the same site as the ship.

The final model is a fusion of three different 3D models. The model obtained from RGB images exploits Structure from Motion algorithm which recovers the 3D aspect of the ship from 2D images. As for the remaining models, RGB-D images were used in conjunction with the Open3D library to create 3D structures from both sides of the ship.

The corrosion classifier model was trained in Google Colab and achieved an accuracy of 97.44 % on the test dataset. The images used to create the SfM 3D model were each divided into a total of 40 regions and fed into the classifier to simulate a less concise image detection algorithm instead of an image classification algorithm. The results were encoded into the 3D model, highlighting the corroded zones.

Keywords: 3D reconstruction, Corrosion classification, Point cloud fusion, Deep learning, Feature-based registration, Structure from Motion

Resumo

A reconstrução 3D tem sido uma área com crescente interesse devido à maior demanda em aplicações como realidade virtual, mapeamento 3D, imagens médicas e muitos outros. Embora, existem ainda muitos problemas associados à reconstrução 3D de um objeto real. Exemplos desses são a captura de zonas oclusas, o ruído e o tempo de processamento necessário para efetuar a reconstrução. Adicionalmente, com o avanço das tecnologias de deep learning, tem havido um acrescido interesse em usar ditos métodos para substituir tarefas realizadas por humanos como, por exemplo, a inspeção de corrosão, pois diminui o risco de lesões ao inspetor, tem maior eficiência devido a um menor tempo gasto, e economiza os custos.

Esta dissertação propõe um método de reconstrução de um modelo 3D de navios, utilizando imagens RGB aéreas e imagens RGB-D terrestres, juntamente com um sistema capaz de detetar as zonas com corrosão no navio e destacá-las no modelo. O uso de dois sensores diferentes em dois meios diferentes atenuará alguns dos problemas de oclusão e aumentará a precisão do modelo final. A presente dissertação também visa escolher os métodos que apresentam o melhor compromisso entre precisão e velocidade de processamento. O modelo final poderá ser vantajoso para os inspetores de corrosão, pois terão o modelo do navio, bem como as zonas com corrosão que, com essa informação, poderão escolher quais os passos a seguir, sem a necessidade de inspecionar manualmente o navio ou mesmo deslocar-se para o local do navio.

O modelo final é uma fusão de três modelos 3D diferentes. O modelo obtido a partir de imagens RGB tirou partido do algoritmo Structure from Motion, que recupera o aspeto 3D do navio a partir de imagens 2D. Quanto aos modelos restantes, as imagens RGB-D foram utilizadas em conjunto com a biblioteca Open3D para criar estruturas 3D de ambos os lados do navio.

O modelo de classificação de corrosão foi treinado em ambiente Google Colab e alcançou uma exatidão de 97.44% no dataset de teste. As imagens usadas para criar o modelo SfM 3D foram, cada uma, fracionadas num total de 40 regiões e dadas ao modelo de classificação com o intuito de simular um modelo de deteção de imagem menos conciso em vez de um modelo de classificação de imagem. Os resultados foram codificados no modelo 3D, destacando as zonas com corrosão.

Palavras-chave: Reconstrução 3D, Classificação de Corrosão, Fusão de nuvens de pontos, Deep learning, Registro baseado em características, Structure from Motion

Contents

List of Figures

List of Tables

Acronyms

2D	Two-Dimensional
3D	Three-dimensional
AI	Artificial Intelligence
ALS	Airborne Laser Scanning
BA	Bundle Adjustment
CNN	Convolutional Neural Network
DL	Deep Learning
DoG	Difference of Gaussian
EPnP	Efficient Perspective-n-Point
FAST	Features from Accelerated Segment Test
FN	False Negative
FP	False Positive
FPFH	Fast Point Feature Histogram
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HSI	Hue, Saturation and Intensity
ICCP	Iterative Closest Compatible Point
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
ISS	Intrinsic Shape Signatures

LiDAR Light Detection and Ranging
LLSQ Linear Least Squares
LSQ Least Squares

ML Machine Learning
MVS Multi-View Stereo

NLSQ Nonlinear Least Squares

ORB Oriented FAST and Rotated BRIEF

PCA Principal Component Analysis
PCL Point Cloud Library
PCP Principal Component Points
PFH Point Feature Histogram
PnP Perspective-n-Point

R-CNN Region-based Convolutional Neural Networks
RANSAC Random Sample Consensus
RGB Red, Green and Blue
RGB-D Red, Green and Blue - Depth
RIMM Reversed Iterative Mathematic Morphological

SDF Signed Distance Function
SfM Structure-from-Motion
SfS Shape-from-Silhouette
SHOT Signature of Histograms of Orientations
SIFT Scale-Invariant Feature Transform
SURF Speeded-Up Robust Features
SVM Support Vector Machine

TLS Terrestrial Laser Scanning
TN True Negative
ToF Time-of-Flight
TP True Positive
TSDF Truncated Signed Distance Function

UAV Unmanned Aerial Vehicle

1 Introduction

With the continuous improvement in sensing technologies, increasing power of GPUs, and the continuous lowering of the costs of the sensors, 3D reconstruction has been attracting more attention, increasing its research fields, as the reconstruction from 2D images are not able to meet many of the requirements in today's high demands. The reconstructed models are useful in a vast number of applications, such as virtual reality, structural inspection, architecture, navigation, and many others. Nevertheless, reconstructing a virtual model of a real object still presents many challenges that need to be addressed. These challenges present themselves in both the sensing methods, such as the accuracy of reconstruction and dealing with occluded zones, and in processing methods, such as the removal of outliers and processing speed of the reconstruction, therefore, these are the most common and vital ones to be dealt with. Utilizing a fusion of sensors, sometimes also combining a fixed sensor with another mobile one, can attenuate some of those problems. For example, the mobile sensor can scan zones occluded to the fixed sensor, or even reach places out of range, while the fixed sensor can collect data with high accuracy, complementing one another.

Since the focus of this dissertation is on ships, another important aspect is the inspection of their structural integrity. Visual inspections have to be performed regularly to evaluate the condition of a ship as critical damages, such as corrosion, put at risk the safety of people on board and the investment placed on the ship. However, this inspection can be strenuous, time-consuming, and prone to human error when performed by a human inspector. With the ongoing research in the field of computer vision, many algorithms have been developed and improved in order to facilitate this job, making it possible to do it faster and, in many cases, more accurately than a human through the use of, for example, a simple camera mounted in an Unmanned Aerial Vehicle (UAV).

1.1 Problem

The two topics mentioned previously are very relevant in today's world and further research needs to be done to improve these technologies. There are almost no studies conducted in 3D reconstruction of maritime vehicles, thus this dissertation will focus on such, while at the same time trying to overcome some of the challenges presented previously, such as accuracy and computing time. When it comes to 3D reconstruction, to find the best

method to achieve the highest accuracy possible, different sensing technologies mounted in a UAV are experimented with. RGB-D sensors are known for achieving higher levels of accuracy when compared to 3D models obtained from images of a simple RGB sensor, but at a higher computational cost due to the large amount of data that is generated and with increased noise and data distortions, so each presents its advantages and disadvantages.

As for corrosion detection, there are numerous studies done relating to image classification and detection, but not as many were done specifically for corrosion so, in this project, emphasis is given to it. With the progress in computational power and improved machine learning detection algorithms, images can be exploited to accurately detect these kinds of defects. Although, one of the main problems associated with these algorithms is the training part, namely the creation of a decent training dataset that enables the model to accurately detect and, at the same time, avoid overfitting the model. This becomes especially difficult when there are no available datasets online for public use of labeled images of corrosion, so the process of acquiring this data can be quite time-consuming to do by a single person.

1.2 Proposed Solution

The proposed solution for the 3D reconstruction will be based on the use of aerial RGB data, obtained with the sensor mounted on a drone, and terrestrial RGB-D data, obtained with the sensor Intel Realsense D435i, to 3D reconstruct a ship. Additionally, the system integrates a corrosion classification algorithm that highlights the corroded parts of the ship in the 3D model. A more detailed explanation of the proposed model is presented in Chapter 3. To simplify the development of this system, a division into four smaller steps is done:

1. 3D Reconstruction using just the RGB sensor.
2. Corrosion classification model creation and highlighting the corroded areas in the previous 3D model.
3. 3D Reconstruction using just the RGB-D sensor.
4. Merging both 3D models into a single one.

1.3 Dissertation Outline

The outline of this document is as follows:

- **Chapter 2:** some relevant theoretical background is explained, namely certain common algorithms used in the reconstruction process, important aspects of computer vision and image classification, and a detailed explanation about convolutional neural networks.

- **Chapter 3:** an overview of the proposed model, along with an outline of some of the developed methods are presented.
- **Chapter 4:** a walkthrough of the research is done by explaining the process of reconstruction and corrosion detection of the ship. Furthermore, this chapter is also dedicated to the justification of the methods chosen during the development of this project.
- **Chapter 5:** a display and analysis of the outcome is conducted, where an evaluation of accuracy and computing time is performed, as well as a visual qualitative analysis.
- **Chapter 6:** the research done is reviewed and discussed, complemented with some conclusions related to the results, and additionally, given a proposal of future research on the subject.

2 Theoretical Background

In this chapter, it is presented the fundamental concepts associated with 3D reconstruction and corrosion classification and also reviewed and summed up some scientific works related to the subject at hand.

In Sec. 2.1, it is presented an overview of the different steps and methods for creating 3D models using solely an RGB-D sensor, an RGB sensor, or a fusion of both. In Sec. 2.2, an overview of computer vision and image classification is presented, as well as an in-depth explanation of convolutional neural networks. Lastly, Sec. 2.3 is dedicated to the literature review and its advantages and disadvantages concerning the current dissertation theme.

2.1 3D Reconstruction

A 3D model portrays real-life objects or scenes that can be visualized digitally. The models can be either surface models or volumetric models. For the former, the surface shape of the object is represented, while in the latter, the inside of the object is also modeled. In order to reconstruct an object, a scanning process is required. This process measures the geometric features and visual appearances in order to obtain the model in a fully or almost fully automatic manner.

3D reconstruction has always been a topic of interest due to the many applications it can have, such as environmental monitoring, security analysis, and autonomous navigation. Laser sensors are widely used for this purpose, but with the advance of machine learning algorithms, there has been a growth in the use of image sensors, also due to being significantly cheaper.

The requirements of each reconstruction process are highly linked to the application. In some applications, the most important factor is high accuracy, while processing and scanning times are not as relevant, while in others applications, the opposite might be true. Knowing some information about the shape of the target object *a priori* will allow for a faster and more precise reconstruction.

2.1.1 Framework for Reconstruction using RGB-D Images

When scanning using RGB-D sensors, depth maps, and RGB images are produced which, in conjunction with the camera's intrinsic parameters, can estimate a 3D point cloud or, in

other words, a set of points in the 3D space domain that represent the object. Creating a precise and accurate 3D model from point clouds may contain many steps, being that not all are required, appropriate, or executed in the same order, as it depends on the object being scanned. Nevertheless, the traditional framework in a reconstruction process is composed of point cloud acquisition, raw point cloud processing, model fitting, feature extraction, segmentation, and classification and registration.

2.1.1.1 Point Cloud Acquisition

A depth sensor is a remote sensing technology that allows to determine the depth of an object by projecting a laser with wavelengths outside the visible spectrum, most commonly in the infrared zone, which bounce from the scanned object back to the sensor. RGB-D sensors have a few advantages in 3D reconstruction. Some of the most crucial advantages are its ability to add texture to the final 3D model, its versatility to be used both indoors and outdoors, and its lower power consumption. Other advantages include a high frame rate and cheaper prices compared to some other sensors. On the other hand, RGB-D sensors are affected by lighting conditions, are susceptible to motion blurs, and are unreliable and inaccurate when the scene does not present sufficient texture. Other minor disadvantages include its limited short range and low resolution.

Many applications exploit RGB-D data. Recently, autonomous vehicles have been increasingly exploiting these sensors for obstacle detection and decision-making [2, 3]. These sensors are also used in navigation systems, real-time localization, and mapping systems [4–6]. However, the most common and known application of RGB-D sensors is in human action recognition [7, 8].

The measuring process used by RGB-D sensors can be referred to as an active method or passive method, but most of the sensors exploit active methods. In active methods, signals are emitted in the form of structured light or modulated laser light, subsequently being reflected by the object and recorded its information by the sensor. Three of the most common types of RGB-D sensors are structure/coded light, stereo vision, and Time-of-Flight (ToF). An overview of these three methods is shown in Fig. 2.1.

Sensors that take advantage of structured light exploit coded structure light pattern projections, which are deformed when hitting surfaces, to recover the surface shape of an object. This allows for correspondences between image points and points of the projected pattern to be found and, consequently, obtain the 3D coordinates by using triangulation. This method is not as used nowadays. Stereo vision depth cameras behave in a similar way to human vision. They possess two sensors with a fixed small distance between them. Each sensor captures an image at a slightly different angle and compares them with each other. This comparison will give the depth information. For ToF depth sensors, such as LiDAR, a laser is emitted and swept across a scene, and the time taken for the light to reflect on the object and return to the sensor is measured, as depicted in Fig. 2.2.

For ToF, the time is measured starting from the moment the pulse is emitted until

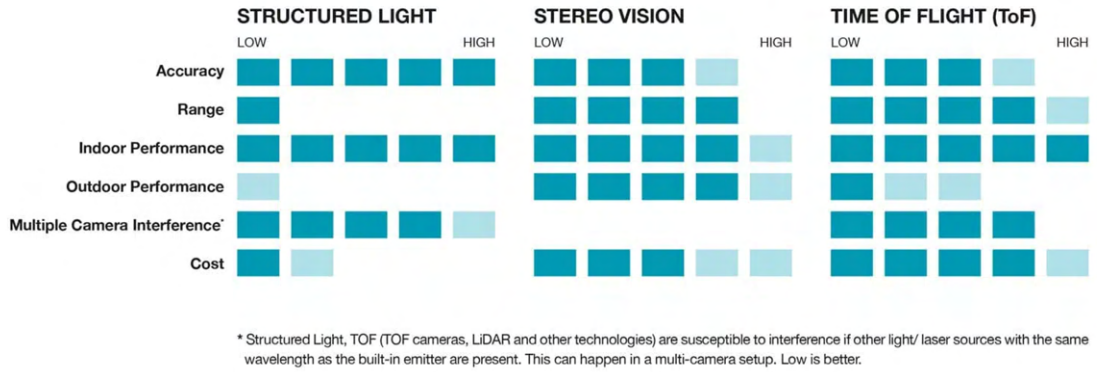


Figure 2.1: Overview of the performances of different types of depth sensors. [9]

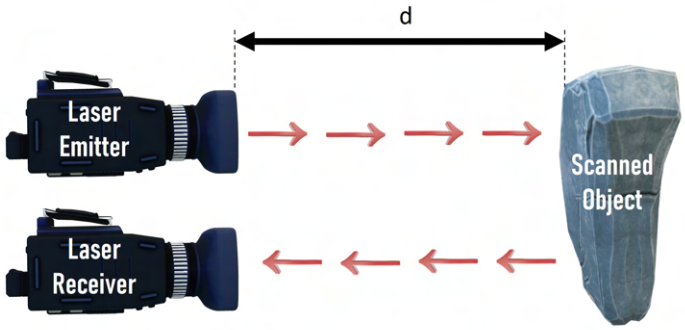


Figure 2.2: Time of Flight principle.

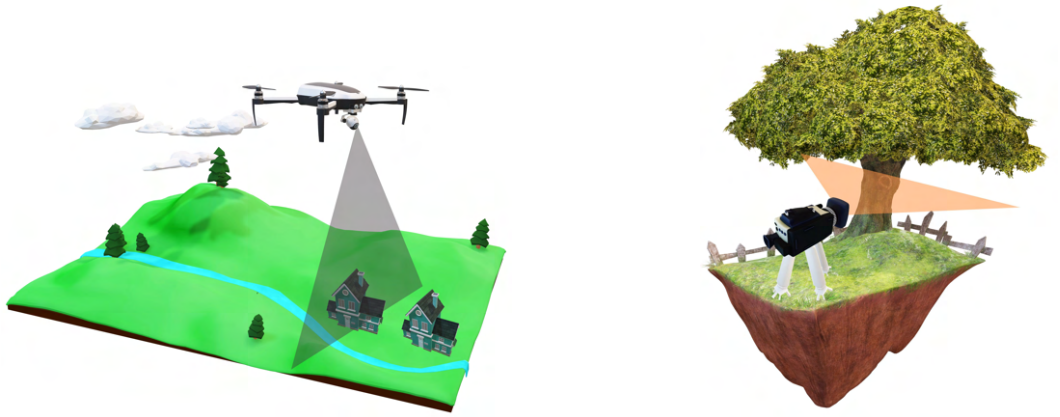
it receives the returning pulse, thus allowing to compute the distance. The formula to calculate this distance d to the object, in meters, is determined according to (2.1).

$$d = \frac{1}{2} c \cdot t \tag{2.1}$$

with c the speed of light, equal to 299792458 m/s, and t the time measured, in seconds.

As for the classification of RGB-D systems that use active methods, two main possible classifications can be attributed based on their location: Airborne Laser Scanning (ALS) and Terrestrial Laser Scanning (TLS). ALS systems are mounted on an aircraft, such as airplanes, drones, or helicopters, and emit the laser towards the ground surface, as illustrated in Fig. 2.3a. TLS systems are mounted on a moving ground vehicle or a fixed structure, such as a tripod, as illustrated in Fig. 2.3b. TLS can achieve a higher resolution than ALS, as the latter has a resolution between the decimetric and metric, and the former between millimetric and centimetric. TLS can also obtain a denser point cloud in comparison with ALS [10]. The most common scanning patterns in ALS are zigzag, parallel, or elliptical (Fig. 2.4).

To obtain the 3D points of an object, a sequential scanning and distance measurement using one of the methods presented previously, depending on the type of sensor, is



(a) (b)

Figure 2.3: Most common classifications of laser systems. a) ALS. b) TLS.

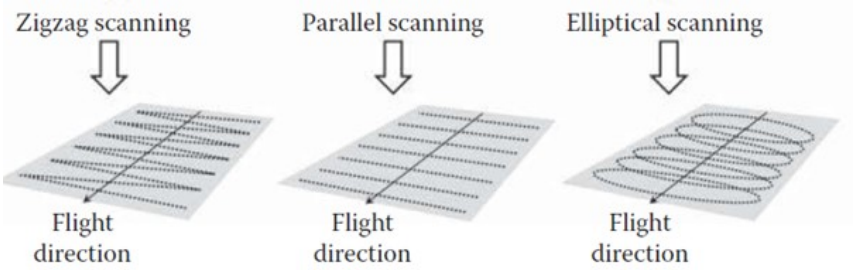


Figure 2.4: Common scanning patterns in ALS (Adapted from [11]).

performed. Subsequently, since the laser pulses are in a well-defined and known direction and the RGB-D sensor coordinates are also known, after converting the depth map into point clouds, we can extract the x, y, and z locations of each scanned point relative to the sensor. Lastly, transforming the sensor's position coordinates into a fixed frame, allows to obtain the final point cloud of the scan relative to a universal coordinate system.

2.1.1.2 Raw Point Cloud Processing

Raw point clouds require to go through a stage called pre-processing. This stage allows for improvement of the accuracy of the points, acceleration of the reconstruction process, and improvement of the quality of the reconstructed object. As mentioned before, each step is dependent on the application, including point cloud processing. Nevertheless, almost all reconstruction processes require some form of filtering, including both downsampling and noise/outlier removal.

Point clouds have large amounts of data, where many of the points are useless and redundant. Also, the points can suffer from noise contamination and outliers due to a variety of factors such as physical limitations of the sensors, the type of technique used for the acquisition, surface complexity, weather conditions, reflective nature of the object, and positioning of the sensor with respect to the object. As a consequence of these factors,

filtering techniques that preserve the geometric features of the object have been an active subject of research.

To reduce the computational cost for later steps, downsampling is used to reduce the number of points. This step is the first in the filtering process. Next are presented the two most common algorithms for downsampling:

- **Voxel Grid Filtering**: Also known as Voxel Grid Downsampling, defines a grid of 3D boxes in a point cloud, called voxel, and afterward, approximates all points that lie within the bounds of a voxel by computing either the center point, the centroid or a random point, converging into a single point. An illustration of the process is shown in Fig. 2.5. This process of calculating the centroid is slower than approximating with the center of the voxel or the random point, but it gives a more accurate representation of the object. The fastest method is by choosing a random point but is also the most inaccurate. The sampling size can be adjusted by increasing or decreasing the size of the voxel, as the smaller it is, the more information is retained.

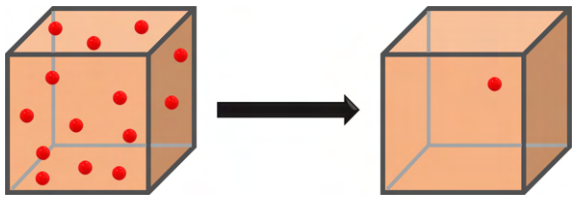


Figure 2.5: Illustration of downsampling by applying Voxel Grid Filter.

- **Surface Normal**: In this method, it is derived the surface normal of a point cloud, meaning that the perpendicular vector of the tangent plane of every point will be determined using its neighborhood points and afterward, those vectors will be represented on a Gaussian sphere, in which the peak points in that sphere will be removed [12].

There are many types of filtering techniques for noise and outlier removal, such as statistical-based, neighborhood-based, and projection-based:

- **Statistical-based**: This type of filtering usually fits a probability distribution to the data in order to determine the points to be removed.

- **Neighbourhood-based**: Are based on measurements between a point and its neighbors. One example of this type of filtering is Radius Outlier Removal, where points are removed if a predefined number of neighbors in a certain radius is inferior to a predefined number (Fig. 2.6).

- **Projection-based**: This filtering technique is based on the adjustment of every point by using different projection strategies.

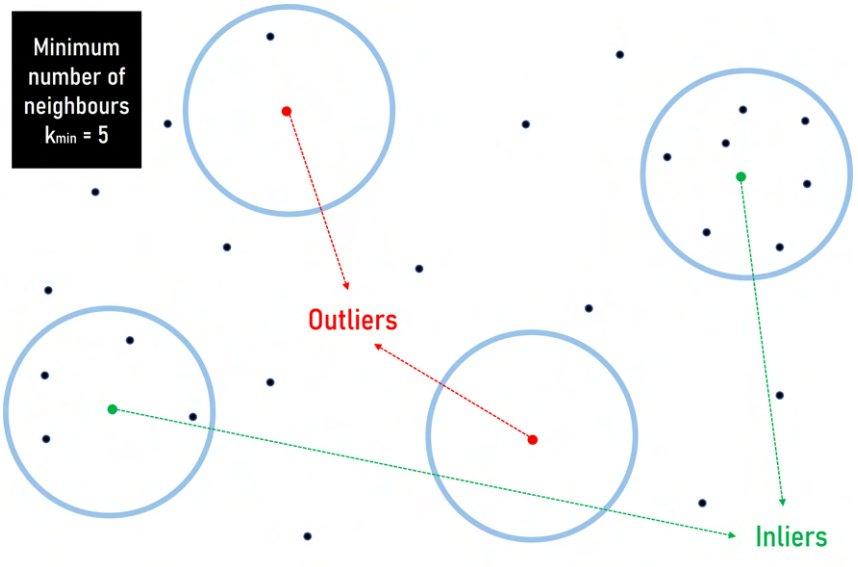
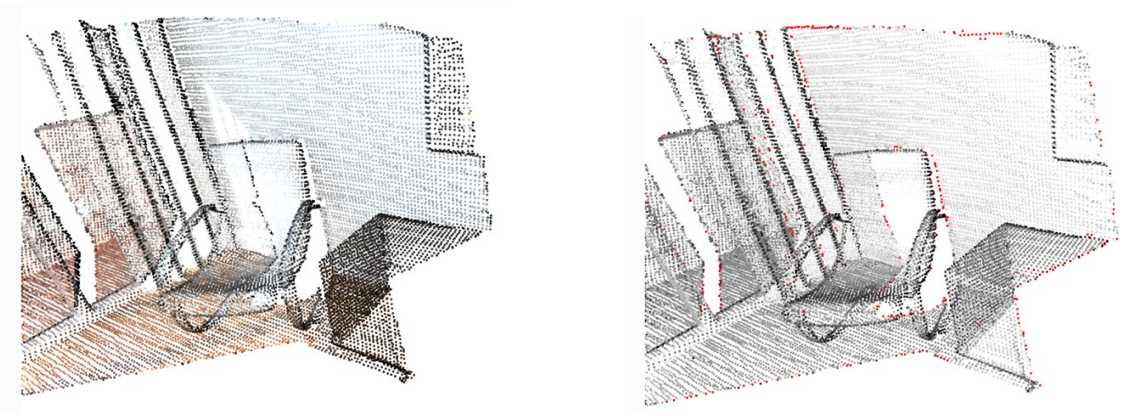


Figure 2.6: Radius Outlier Removal method for detecting inliers and outliers.

- **Hybrid-based**: Used to achieve filtering results that solely one of the previous methods could not achieve. For example, some of the noise present in the cloud data is caused by measurement errors and in order to minimize them, a Statistical Outlier Removal filter could be applied (Fig. 2.7). This filter deletes points inconsistent with the others, and to do so, it takes into consideration the mean distance from each point to all its neighbors [13]. Afterward, by modeling the distances by a Gaussian distribution, if the distance is superior to a certain chosen interval, it is removed. This type of filtering is a hybrid between statistical and neighborhood-based, although it is commonly described only as statistical-based filtering.



(a) (b)

Figure 2.7: Result of applying a Statistical Outlier Removal filter [14]. a) Image after downsampling with Voxel Grid Downsampling. b) Statistical Outlier Removal of the downsampled point cloud.

2.1.1.3 Model Fitting

Many of the objects being scanned can be described by a mathematical model or equations, such as polynomials, planes, lines, etc. Besides, point clouds almost always contain the target object and other elements, such as other components in the background or the ground. Since the ground can be considered a plane and the points belonging to it are irrelevant in the work of this dissertation, they can be eliminated using model fitting and, therefore, aiding in further reducing computational cost for later steps. There are multiple methods for model fitting in the literature, but the most common ones are Random Sample Consensus (RANSAC), Least Squares (LSQ) fitting, and Hough Transform.

RANSAC is an iterative method that identifies outliers in a point cloud and estimates the parameters for a mathematical model that disregards those identified outliers (Fig. 2.8). In this method, a random sub-sample of points in the point cloud is chosen, a fitting model for it is computed and the points that do not fit in the fitting model are considered outliers, while the remainders are considered inliers. This process is repeated for N iterations. Afterward, the model with the most inliers is picked as the final model. However, to increase computation efficiency, the number of iterations can be reduced if the iteration is stopped after finding a model with a defined inlier-to-outlier ratio. The advantage of RANSAC is its ability to estimate models with high accuracy even when there is a large number of outliers or the model shape is complex. Another advantage is its simplicity to use. A disadvantage of RANSAC is that, although working well when many outliers are present, if the percentage of inliers is less than 50% it does not achieve satisfactory results. Despite that, many optimized RANSAC algorithms have been proposed in the literature that allows to achieve a good result for a percentage of inliers up to around 5%, as seen in a work performed by Schnabel *et al* [15].

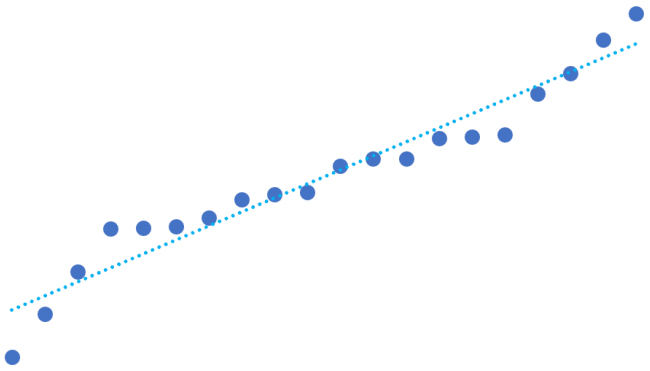


Figure 2.8: Example of model fitting using RANSAC.

A mathematical method for determining the best way of fitting a curve in a given set of points, approximating overdetermined systems, where the number of equations is greater than the unknown variables, is determined using LSQ. This method consists in minimizing the sum of the squares of the residuals, that is minimizing the sum of the difference between the points and the curve of the fitted model, in every equation. This

technique can be either linear or nonlinear. The Linear Least Squares (LLSQ) fitting is the simplest and consists in fitting a straight line through the set of points. The Nonlinear Least Squares (NLSQ) fitting is a process that iteratively performs LLSQ until convergence is achieved, to form a function. The main disadvantage of LSQ is that outliers have a huge effect on obtaining the fitting model, as they move the curve toward their direction.

As for the Hough Transform, this method is used for feature extraction in images to detect shapes within, such as lines, circles, and ellipses. This, however, can be extended to 3D to detect planes within point clouds, characterized by (2.2):

$$ax + by + c = z \tag{2.2}$$

where a, b, and c are parameters determined by the transform. The Hough Transform gives a robust detection when there is noise and occlusion present in the set of points, but it is not quite as accurate when the model to be fitted has a complex shape.

2.1.1.4 Feature Extraction

A feature is a representation of information about a region of interest in the point cloud. These features represent geometric characteristics that describe the structure of an object. The purpose of feature extraction is to extract certain pieces of information which are critical, non-redundant, summarize the original set of features, and are suitable for the respective application which, consequently, reduces the amount of data to be processed, also known as dimensionality reduction [16]. In point clouds, the features can be:

- **Point attributes**: where points are accounted for.
- **Shape**: which takes into consideration connected regions, to later extract characteristics such as volume, size, and orientation.
- **Local features**: the variation of intensity from the 3D neighbourhood points is considered.

To perform a feature extraction two steps are needed: feature detection and feature description. In feature detection, the points are analyzed with the objective of finding any feature and its locations. Common examples of feature detectors for point clouds are Harris 3D and Intrinsic Shape Signatures (ISS).

Harris Corner Detector was developed by Harris and Stephens [17] for 3D reconstruction via 2D images, where the corners and edges were detected by the sudden change in pixel intensity from its neighborhood. Since some point clouds are extracted from sensors that do not capture color, such as LiDAR, the necessity to create an improved technique resulted in the development of a method called Harris 3D, proposed by Sipiran and Bustos [18], where instead of considering color intensities, the local surfaces around a point were exploited to detect vertices.

ISS was proposed by Yu [19]. It is a shape detector for a local region of a point cloud. This method is based on the neighborhood of each point in a region and the nearest neighbors of a point in a radius are attributed with a certain weight. For points where its neighborhood is more spread out, it is given a bigger weight than for those more dense. Keypoints of the feature are selected if the local neighborhood has large variations.

Following the detection of the features and their location in the point set, a descriptor is used to describe them. The descriptors can be either local descriptors, global descriptors, or a hybrid of the two. For simplicity purposes, only local descriptors will be further explained, as they are the most commonly used. Local descriptors encode the geometric information, such as surface normal. Some of the commonly used methods for this approach are Point Feature Histogram (PFH), Eigenvalues-based Descriptors, and Signature of Histograms of Orientations (SHOT).

PFH encodes the local geometric properties of a point into a multi-dimensional histogram. This method analyzes the neighborhood in a given radius of a point and the variation with their surface normal is represented. For real-time applications where computational costs need to be reduced for increased speed, Fast Point Feature Histogram (FPFH). This method is similar to PFH, but while PFH connects all points in the radius, FPFH connects partially the points in a radius up to twice as big as the one used in PFH, as seen in Fig. 2.9.

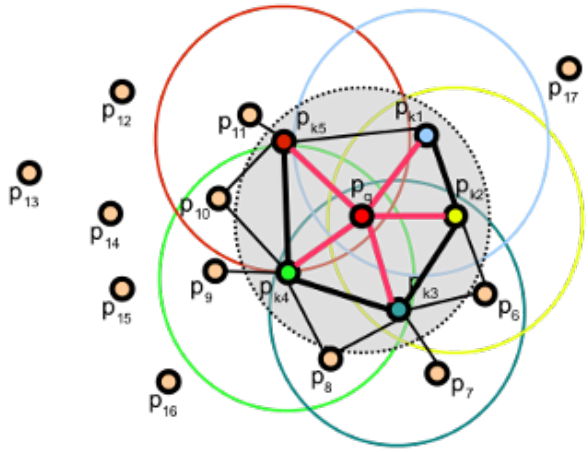


Figure 2.9: Fast Point Feature Histogram point connection [20].

Eigenvalues-based descriptors are used to extract pointness, curveness, and surfaceness through eigenvalues obtained from the decomposition of local covariance of a region [21].

SHOT, developed by Tombari *et al* [22], is a mix between signatures and histograms. This method exploits a spherical grid around a point, in which it is divided into 32 sections. The points that fall into a section are counted based on the variation between the surface normal of the neighboring points and the point's surface normal, which are then encoded into a local histogram of 32 bins, one bin for each section.

2.1.1.5 Segmentation and Classification

In many applications, especially in ones where the scanned object has a complex shape, it benefits from some type of segmentation so that it can be divided into multiple simpler components or if a point cloud contains multiple objects, it can be segmented by retrieving each object. This type of segmentation, or in other words semantic segmentation, has the objective of finding the relationships between points with regards to relative positions and locations, therefore aiding in localizing each component before a classification process is performed [23].

Point clouds can be further processed in order to classify each segmented component, e.g. in a scanning of a park, the benches, trees, and birds could be categorized accordingly to predefined classes. Concerning the classification process, the features extracted previously are run through a machine learning classifier algorithm to obtain a category.

According to [24], there are 5 methods for segmentation:

- **Edge based**: these methods identify the boundaries present in the point cloud in order to obtain segmented regions. This method allows for fast segmentation but is not accurate in the presence of noise or low-density point cloud regions.

- **Region based**: searches through the neighborhood to combine points with similar properties to form isolated regions. This method can be divided into seeded and unseeded regions. In the seeded region, a seed point will be picked from where the region will grow by adding points. This process is not adequate for real-time processing as it is time-consuming and its accuracy is dependent on the seed point chosen. In an unseeded region, all points are combined into a single region, and from there, it starts dividing into smaller regions. This method has the disadvantage of requiring knowledge of the object *a priori*.

- **Attributes based**: this method is based on attaining attributes, followed by the clustering of points based on those attributes.

- **Model based**: it is based on grouping points with the same geometric shape, such as spheres, planes, and cylinders.

- **Graph based**: views the point clouds as a graph. An example is considering that each vertex corresponds to a point and the edges connect to pairs of neighboring points. This method provides an accurate segmentation of complex shapes, even in the presence of noise, but is not adequate for real-time processing.

As for classification, usually, machine learning algorithms are used. From the features extracted, these are grouped into a vector and run through a Support Vector Machine (SVM) or Random Forest for semantic labeling.

2.1.1.6 Scan Registration

Scan registration is the process of optimally aligning two point clouds to the same reference system by finding the corresponding rotation and translation to form a representation of the scene or object. Each pair that is registered usually has an overlapping part. Registration is also a required step when there's a need to align different models of the same object.

One of the problems that occur when scanning an object or scene from a single viewpoint is that occlusions can occur. Besides, point clouds can't be acquired in their entirety from a single view so, to obtain a dense, accurate, and complete acquisition, it is required to scan it from multiple different views covering the whole scene or object. Subsequently, since each scan has its own coordinate frame, an alignment of the different point clouds should be done using a common reference frame which this process is referred to as point cloud registration.

The most known and used method for registration is Iterative Closest Point (ICP) and its many variants (Fig. 2.10). This method starts by finding correspondences between the two point clouds, followed by estimating the transformation matrix, containing rotation and translation, that minimizes the square errors of the distance between matching points. This is an iterative method that relies solely on the coordinates of the points and requires a previous initial alignment and if that alignment is not optimal, it may get stuck on a local minimum. For this reason, ICP is better used as a fine registration. A coarse registration should be done before fine registration. This is where an initial alignment is roughly performed through a global registration algorithm.

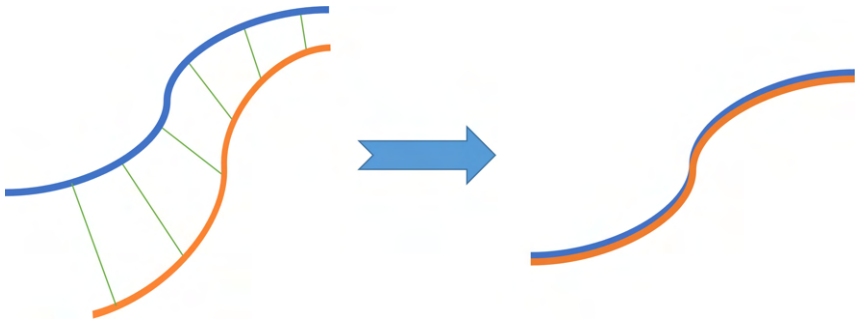


Figure 2.10: Iterative Closest Point method.

2.1.2 Public Point Cloud Datasets

To simulate and evaluate the framework, public point cloud datasets can be used to determine its performance on different objects. There are a few useful dataset repositories containing models of ships:

- **ShapeNet** is a large-scale repository that contains datasets of 3D shapes in many categories, made by researchers in Princeton, Stanford, and Toyota Technological Institute at Chicago (Fig. 2.11a).

- **ModelNet** contains a collection of 3D CAD models of point clouds, all pre-aligned to a reference frame (Fig. 2.11b).

(a) (b)

Figure 2.11: Examples of 3D models of ships. a) Model from ShapeNet [25]. b) Model from ModelNet [26].

2.1.3 Framework for Reconstruction using Images

The process of obtaining the 3D model through the use of simple RGB images differs from when using a depth sensor. Since RGB images only provide radiometric information, extra steps are required for extracting point clouds from them. In this subsection, it will be exploited the framework for the complete 3D reconstruction solely from RGB images.

2.1.3.1 Image Acquisition and Camera Calibration

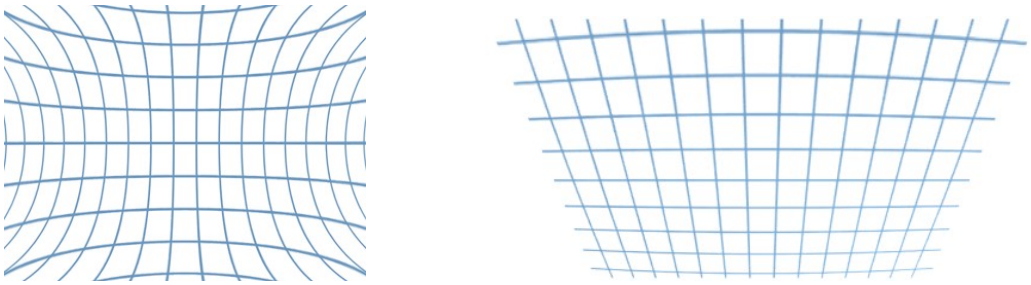
An RGB camera is a sensor that detects visible light and converts it to an electrical signal. This type of sensor is called a passive sensor, as it only detects input from its environment without transmitting any signals. Therefore, passive techniques are used in order to generate point clouds. These techniques are solely based on radiometric data, thus on intensity measurements of each pixel of the 2D image, so adequate lighting conditions are required.

The use of these sensors for 3D reconstruction is commonly used in applications such as the creation of virtual environments like cities, for tourism or gaming purposes [27, 28]. It is also widely used in the monitoring and inspection of structures [29]. Cameras have also been used in underwater reconstruction of shipwrecks [30].

Recently, RGB cameras have been increasing in popularity due to their much lower cost and size, making them easier to incorporate into any system. Also, they are not affected by extreme weather conditions such as snow, making them a good choice for outdoor environments when the weather is harsh and navigation is needed. Nevertheless, cameras also have their flaws. One major disadvantage is that they may be affected by lighting conditions, such as bright lights or heavy shadows, making the image imperceptible.

The images obtained from these sensors can suffer from distortions caused by the camera's optical lens, such as radial distortion (Fig. 2.12a), where straight lines appear to be curved in the image, or tangential distortion (Fig. 2.12b), where the image appears stretched and some areas look nearer than it should. For that reason, a process called

camera calibration is required. This process estimates the parameters of a camera, namely intrinsic parameters (e.g. focal length, optical center), allowing to map the pixel coordinates to the camera coordinates, as well as extrinsic parameters, to obtain the rotation and translation of the camera with respect to a world coordinate system, and even distortion coefficients.



(a) (b)

Figure 2.12: Effects of different distortions. a) Negative radial distortion. b) Tangential distortion.

The camera's intrinsic matrix K is given by (2.3):

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

where c_x and c_y are the optical center, in pixels, and f_x and f_y the focal length, in pixels, given by (2.4) and (2.5):

$$f_x = \frac{\text{focal length [mm]}}{\text{pixel width [pixels]}} \tag{2.4}$$

$$f_y = \frac{\text{focal length [mm]}}{\text{pixel height [pixels]}} \tag{2.5}$$

As for distortion, as mentioned previously, there are two most common ones: radial distortion and tangential distortion. To find the distortion coefficients, a method vastly used is the checkerboard method, in which an image of a rectangular checkerboard pattern with known dimensions is captured, its grid corners are extracted and a calibration procedure is applied. This method can also be used to determine the intrinsic parameters.

2.1.3.2 Point Cloud Generation

As mentioned before, to acquire the 3D location of an object in an image, a 3D location from a 2D image estimation-based technique is needed. The most commonly used technique is Structure-from-Motion (SfM), which is based on the estimation of the 3D object structure and extrinsic viewing parameters, i.e. camera pose, from a sequence of 2D images in different positions. This is done by extracting and matching features or points between

the multiple images. The steps in the framework of SfM will be discussed in more detail posteriorly.

Shape-from-Silhouette (SfS) is another technique, although not as used as SfM due to higher levels of inaccuracy. This technique estimates the 3D object structure using multiple silhouette images of the object. The silhouettes can be obtained by using contour-based extraction methods followed by background subtraction. The multiple silhouettes are then intersected to build an object with some degree of volume, resulting in an output called Visual Hull. This technique allows to easily estimate the shape of an object, especially in indoor environments as it is a more static scene. Although, for more complex shaped and textured objects, if the number of images is not enough, it can result in a very coarse approximation and not portrait accurately the object at hand.

2.1.3.3 Feature Extraction and Matching

The first step in SfM algorithms is feature extraction. In 2D images, features are denominated as visual features. The concept of feature and its utilities has already been explained, thus this section will focus more on the methods of extracting.

In 2D images, the features can cover:

- **Overall appearance**: in other words, the big picture. Histograms may be exerted in determining the distribution of intensity values without considering their spatial arrangement. To consider this arrangement, it is needed to involve moment theory and moment invariants to get a weighted average of the pixel intensities, with the objective of obtaining information about the shape of an object in an image and its pixels locations [16]. Besides using moments, a discrete cosine transform can also be used to acquire a coarse-to-fine representation.

- **Pixels attributes**: the intensity values are analysed in binary, RGB or HSI. Operations like mean intensity, standard deviation, or normalization are frequently used.

- **Texture**: local image regions are the focus and its description is based on the variation of intensity within a local neighborhood.

- **Shape**: has more focus on the contour of an object. This contour is normally detected by applying a filter mask to the image to find sudden changes in intensity levels. If a particular contour is known *a priori*, e.g. lines, circles, etc., parametric models can be used, such as the Hough transform.

- **Local features**: allows for individual identification of an object. To extract this feature, the change of intensity or texture across its neighborhood is tracked. This type of feature is the most commonly used.

The two most commonly used feature extraction algorithms are Scale-Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF). Other methods also famous

are Features from Features from Accelerated Segment Test (FAST) and Oriented FAST and Rotated BRIEF (ORB), but since SIFT and SURF are the most applied in the literature, more emphasis will be given to them.

SIFT was first proposed by Lowe [31]. This method starts by extracting the keypoints of a set of images to a feature vector. The advantage of the algorithm is its scale-invariant property, meaning that features can be detected regardless of their transformation, e.g. translation, scaling, or rotation. To do so, it applies the Difference of Gaussian (DoG) algorithm, followed by locating the local maxima and minima in order to determine potential keypoints. Secondly, to generate descriptors of those points, it uses a 3D histogram-based approach of the image gradients. Finally, it matches the features by finding correspondences across the images based on the Euclidean distance in the feature vector (Fig. 2.13).

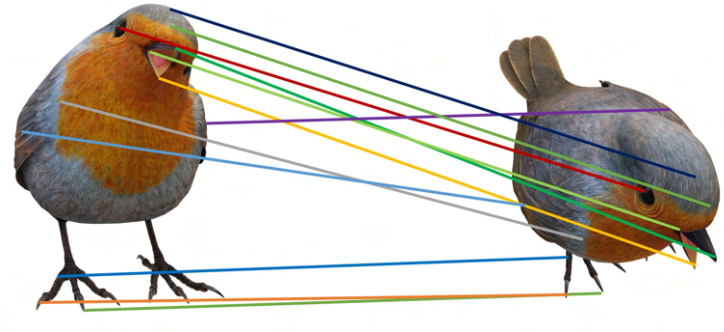


Figure 2.13: SIFT feature matching.

SURF was developed by Bay *et al* [32]. This algorithm is an improved version of SIFT, with increased speed and robustness. This algorithm, instead of approximating images with DoG to find scale space, uses box filters, increasing its speed. Also, to detect keypoints, it uses a blob detector based on the Hessian matrix. As for feature descriptors, it doesn't use a histogram-based approach like SIFT. Instead, it divides into regions the neighborhood of the keypoints, and the wavelet response of each region is used as a descriptor. In feature matching, it compares the keypoints and verifies if they have the same type of contrast (bright and dark blobs), described by the descriptors.

Regardless of the algorithm used for feature matching, it may contain noisy matches. To remove those noisy matches, RANSAC, explained in Sec. 2.1.1.4, can be applied in order to decide which are inliers and which are the outliers to filter out, thus improving the quality of the matching features vector.

2.1.3.4 Bundle Adjustment

Bundle Adjustment (BA) is an important step in the SfM process. It takes as input the features extracted and matched previously and computes the 3D points location, intrinsic parameters, extrinsic parameters, and camera pose that minimize reprojection errors, as illustrated in Fig. 2.14, thus when using this algorithm, the methods for camera calibration

explained in Sec. 2.1.3.1 are not required as the parameters are already computed in BA. The minimization of the errors is based on the non-linear least squares principle. The generated 3D points result in a sparse point cloud, as the algorithm does not provide 3D information for every pixel, that can be used as the final product or further densification of the point cloud can be performed, as explained in Sec. 2.1.3.5.

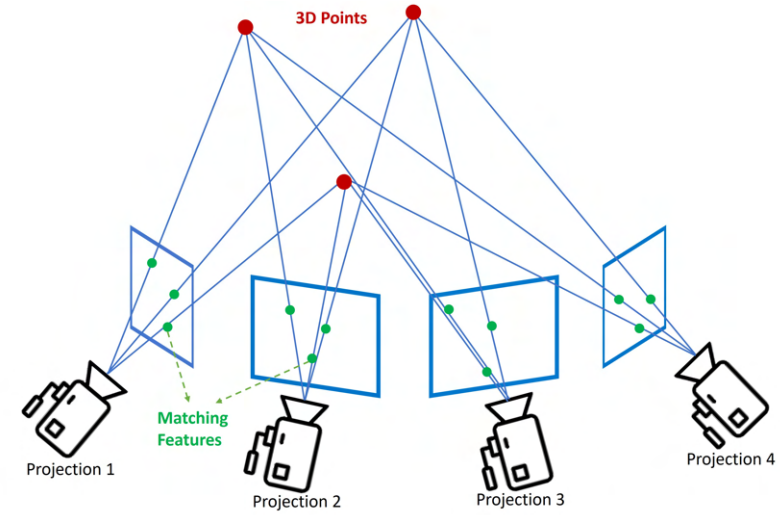


Figure 2.14: Illustration of Bundle Adjustment principle.

2.1.3.5 Dense 3D Reconstruction

Multi-View Stereo (MVS) is a stereo-matching technique that reconstructs the 3D object from multiple images taken from known viewpoints. Matching points across images can be quite challenging, especially when the camera's viewpoints are not known, thus that is the reason this method is usually applied after SfM, as it has already estimated the camera's parameters and pose. Consequently, this method is highly dependent on the quality of the images and the accuracy of the camera parameters, being only as accurate as those inputs.

This algorithm requires at least two images containing the same part or feature of the real object from different viewpoints in order to function properly, even though three or more images will provide a more robust reconstruction. As opposed to SfM, MVS generates 3D information for every pixel, obtaining a more dense point cloud as a result of reconstructing more 3D points. This method can be used as a standalone, as long as the camera parameters are previously known, but combining it with the sparse point cloud obtained from SfM and using it as a guide allows for a dense reconstruction with better results.

2.1.4 Sensor Fusion

Although a good alignment of the scans can achieve a relatively accurate representation of the 3D object or scene, fusing data from different sensors can add more information

to the model which, in turn, will assist in many different applications due to disclosing information that was not present in a scanning from just one sensor. For example, LiDAR can't detect transparent objects such as glass, while cameras can.

The simplest method of reconstruction using both RGB-D data and RGB is by following the framework described in Sec. 2.1.1 and 2.1.3, respectively, in order to obtain the desired point cloud for each and perform a scan registration of both point clouds, as described in Sec. 2.1.1.6.

A more efficient and faster method is performing a 3D/2D feature matching, where the extracted features from the 2D images are corresponded with the extracted features from the 3D point cloud acquired, followed by a co-registration based on projective scan matching. A method that can be used for scan matching is the Efficient Perspective-n-Point (EPnP), which has the objective of estimating the pose of the camera from a set of correspondences and subsequently obtaining the point cloud of the fused data via transformations, i.e. translations and rotations [16]. EPnP has the advantage of having high accuracy, high speed and doesn't require an initial alignment like ICP. Like in some methods seen previously, RANSAC can be applied afterward to eliminate outliers in the point cloud for increased robustness.

2.2 Machine Learning

With the advancement of technology and, consequently, the growth in the amount of data used in certain applications, Machine Learning (ML) has increasingly become more popular. It is a subset of Artificial Intelligence (AI) and by using the data available and training the algorithms, it produces a model capable of predicting or classifying images, possibly also reveling important characteristics in a dataset that were unknown, helping the humans in making data-driven decisions and making life easier. As seen in previous sections, ML is useful in computer vision, but in other applications also, such as speech recognition, surveillance, email filtering, search engines, etc. An important characteristic of ML is its ability to automatically learn from mistakes, without being explicitly programmed by humans to do so.

Deep Learning (DL) is a subset of ML. It has similar functions and many times it is interchangeably mentioned as ML, but it somewhat differs in a few aspects. One difference is that ML relies on structured data and statistical algorithms to make predictions, while in DL, the human brain is mimicked, thus it is created a neural network capable of learning and making decisions independently. Furthermore, DL requires a lot less human intervention than ML. In addition, for DL to work properly, it requires a lot more data for the training process than ML.

2.2.1 Convolutional Neural Networks

Recently, DL has received great attention due to its ability to encode certain properties into architectures such as Convolutional Neural Network (CNN). Part of this attention

also comes from the fact that these algorithms can achieve better results in object detection than humans in certain images.

CNN, also known as convolution nets, is a deep learning algorithm specialized in analyzing images. It is composed of several layers, each with its own purpose. The first layer, also known as the edge detection layer, detects features such as horizontal or diagonal edges. The next layers use the output of the previous layer to extract simpler shapes and progress to more complex shapes as it progresses in layers. The last layer outputs a confidence score of the probability of an image to belong to a class. If more than one class is available, a confidence score is given to every class, being that the highest score will result in the image belonging to that same class.

CNN has three different types of layers (Fig. 2.15):

1. Convolution Layer.
2. Pooling Layer.
3. Fully Connected Layer.

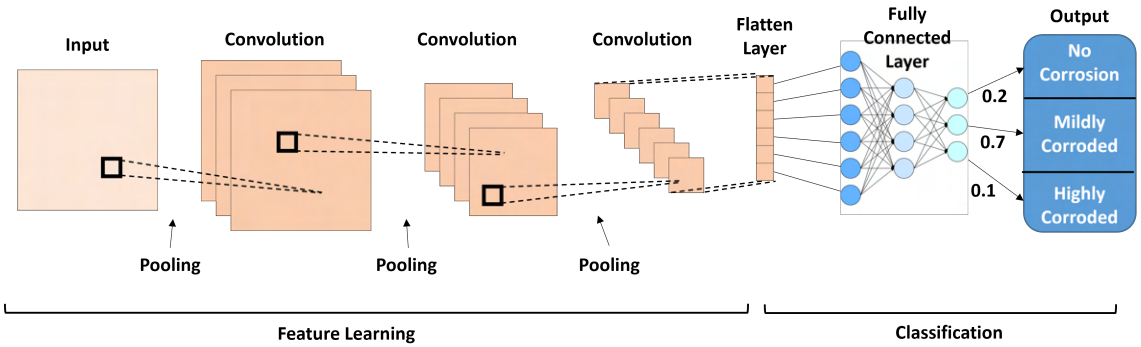


Figure 2.15: Example of a CNN architecture.

An image is processed by the computer as a data stream, represented as a matrix of pixels divided by channels, where, in the case of an RGB image, 8 bits are used for the blue channel, plus 8 for the green channel, 8 for the red channel and, in more modern computers, an additional 8 for transparency, also known as the alpha channel. Each pixel of the image is fed to the input nodes of the network.

In a convolutional layer, a filter with a certain kernel size is applied to the input image, sliding over it, where its result is added to a feature map. Convolution filters allow edge detection in images and enhance other characteristics [33]. Each kernel serves to determine specific features of an image, thus by training the model to a specific dataset, the kernel values are determined in order to detect the desired features, regardless of orientation and scaling, making it efficient in object detection. The output of a convolutional layer is the resulting feature map.

Usually, the initial convolution layers are in charge of extracting edges from the image. The middle layers exploit those detected edges to extract more complex shapes. Lastly, the final layers extract more distinctive features from the image, such as faces for example.

The output of a convolutional layer is used as input in pooling layers and its objective is to simplify it, keeping the complexity of the architecture of the network manageable, as it reduces the number of convolutional layers required later, therefore reducing training time and additionally, avoid overfitting. To do so, it performs similarly to the convolutional layer, in the sense of applying a filter over the feature map to summarize the features, reducing its size and saving computational efforts later on. There are mainly two types of pooling layers: max pooling and average pooling. Max pooling consists in extracting the maximum value of a feature map and it is the most commonly used technique. Average pooling consists in computing the average value of the feature map.

After the last convolutional or pooling layer, comes the fully connected layer, also known as the dense layer. The output of the convolution or pooling layer is flattened into a 1D array and fed into the dense layer. This, in turn, is in charge of computing the confidence scores of each class, therefore this layer is composed of as many outputs as the number of classes.

There are many different architectures of CNN with different parameters. For example, in the AlexNet architecture [34], input images need to be reshaped into a size of 227 x 227 x 3. The most famous architectures are:

- AlexNet [34];
- GoogLeNet [35];
- Inception [36];
- Efficient Net [37];
- LeNet [38];
- ResNet [39].

CNN has many advantages, including great accuracy in image classification, does not require human supervision, and can be used in a wide variety of applications, such as part of an image detection algorithm, speech recognition, etc. On the other hand, a significant amount of training data is required for it to be accurate, the training process can be slow specially when its architecture is composed of multiple layers, and they fail to classify images with different positions unless it is trained to, increasing even more the required data through the use of data augmentation.

2.3 Literature Review

In this section, a review of the literature about the different methods of 3D reconstruction and defect detection, such as corrosion, is done, as well as a brief summary of the major strengths and weaknesses of each.

2.3.1 3D Reconstruction using Different Methods

Relevant scientific work done on 3D reconstruction of ships, or any other type of watercraft, could not be found during the research process and, consequently, the focus had to be shifted to another direction, namely 3D reconstruction of other types of structures. Nevertheless, the techniques applied in these other studies can provide inspiration for a solution.

The most common application of 3D reconstruction is in the generation of urban models, such as buildings, bridges, trees, power lines, and roads. The objective of creating these models is to analyze their structural health without requiring a human inspector to manually perform such a task, as it is hard and inefficient.

Liang *et al* [40] proposed an LSQ fitting algorithm to reconstruct multiple power lines from a point cloud obtained via ALS. The process started by extracting the points belonging to the power lines, by first classifying ground points and non-ground points, such as buildings, vegetation, and power lines, followed by manually delineating the lines' points, and finally, extracting them automatically using the RANSAC algorithm, which determines the points belonging to the lines. Lastly, as a power line resembles a quadratic polynomial, this model was used to fit each line, solving some missing gaps. This solution presented great results, with a 97% accuracy. However, the accuracy is strongly reliant on the density of the point cloud so, for sparse or incomplete point sets, its accuracy drops to a much lower level.

Cheng *et al* [41] proposed a framework for reconstructing a large interchange bridge based on dividing it into smaller, simpler structures. The framework consists of five steps. The first step is to automatically extract the bridge points from the rest of the point cloud using the Reversed Iterative Mathematic Morphological (RIMM) method. In the second and third steps, the extracted bridge is divided into structure units by using voxel segmentation to find the connected bridge structures, followed by a 2D analysis, in order to compute the skeleton of each structure and divide it into smaller units. Step four is where the occluded structures are restored automatically, using 2D line fitting and height interpolation. Lastly, in the final step, the 3D model is restored. The results of this work were quite satisfying, as the geometric shape of the bridge was preserved and the level of completeness and fullness were very high, meaning that the occluded parts were well restored without requiring the help of a human. The only issue with this solution relates to the edges of the bridge, where it had a few inconsistencies due to applying the RIMM method.

Wu *et al* [42] proposed to reconstruct 3D building models from contours based on graph-based localized contour tree and bipartite graph-matching theory. This method relies solely on contour information. These contours are obtained using a graph-based localized contour, which will after be analyzed and represented by a contour tree and, subsequently, be divided into several sub-trees, corresponding to different parts of the building. For the bipartite graph-matching, a weighted bipartite graph is built for each two adjacent contours to solve the correspondence problem, which will then specify the geometry. Lastly, the model is reconstructed. This proposed work has shown good results in complex-shaped buildings, especially when the defined contour intervals are small. As this interval rises, the model errors increase accordingly, causing a loss of small details. Choosing to preserve all details by choosing a smaller interval will result in high computation power, which would be expected. Besides, this technique produces some geometric distortion, due to deviations between the contour and raw point clouds.

Pan *et al* [43] performed a semi-automatic 3D reconstruction of heritage bridges using photogrammetric point clouds. It started by capturing the images via a UAV, followed by camera calibrations, SfM and MVS in order to obtain a dense point cloud. Afterward, it was proposed a novel segmentation method on supervoxel structure and global graph optimization, where different elements of the bridge, such as fences, bushes, pier, base, and deck, were segmented based on their geometric features. This process was then combined with a rule-based classification method, where each segmented element of the bridge was labeled based on its saliency. Finally, a Poisson surface model was applied to reconstruct surface models. The work proposed by these authors achieved a high accuracy in terms of the reconstructed point cloud as the error of reconstruction could be as low as 0.4% and as high as 4.6%, with both extreme percentages still quite low. As for segmentation, the results were not good when the point cloud density was low or suffered from noise, so the boundaries of the elements did have some errors. Lastly, the precision of the recognized and labeled elements was decent, reaching an overall accuracy of 83%.

Qu *et al* [44] proposed a fast method for reconstructing models from image sequences acquired from a UAV. The first step in this method is by using a fully automatic approach to build an image queue from key images of a video, where two consecutive images must have a good overlapped area. This area is determined by feature correspondence, in which the features are detected via SIFT and then compressed based on Principal Component Analysis (PCA) to reduce computational costs during correspondence. From PCA, Principal Component Points (PCP) can be extracted and each point displays the distribution of the feature points in the images. The correspondence between PCPs from two consecutive images allows to determine the overlapped area. Subsequently, the SfM algorithm is applied to the queue of key images, in which a weighted bundle adjustment is also applied. The queue is after updated by deleting N images from the front of the queue and adding N images to the end. This process is repeated until all images have been processed. Lastly, MVS is used to obtain a dense reconstruction, namely the depth-map fusion method. As for results, this method achieved a relative error inferior to 1% and a

speed of reconstruction significantly faster than some other methods. The downside of the proposed method is the distortion of the model for flight distances greater than 300 m caused by accumulated errors.

Xu *et al* [45] proposed a framework for reconstructing a cultural heritage using both TLS and images taken from a UAV. The process starts by obtaining the point cloud from the images using the SfM algorithm. For feature extraction, SIFT is used, but since it can result in memory explosion due to many high-resolution images, each image is divided into smaller blocks, extracting features from each block and then combining them. During feature matching, to reduce the computational cost, it was proposed a camera network in which the airborne flight control data was used, also reducing the number of false matches between correspondences. Next, bundle adjustment was applied, obtaining a point cloud with RGB information.

Proceeding to co-registration with the TLS point cloud, a coarse registration was first applied, where the model with the highest accuracy and resolution was used as a reference and each set was separately oriented to user coordinates through a Bursa transformation model, followed by a fine registration, in which a variant of ICP known as Iterative Closest Compatible Point (ICCP) is used, which performs better at rejecting incorrect correspondences than the traditional ICP. The resulting point cloud obtained from images had a model error between 8% and 12%, which is not the greatest, and the point cloud from TLS has an error of around 1% in the range of 150 m, clearly seeing the difference in the accuracy of both methods separately, but when combining both data, the error remained at 12%. The advantage of this method of combining data relies solely on acquiring data through a UAV in zones that TLS cannot scan, such as rooftops.

2.3.2 Defect Detection using Deep Learning

Similar to 3D reconstruction, scientific research done for defect detection specific to ships and vessels is scarce. Thus, research was performed in steel structures, due to its similarities with the vessel's material.

Bastian *et al* [46] proposed a CNN architecture to detect, in images, four different classes of corrosion in pipelines: no corrosion, low-level corrosion, medium-level corrosion, and high-level corrosion. To train the model, a manual collection of videos of pipelines with different levels of corrosion were acquired, in which some frames of the videos were extracted as samples. Some extra techniques were applied to those images in order to obtain a more rich data set, such as horizontal and vertical flipping, rotation, and varying resolution of the images. For each image, the level of corrosion was attributed manually. For classification, a custom CNN architecture was developed, composed of five convolutional layers, five pooling layers, and four fully connected layers. Next, for localizing the corrosion in the image, a recursive region-based method was applied, where if an image was classified with medium-level corrosion or higher, it would be divided into four regions and each region would be classified, and if the region had a corroded area,

Table 2.1: Comparison of strengths and weaknesses of the literature in different acquisition methods in 3D reconstruction.

Authors	Sensor	Model	Strengths	Weaknesses
Liang *et al* [40]	Laser Scanner	Power Lines	Makes usage of well documented algorithms	Only accurate for dense point clouds
Cheng *et al* [41]	Laser Scanner	Bridge	Divides complex problem into smaller problems, handles automatically and with great accuracy the occluded zones	Slight loss of information on edges
Wu *et al* [42]	Laser Scanner	Building	Divides complex shapes into more manageable shapes	Geometric distortion
Pan *et al* [43]	Camera	Heritage Bridge	High accuracy of reconstructed model	Not as accurate for zones where point cloud is sparse
Qu *et al* [44]	Camera	Misc	High accuracy of the reconstructed model, fast speed of reconstruction, generic algorithm	Accumulated error are too much for UAV flight distances greater than 300 m
Xu *et al* [45]	Camera and Laser Scanner	Cultural Heritage	Makes usage of well documented algorithms	Only as accurate as the worst accurate point cloud

it would be further divided. This process is repeated until a defined subregion size was achieved, resulting in a delineation of the corroded area, replacing the common bounding box method. This entire procedure achieved great accuracy in detecting the presence of corrosion, with a 98.2% accuracy, and a fast speed in localizing the corrosion area in the image, as it took less than one second to do so, making it suitable for real-time processing.

Cha *et al* [47] used Faster R-CNN to detect and classify five types of defects (steel delamination, medium-level steel corrosion, high-level steel corrosion, bolt corrosion, and concrete cracks) in almost real-time in videos. For creating a dataset for training, different images with the defects present at different lighting conditions were taken, and data augmentation was applied where images were flipped horizontally. Each image was also classified manually. This method showed to be adequate for real-time processing.

Table 2.2: Comparison of strengths and weaknesses of the literature in defect detection.

Authors	Model	Strengths	Weaknesses
Bastian *et al* [46]	Pipelines	High accuracy	Creates new CNN instead of using making usage of existing ones
Cha *et al* [47]	Misc	Uses well documented algorithm, able to recognize multiple defects	None to be reported

3 Proposed Model

The objective of this dissertation is to develop a system capable of generating a single 3D model of a ship, from the merging of two other 3D models generated from images and readings of an RGB-D sensor, as well as detecting corrosion present in the ship and representing it on the final model. Another focus of this dissertation is the speed and accuracy with which this final model is achieved, meaning that a high accuracy with a low computational cost is what is intended. This chapter provides a high-level explanation of the proposed model created to achieve the goals.

In Sec. 3.1, a general overview of the framework to achieve a 3D model of a ship incorporated with corrosion detection is given. In Sec. 3.2, the process of obtaining the 3D point cloud from images through SfM is explained. In Sec. 3.3, the approach taken to scan the ship using a RGB-D sensor is described, complementing with the process of generating the model. In Sec. 3.5, the procedure used to generate a single model from the merging of the models generated from SfM and RGB-D is described. Lastly, in Sec. 3.4, the algorithm to detect corrosion in images and represent them in the model is explained.

3.1 General Overview

As mentioned previously, achieving an accurate model with low computational cost, meaning in less time, is what is intended. To do so, the methods chosen were carefully thought-out in order to achieve so.

The system is composed of three main components: a UAV, a LiDAR, and a RGB camera. Both the LiDAR and the camera will be mounted on the UAV, which will survey the ship while taking pictures and recording data from the LiDAR.

An overview of the architecture of the proposed model is depicted in Fig. 3.1. As it can be seen, the architecture is composed of seven main blocks:

- **Structure From Motion**: from the images obtained during the scanning of the ship, it extracts the 3D point cloud.

- **Register Consecutive Point Clouds**: the multiple RGB-D images are converted point clouds that need to be merged into a single point cloud.

- **Point Cloud Processing**: point clouds usually contain outliers and noise that need to be removed. This block is in charge of filtering the point cloud, as well as further process to achieve an accurate model.
- **Image Division**: each image used in SfM is divided into regions for later classification.
- **Image Classification**: Each region previously obtained goes through a classification process to determine if corrosion is present.
- **Corrosion Representation**: From the regions classified as corrosion, the points in the point cloud generated from SfM belonging to those regions are highlighted in a different color in the model, to represent the corroded parts.
- **Model Fusion**: the model obtained from SfM and the model obtained from RGB-D are merged into a single model, with the objective of improving accuracy.

3.2 3D Model from RGB Images

As mentioned previously, SfM is a photogrammetry framework that allows to extract 3D information from 2D images. The process of obtaining a 3D point cloud through SfM goes through many stages, as shown in Fig. 3.2.

Before starting to acquire images, the intrinsic, extrinsic, and distortion parameters of the camera must be determined. The first two are important for SfM, as they allow to deduce the geometric information of the scanned ship. The latter allows to fix images from possible distortions present, such as radial and tangential, depending on the camera model (fisheye, pinhole, etc), resulting in a better representation of the scene.

Since not all cameras provide these parameters, a calibration has to be performed, in order to extract them. As explained in Sec. 2.1.3.1., a common way to calibrate is by using a checkerboard pattern with known dimensions and taking pictures of it from different views. This allows to extract focal length, optical centers, principal points, and distortion parameters of radial and/or tangential distortion.

Subsequently, to obtain a high-quality 3D model, image acquisition has to be done, as this is a core step of the process. The images acquired need to have high quality and there must be some overlap of the scene between consecutive images. Lighting conditions are a major factor, as they should be ideal, therefore avoiding the presence of heavy shadows and bright lights is fundamental. Usually, ships are placed in an open outdoor environment, thus the best way to achieve this is by scanning it on a cloudy day. Furthermore, complex shapes require more images to be taken from different views to obtain as much detail as possible.

Succeeding image acquisition comes feature extraction. The objective of feature extraction is to detect a group of key pixels and describe those features based on their overall

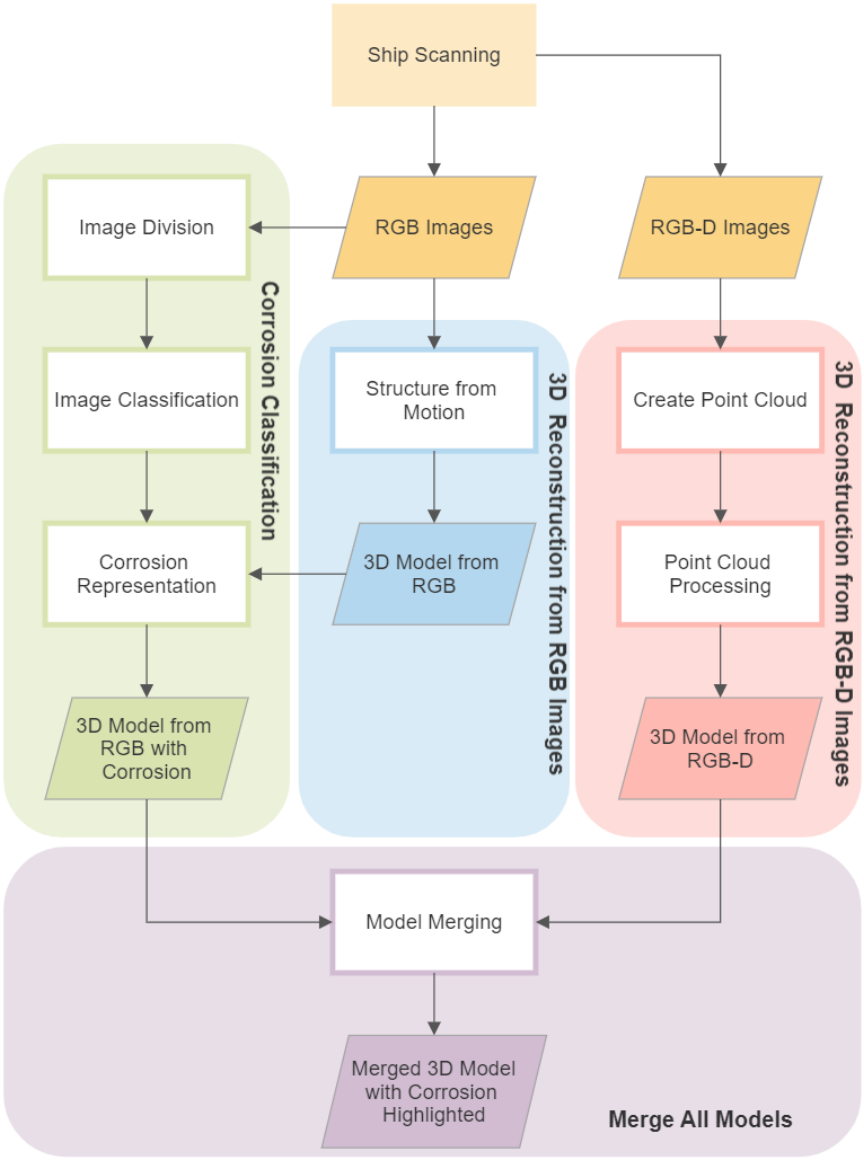


Figure 3.1: Chart of the proposed method for 3D model acquisition and corrosion detection.

appearance, pixel attributes, texture, shape, or local features (Sec. 2.1.3.3). Therefore, a feature belonging to a part of the ship should have similar descriptors in all images that capture that part. The descriptors, ideally, are invariant to scale, transformation (rotation and translation), and illumination. In the end, each feature detected is accompanied by a descriptor vector.

Feature extraction is considered a dimensionality reduction process, as the data size is reduced to a set of only the important aspects that represent that data. Reducing the data size to be processed will, consequently, reduce the computational cost, as intended. Even though the data is significantly reduced with this process, the number of features extracted can still be significant, thus a further filtering process may be required to limit the number of features, depending on the number of features detected.

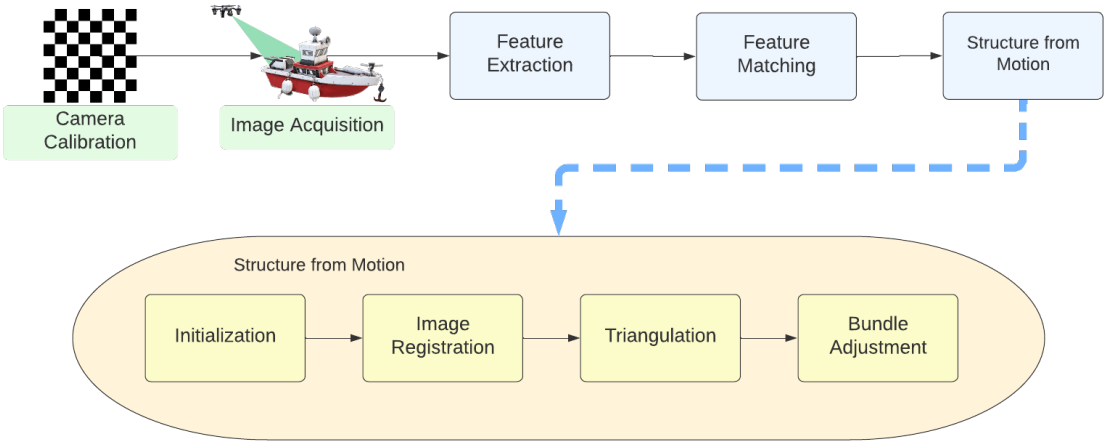


Figure 3.2: Full workflow for acquiring 3D point cloud through SfM.

The following step to feature extraction is feature matching. This consists of matching features between images based on their descriptors. The effectiveness of this step highly relies on the features extracted previously, so an appropriate algorithm for the problem at hand must be chosen for the best results. The feature matching algorithm starts by taking the features of a source image and comparing them to a feature in the target image and repeats this process to all features in the target image, matching features with similar descriptors, resulting in what are called corresponding features. Usually, it is assumed that one feature of the target image can only match one feature of the source image. Lastly, a filtering process is applied to eliminate false correspondences based on determinate parameters chosen, usually by applying the RANSAC algorithm.

Finally, the Structure from Motion process itself is applied to deduce the 3D geometric scene. This process consists of four iterative steps [48]:

- **Initialization**: an initial image pair is chosen as the seed of the process. Choosing the most appropriate image pair is essential as it dictates the accuracy of the resulting 3D structure. For better results, the image pair should be chosen based on the presence of a physical zone of the ship where many camera views capture it, therefore, the images chosen must maximize the number of matches.

- **Image Registration**: having chosen two initial images, new images are iteratively added by solving the Perspective-n-Point (PnP) problem, taking leverage of the previously determined feature correspondences and the triangulated points of the images that were already registered, in which the camera pose is estimated for each image, assuming that the camera is calibrated. A camera pose is composed of 6 degrees of freedom, 3 corresponding to rotation and 3 corresponding to translation.

In order to determine this transformation, (3.1) must be solved for n points present:

$$X_c = K \left[R|T\right] X_w$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1}$$

where K is the matrix of the intrinsic camera parameters, namely f_x, f_y as focal length, c_x, c_y as principal point, and γ as the skew parameter, where R and T are the rotation and translation, respectively, $X_c = \begin{bmatrix} u & v & 1 \end{bmatrix}$ is the image point, and $X_w = \begin{bmatrix} x & y & z & 1 \end{bmatrix}$ is the point represented in the world frame. A common way to solve this problem is by choosing 3-point correspondences (P3P) when the sum of the point correspondences has low levels of noise. For higher levels of noise, EPnP is used, where at least 4-point correspondences are needed.

- **Triangulation**: the 2D features are triangulated into 3D points. Since triangulation inserts redundancy in the process, as many images can contain the same triangulated point, even though it is not the most efficient way in terms of speed, it does increase accuracy as it improves camera poses determined during image registration, thus, for this project, it was chosen to abdicate a bit of computational speed for some accuracy.

- **Bundle Adjustment**: as image registration and triangulation can insert uncertainties that affect one another, an additional process, known as Bundle Adjustment, needs to be performed to refine the 3D points, minimizing the reprojection errors by projecting the triangulated points to the image space and compute a loss function that will be used to filter out observations with high reprojection error. This minimization is performed by applying a non-linear least squares algorithm, such as the Levenberg-Marquardt.

3.3 3D Model from RGB-D Images

Originally, the idea for this project was to use a LiDAR for 3D reconstruction, but since this sensor was unavailable, an RGB-D camera was used instead. These sensors provide different types of data, as LiDAR provides point clouds and RGB-D cameras provide both RGB and depth images. Fortunately, there are ways to convert RGB-D images in colored point clouds, thus achieving the same purpose, albeit with some degree of accuracy lost. The difference between LiDAR and RGB-D sensors is that while the former emits beams of light and measures the time it takes to return, the latter measures the intensity at which it arrives. Furthermore, the latter costs significantly less and provides more information,

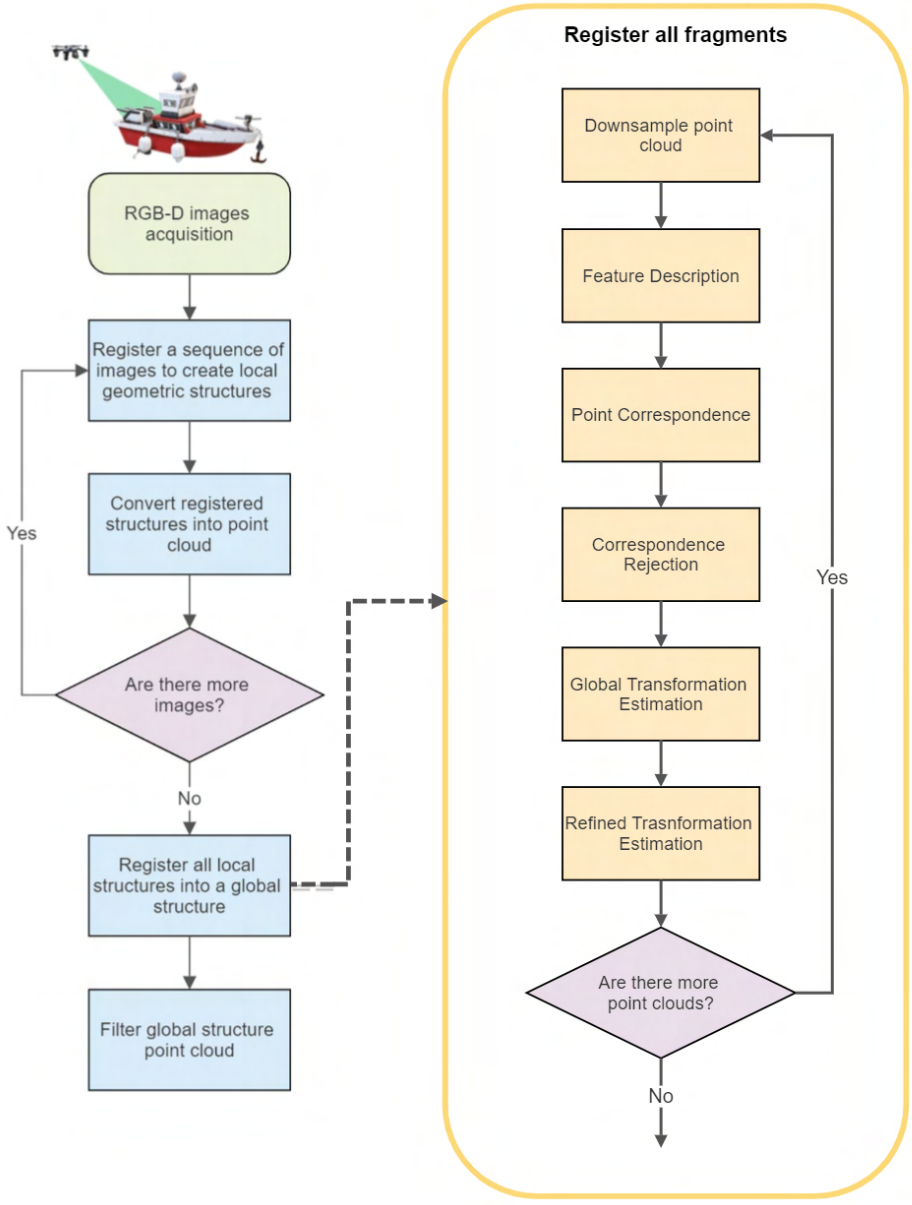


Figure 3.3: Full workflow for acquiring 3D model from RGB-D data.

such as color. The disadvantage to this type of sensor is its increased susceptibility to motion blurs and decreased accuracy.

Nevertheless, RGB-D sensors are a great instrument to capture the geometric and radiometric information of a physical object, as it is more accurate for 3D reconstruction than simply with RGB images. Even so, data collected from them require some form of processing to achieve the best results possible.

The objective of this part of the project is to create a partial model of the ship that enables to fill of potential holes and adds density to the model obtained from SfM, after merging. This section focuses only on the reconstruction of the partial model, as the merging of both models will be seen in a later section.

As proposed in Fig. 3.3, to reconstruct the ship, a survey of it must be performed and RGB-D images of it must be extracted sequentially. After, a sequence of images is registered and converted into point clouds, reconstructing portions of the ship, thus creating local geometric structures. Since RGB-D cameras capture a wide view, some structures will have a significant overlap which will be critically important in a later step.

After all the images are registered and extracted point clouds of the local geometric structures, all point clouds are registered in a global space using the traditional point cloud registration workflow, in which each step requires a source and a target point cloud:

- **Filtering:** downsampling is applied in order to reduce the number of points and consequently, the computational cost.
- **Feature description:** all the points in the downsampled cloud are described according to its properties and its neighbors.
- **Point matching:** the keypoints from the source cloud are matched with similar ones from the target cloud based on their feature descriptors.
- **Correspondence rejection:** since the matched keypoints might have false matches, a filtering based on a certain parameter, such as distance or neighborhood, is applied.
- **Transformation estimation:** lastly, the transformation from the source cloud to the target cloud is determined. First, a global transformation from the matched keypoints is estimated and afterward, a refined transformation is estimated, using ICP, increasing the accuracy of the final model.

With the global structure obtained, a final filtering process is performed. The objective of this filter is to remove any unwanted structures surrounding the ship, as well as filter possible artifacts and outliers in the ship model itself caused by the reconstruction process or sensor inaccuracies and noise.

3.4 Corrosion Classification

The intended original proposal of this project was to perform corrosion detection using a multispectral camera instead of corrosion classification with an RGB camera, but due to the lack of publicly labeled datasets for both RGB and multispectral images, and the time consumption it takes to create a new dataset for such an irregular aspect object, it was not possible to carry out this idea, as the time for completing the entire project at hand would not be possible if so. Therefore, the alternative method chosen was to just execute the corrosion classification and represent them in the point cloud. The process of classification goes through some steps, as depicted in Fig. 3.4.

The first step of the proposed workflow is to create a dataset. Since the objective is to only classify one object, this type of classification is considered binary, where the options

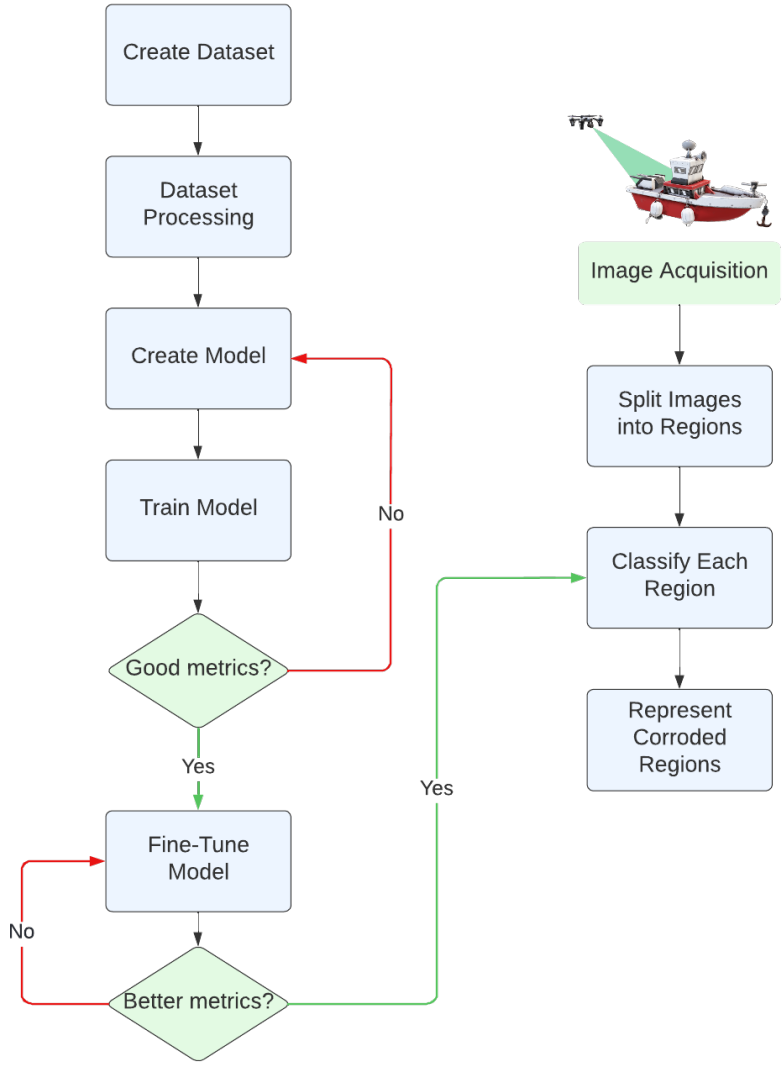


Figure 3.4: Full workflow for classifying and representing the corroded parts of the ship in the model.

are either "corrosion" or "no corrosion". To obtain the most accurate model, it is important to have a diverse dataset composed of images with corroded parts from different scenes and lighting conditions, as well as another dataset with images where no corrosion is present.

Afterward, the datasets are processed. Firstly, both datasets are split into training, validation, and test sets. The training set will be used during the model training to fit its parameters and, usually, is the set that contains the most number of images from the split. The validation set will also be used during the training, but it is not part of the fitting process. Instead, it is used as an evaluation of how is the model fitting to the training set. This dataset also plays a big role in regularization as it can prevent over-fitting the model by stopping the training process earlier. Lastly, the test set is used after the training

process has stopped, and, similarly to the validation set, it is used to evaluate the fitting of the final model's parameters.

After splitting the datasets, it is good practice to apply image augmentation on the training dataset. This data augmentation introduces diversity and increases dataset size which, as mentioned previously, will result in a more accurate model. There are many types of image augmentation such as flip, rotate, scale, etc, but when applying to the images, it must be a realistic transformation to the real-life scenario, because if not, it may decrease the accuracy of the model.

The next step consists in creating a model. Here, transfer learning is exploited, which consists of an architecture with a base pre-trained model being retrained on another task, which in this case, is corrosion. The reason for this choice revolves around the simplicity of it, as it's not necessary to create an entirely new model and, most importantly, be able to utilize the knowledge previously learned and apply it to this new problem. Consequently, this results in less training time and better accuracy of the final model.

Following the model creation, it is trained for a given number of epochs. This number can be cut shorter by evaluating the metrics, during training, of the loss function of the validation dataset, as mentioned before, therefore avoiding overfitting. Following this process, the model is evaluated with the test dataset, in order to determine if a better fitting could be achieved through the tuning of some parameters.

Subsequently, a fine-tuning of the model is performed to increase its performance even more. As previously only a small portion of the layers of the architecture is trained, the process of fine-tuning will update the weights of the base model through training, enabling the entire model to be more specific to corrosion classification, hence the better performance. Like the first training, an evaluation must be performed to determine if it's required further tune the parameters to achieve better results.

All the previous explanation of the proposed model for corrosion detection only corresponds to obtaining a deep-learning model capable of classifying corrosion. The second part of this workflow is related to the classification *per se*. The objective is to represent the corroded parts in the 3D model. Most images taken during surveying will likely capture large portions of the ship, which can lead to big portions being classified as corroded when there could only be a small part that is actually corroded. Therefore, the solution proposed was to divide each image into a predetermined number of regions and classify each. Afterward, the image regions that were classified as corroded are represented in the point cloud as such.

3.5 Merging Point Clouds

To improve accuracy and possible occlusions that either of the previously mentioned models may have, merging both point clouds will result in a more accurate and full model. Although, a few problems arise when trying to merge point clouds from different types of sensors [49]:

1. **Density variation:** the models from each sensor will have different variations in density, where one point cloud could have significantly more points than the other point cloud;

2. **Scale variation:** the scale for the RGB-D sensor is in the metric scale, but the scale of SfM is unknown.

3. **Noise:** the noise that each sensor produces is different.

4. **Viewpoint variation:** the viewpoint is different between sensors as one is used from a higher ground while the other is used from a lower ground.

The feature-based registration method explained in an earlier section will not work, as it requires the points to be the same in order to match the ones with similar features and neighbors. Thus a different method is required. The proposed method is to use PCA to scale and orient the models. This technique relies on dimensionality reduction in which all the points and their information, such as location, are reduced into a smaller set of information that still contains most of the information of the larger set. The purpose is not to reduce the number of points but encode all the points in a smaller set of information. From this analysis, we can extract the principal components, which represent the direction of the points where there is more variance. Since the point clouds are represented by 3-dimensional information (x, y, and z location), it equates to 3 principal components, thus the 3 directions where exists more variation between the points are the direction in which the principal components will point to. These components can be viewed as axes, which can form a coordinate system (Fig. 3.5).

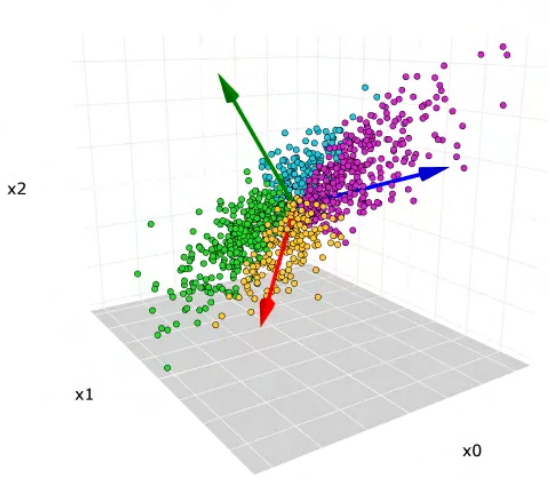


Figure 3.5: Example of principal components for a 3D data [50].

After computing the principal components of the point clouds, its 3D center location will be known as well as the axes directions for each. With this, it is possible to create a tight bounding box around each one and compute the difference in size between each, thus scaling the SfM model to the same size as the RGB-D model, as the latter will have a

more accurate scale. Afterward, a transformation composed of a translation and rotation can be done to translate the centroids to the same location and then rotate one of the models so that both models' principal components are coincident.

Subsequently, after the point clouds are aligned, a merging of the point clouds is performed, by concatenating the data into a single variable and then converting it into a mesh. Since it is required to preserve the pure blue points representing corrosion, when converting into a final mesh, the SfM model will dictate the color of it.

4 Implementation

This chapter has the objective of describing the setup (hardware and software), and methods and algorithms developed to fulfill the proposed model explained in the previous chapter, as well as providing the reasons for each choice.

In Sec.4.1, the hardware and software used are described, as well as some of its important characteristics. In Sec. 4.2, an overview of how the data was collected is shown. In Sec. 4.3, the methods to obtain a 3D model through SfM are thoroughly explained. In Sec. 4.4, the algorithm developed for obtaining a 3D model through RGB-D images is described, as well as the method for adapting the proposed model from using LiDAR data to using depth images is clarified. After, in Sec. 4.6, the steps performed to merge the two point clouds from the two different sensors are described. Lastly, in Sec. 4.5, an in-depth explanation of the algorithm created for corrosion classification is given.

4.1 Setup

As most of the methods are highly correlated with the available material for developing this project, the setup was chosen first. This material also has a huge influence on both the accuracy and speed of the algorithms, so these metrics are somewhat dependent on their performance.

4.1.1 Hardware Infrastructure

For the scope of this project, a UAV was used to survey the ship, more concretely, the DJI Phantom 4 Multispectral (Fig. 4.1). This vehicle was used for RGB and multispectral image acquisition of the ship. It is equipped with a highly sensitive GNSS and a camera composed of six sensors, where one is an RGB sensor and the remaining five are monochrome sensors for multispectral imaging. The resolution of each sensor is 2.12 MP. It is mounted with a micro-SD card for retrieving the images taken after the scanning is complete [51]. Furthermore, the RGB-D sensor used was Intel Realsense Depth Camera D435i [52], with an integrated IMU, a fairly accurate depth accuracy of up to 2 meters, and an RGB camera.

The reason for using this specific UAV is mainly due to the incorporated multispectral camera. Even though this dissertation wasn't going to be using multispectral images for

Figure 4.1: DJI P4 Multispectral [53]

corrosion classification, as initially intended, the idea wasn't discarded in case there was enough time for adding this feature to the project.

4.1.2 Supporting Tools

To develop the methods mentioned in the previous chapter, the following tools were used:

- **Google Colab**: it's a tool that allows to write Python code online. The reason behind the choice of this service is due to the GPUs provided by Google for processing and its simple use, as it doesn't require anything to be configured [54];
- **TensorFlow**: this is an open-source library that allows the creation of machine learning applications, although its main focus is on the training and inference of deep neural networks. On account that it has already many in-built tools for creating an ML model and is easy for prototyping, it was the elected tool for creating a model for corrosion classification [55].
- **Roboflow**: is a framework that enables the user to create a fast and easy computer vision pipeline. It allows for dataset preprocessing, annotation, model training, etc. Due to its user-friendly interface and since it can process datasets easily, it was chosen to preprocess the datasets acquired [56].
- **Meshroom**: Meshroom is an open-source 3D reconstruction software that is able to obtain 3D models from images. This software allows to analyze and compare many different algorithms and pipelines for reconstruction, while barely requiring user interaction. This software was used for creating a 3D model through SfM [57].
- **Point Cloud Library (PCL)**: it is an open-source library for point cloud processing, with state-of-the-art algorithms for filtering, feature extraction, and many others, in point clouds [58]. This software was used to process data acquired from the depth sensor, after converted into a point cloud.
- **Open3D**: it is an open-source library for 3D data processing, with state-of-the-art algorithms. It allows parallelization for processing time optimization [59]. This software was used to create a point cloud model from RGB-D data.

- **Intel RealSense SDK 2.0**: this software was used to interface with the D435i sensor, namely the Intel RealSense Viewer that allowed visualization of the depth, RGB, and motion streams, as well as configuring camera settings, add filters, and record and playback streams. [60].

- **Meshlab**: Meshlab is an open-source software specialized in 3D meshes, which offers a wide variety of tools such as editing, cleaning, healing, inspecting, rendering, and converting meshes. This software was used to register the different point clouds and, subsequently, convert them into a mesh [61].

4.2 Data Acquisition

Initially, the idea was to mount the RGB-D sensor on a second UAV since the DJI P4 does not allow to mount other components due to payload, and have both scannings (RGB and RGB-D) performed on this type of vehicle, but due to unexpected construction work surrounding the ship that blocked some of the passage of the UAV, only the RGB scanning was performed using it, while the RGB-D scanning was done holding the sensor in the hand and going through the zones where the UAV could not access nor scan.

For scanning the ship with DJI P4, its motion was controlled via a remote controller and set to automatically take pictures with a counter of 2 seconds, storing the data in a micro SD card placed inside. As for scanning with the RGB-D sensor, it was connected to a computer running the Intel Realsense Viewer and was held by hand while walking around the ship and recording the streams into a bag file.

When it comes to the flight pattern of the UAV, the chosen pattern for scanning the ship was to perform a circular raster scan, with the sensor pointing at different angles, ending with a straight path over the ship to capture its top surface (Fig. 4.2). This way, there will be overlaps between readings and multiple scans of the same parts of the ship are taken from different views, leading to a higher-quality 3D model. Obviously, parts of this flight pattern will be disrupted by the construction materials.

4.3 Structure from Motion

After acquiring the RGB images, a 3D reconstruction is performed. To do so, as mentioned previously, it was used Meshroom framework instead of doing manually, as SfM workflows usually are the same independently of the application, meaning reconstructing a ship or a car uses the same steps, so time can be saved for tuning different parameters and evaluating different algorithms for each step for the specific application, with the objective to obtain the most accurate model. The pipeline of Meshroom is composed of camera initialization, feature extraction, image matching, feature matching, and Structure from Motion (Fig. 4.3).

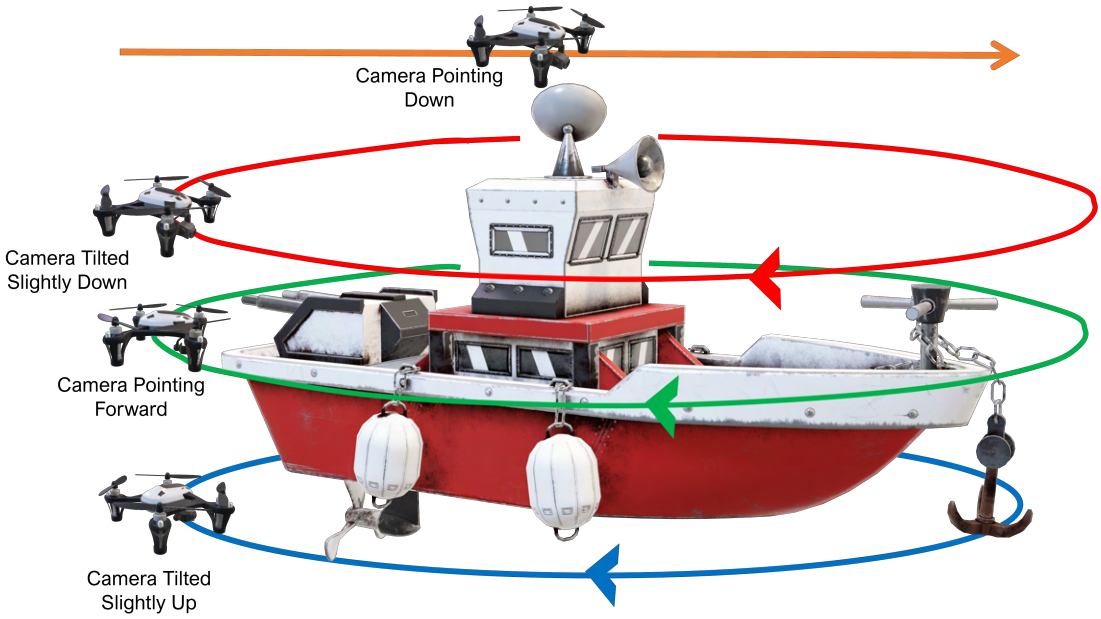


Figure 4.2: Proposed ship scanning pattern.

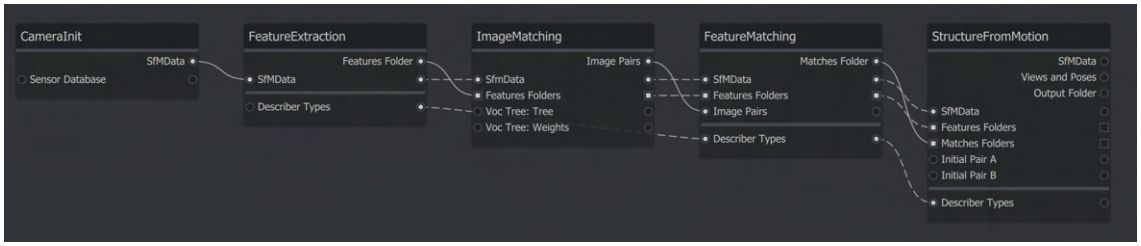


Figure 4.3: Meshroom pipeline for Structure from Motion.

4.3.1 Camera Init

Although this is not an inherent step in SfM, it is a necessary step in Meshroom workflow. Here, the images acquired are loaded, including their metadata. The metadata is very important, as it contains information relative to the images that aid in the reconstruction in terms of accuracy and speed, thus the more parameters known, the better results are achieved. This information includes GPS location, camera intrinsics, camera type (pinhole, fisheye, etc), sensor width and height, distortion parameters, and other parameters not so important for the reconstruction. GPS location, if available, is of extreme importance as it reduces drastically the camera poses errors.

4.3.2 Feature Extraction

There are a few feature detection and description approaches available in Meshroom, as well as variants of each, but the main ones are:

- **Scale-Invariant Feature Transform (SIFT)**: it's one of the most popular and commonly used algorithm for feature extraction in images. This algorithm is invariant to image transformations, including translation, rotation, and scaling, as well as somewhat invariant to illumination changes. Nonetheless, this method uses the Gaussian scale space, so when there's Gaussian noise present in the image, it is not an effective method. Furthermore, it has a high computational cost [62].

- **Domain-Size Pooling SIFT (DSP-SIFT)**: this algorithm is an improvement of the original SIFT. It outperforms SIFT when there are larger scale changes between images, as well as it's more robust to Gaussian noise and illumination changes [63].

- **Accelerated KAZE (AKAZE)**: it's an improvement of KAZE. This feature extraction method can detect features in nonlinear scale spaces, meaning it is more invariant to Gaussian blur and, furthermore, invariant to image transformations. Also, it has a low computational cost and low storage requirements for feature descriptors [64].

- **CCTAG**: this type of feature extraction requires the placement of CCTAG markers, made up of circles, on the ship, to be localized in the images enabling to detect the camera pose.

For the scope of this project, as it will be seen in Chapter 5, an evaluation of the performance of each algorithm, except for CCTAG as no marker will be placed, is conducted by performing a quantitative research of the results, in terms of speed and other relevant metrics, as well as a qualitative evaluation through visual inspection of the resulting model. Based on this evaluation, it is chosen what algorithm is considered the best for extracting features from the ship.

4.3.3 Image Matching

Image matching is not a mandatory step in the SfM workflow and this can be skipped straight to feature matching, as described in Chapter 3. Nevertheless, Meshroom includes this process and this has many benefits, in terms of accuracy and reduction of computational cost. As the image dataset collected during the survey might be unordered, matching the features extracted in the previous step can have a huge computational cost. Previously matching images that have part of the same scene reduces significantly the time to perform feature matching. To do so, Meshroom has available some methods to perform this matching:

- **Vocabulary Tree**: the feature descriptors extracted previously are organized in a tree using hierarchical clustering. Each descriptor is compared to the ones in the indexed tree nodes and gets placed on the one closest to it. An image descriptor is generated using the node indexes.

- **Sequential**: this method speeds up the process of image matching if the images were taken in sequential order, thus consecutive images are matches with one another instead of searching for matches in all images. Meshroom also has available a combination of both sequential and vocabulary tree.

- **Exhaustive**: this is the method with the highest computation cost, as every image is matched with all the images available. Still, it is the method that achieves the best results in terms of accuracy.

- **Frustum**: if the camera poses are known, this method can be used. It matches the images by computing the intersection between the different cameras. It also has an available method in which Meshroom decides if it's better to use Frustum or Vocabulary Tree.

As in feature extraction, a quantitative and visual study is performed in Chapter 5 of all the mentioned methods for image matching, for a better grasp of the most adequate algorithm for the application.

4.3.4 Feature Matching

For matching all the features extracted from the matched image pairs, Meshroom uses two steps. In the first step, for each feature in one image is created a list of possible features from another image that could be matched with. Then, based on the feature's descriptors, the two features in the list with the closest feature descriptors are used and the remaining are eliminated. Since a descriptor should only have one match in the other image, a threshold between the two is applied in order to discard one of them. In the second step, a filtering process of the bad correspondences is done. To do so, RANSAC is used for geometric filtering based on the feature positions.

Two parameters in this step are available to use: cross-matching and guided matching. In cross-matching, the first step explained before is applied to the image pair, from image A to image B and vice-versa. This achieves better accuracy as it only keeps the match if both have the same match. The resulting correspondences then go through the RANSAC process. Since it has to check for matches twice as many times as if this parameter wasn't used, the computational time will also be greater. Relative to guided matching, by enabling this parameter, a re-computation of the descriptors matching is done, with added constraints, where the geometric information and camera poses determined on the first computation of the matching process are used, improving the number of matches. This will provide better matching results, but since the computation has to be performed a second time, it will take longer. Both of the parameters will be evaluated later on and the results are what dictates if it is worth using them or not.

4.3.5 Structure from Motion

The last step in the pipeline is the Structure from Motion *per se*. The way Meshroom performs this step is exactly as explained in Sec. 3.2. First, an initialization is performed, where an initial image pair is picked. Afterward, the fundamental matrix is determined for the imaged pair to estimate the camera pose of both images, so that the triangulation process can occur, where the 2D corresponded features can be turned into 3D points. Afterward, new images are added by solving the PnP problem. To refine the 3D points, BA is applied to minimize reprojection errors. This entire algorithm, except for the finding of an initial image pair, is iterated until no more views can be localized.

In this step, Meshroom does not provide any relevant parameters to tinker with. For this reason, only its performance will be tested out.

4.4 RGB-D Image-Based 3D Reconstruction

To reconstruct the 3D model of the ship based on RGB-D images, the reconstruction pipeline was based on [65] and [66], as well as examples available in the Open3D documentation.

4.4.1 RGB-D Image to Point Cloud of Local Geometric Structures

The first step in the proposed pipeline mentioned in Chapter 3 is the registration of RGB-D images to create point clouds for various local geometric structures of the ship. Since the ship's recording is a video encoded in a bag file, each video frame pair (RGB and depth) is extracted from it to enable the registration process. Each local structure point cloud is derived from 100 frames, so the number of point clouds that are generated is dependent on the number of frames recorded during surveying. Furthermore, since the number of frames recorded might not be a multiple of 100, the last structure to be created is formed from several frames ranging from 1 to 100. Seeing that the functions available in Open3D allow for parallelization, the number of points clouds that can be extracted simultaneously are the same as the number of processing units that the computer has.

To start the registration of a single structure (or fragments, as the authors of [65] call it), RGB-D odometry is used to merge all the RGB-D frames belonging to this specific structure, thus a pose graph is created (Fig. 4.4). A pose graph is a type of graph that encodes the 3D positions and orientations. For this pose graph, the pose information of each RGB-D image is stored. It is composed of nodes and edge constraints and the number of nodes is equal to the number of RGB-D images, which in this case is 100. Each node contains a 4x4 pose matrix relative to the first node (i.e. the first RGB-D frame), while the first node contains the pose relative to the identity matrix. Consecutive nodes are connected by a type of edge that adds a spatial constraint called the odometry edge. These edges represent the odometry measurement between frame x_i and x_{i+1}, therefore it is encoded in them the transformation matrix from one node to the other and a 6x6 information matrix, which indicates its uncertainty. For non-consecutive nodes (e.g. between frame x_i

and x_{i+2}, x_{i+3}, ...), edge constraints are also determined, which are known as loop closures. Each edge of this type contains encoded the same data as the odometry edge, except the uncertainty in the information matrix will be higher.

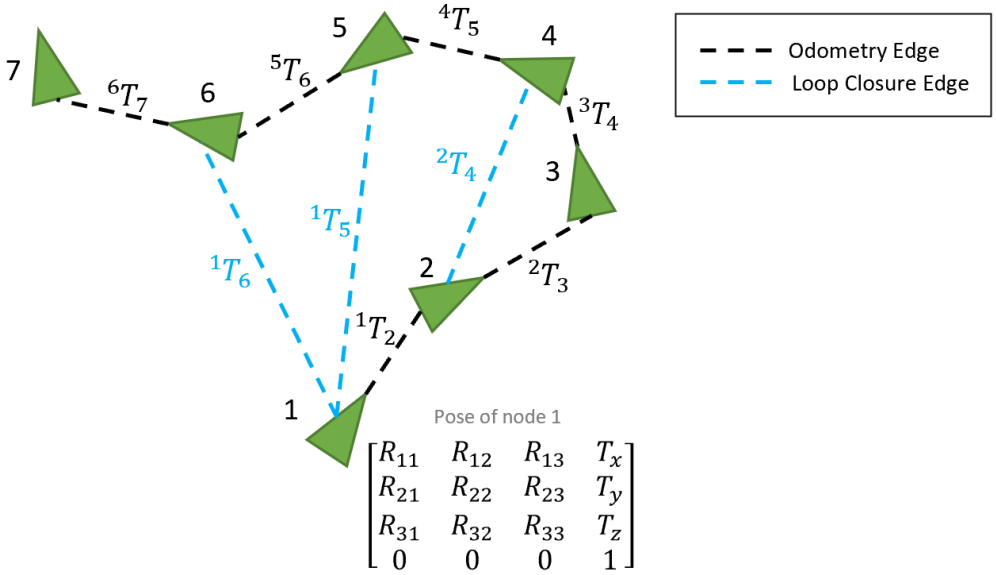


Figure 4.4: Sample demonstration of a pose graph.

To register a pair of RGB-D images, in which one is the source and the other the target image, an algorithm available in Open3D is used, which is based in [66]. Since this algorithm requires an initial transformation, when the pair is composed of consecutive image frames, the identity matrix is used, thus assuming there is no movement between frames. When the pair is composed of non-consecutive frames, the initial transform is calculated by, first, using an OpenCV function that determines ORB features in both images, followed by applying a 5-point RANSAC to estimate a rough transform between them. Each image frame is registered with its next frame and for every 5th image, additional registrations are performed with frames multiple of 5 (e.g. registration between frames 5 and 10, 5 and 15, 5 and 20, 5 and 25, ..., 10 and 15, 10 and 20, ...), although only if a valid initial transform was obtained when applying RANSAC. After each registration, an information matrix and a transformation matrix are obtained which is then encoded into the according edges of the pose graph. As for the nodes, each transformation matrix obtained for consecutive frames is accumulated with the transformation matrix of the previous node.

After all the relevant parameters are encoded in the pose graph, an optimization process is applied to it to adjust each pose according to the nodes and edge constraints. Besides, some pairwise alignments in non-consecutive frames might be false positives, so loop closure edges must be corrected accordingly. By solving the least square curve fitting problem, the global configuration of the scene can be estimated and can remove from the graph edge constraints that are incorrect [65]. To solve the least square problem, two options in Open3D were available: Levenberg-Marquardt or Gauss-Newton algorithm.

Since the former has better chances of local convergence even when the parameters are far from the optimal value, this was the preferred algorithm.

Once the final pose graph is obtained, a Truncated Signed Distance Function (TSDF) volumetric integration algorithm is applied. What this algorithm does is integrate all the RGB-D frames with known poses into a voxel block grid, using a weighted sum of Signed Distance Function (SDF). This grid divides the 3D space into blocks and each block is further divided into equally sized voxels. SDF gives us the distance between the center of a voxel and the nearest surface boundary present in the RGB-D frame, along the direction of the measurement. Furthermore, each voxel has a corresponding weight that indicates the level of uncertainty of its SDF. As shown in Fig. 4.5, it is possible to calculate SDF of a voxel for an i-th observation with (4.1), which was extracted from [67].

$$sdf_i(x) = depth_i(pic(x)) - cam(x) \tag{4.1}$$

where pic(x) is the projection of the voxel center x onto the depth image, depth(pic(x)) is the distance between the camera and the nearest surface boundary point p, and cam(x) is the distance between the camera and the voxel, measured along the optical axis. Since large distances have no relevance for surface reconstruction, it is possible to truncate the value obtained for SDF between values -t and +t, called truncated variant of sdf (tsdf), as demonstrated in (4.2).

$$tsdf_i(x) = max(-1, min(1, \frac{sdf_i(x)}{t})) \tag{4.2}$$

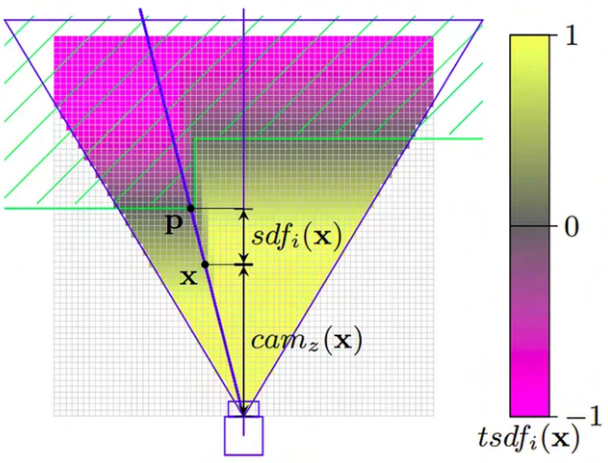


Figure 4.5: SDF and TSDF example, where the object is given by the green illustration, camera and FOV in blue, and voxels by the spaces present in the grid. [67]

Multiple observations of an object can be combined into a single TSDF by integrating all $tsdf_i$ of different views corresponding to that observation, using a weighted summation. The weight of the uncertainty is set to 1 for voxels where $tsdf_i$ was calculated and 0 for voxels outside the camera's FOV. The average of the weights composes the final weight for a TSDF. Once all the RGB-D images that compose a local geometric structure are integrated

and all TSDF and weights are calculated, a surface point cloud of it can effectively be extracted.

4.4.2 Registration into a Global Scene

Once all RGB-D frames are registered and integrated and all the local structures point clouds are derived from that process, these must be aligned to create a single global structure. The typical process for registering points clouds, presented in Fig. 3.3 delineated by the orange rectangle, is applied.

To register one point cloud with another, a description of the features of each single point must be determined to enable matching one another. Although, describing and matching all the points of the point clouds can be quite time-consuming and redundant. Since we want to keep the computational cost as low as possible, a voxel grid filter was applied to all point clouds, with a leaf size of 5 cm, to reduce the number of points by approximating all points that fall inside a voxel into its center, also creating a uniform spatial distribution of the point cloud.

For registering all the point clouds, a similar approach to RGB-D frame registration was taken: creating a pose graph composed of constraint edges and as many nodes as the number of point clouds. As before, there are two scenarios to roughly align two point clouds:

- **Non-consecutive point clouds**: in this case, FPFH feature descriptors of each point of the downsampled point cloud are computed and the source and target cloud are registered using RANSAC, where a random subset of points from the source cloud are matched with the corresponding points in the target cloud. If the aligned points are at a distance greater than 7 cm, the correspondence is discarded. If not enough matches remain at the end to obtain an accurate transformation, a loop closure edge is not added.

- **Consecutive point clouds**: for this case, computing FPFH features to subsequently match is not necessary, as the transform that roughly aligns the clouds can be extracted from the last node of the source pose graph calculated previously when registering the RGB-D frames. On top of this transformation, the ICP algorithm is applied with loose parameters, as the objective is to simply obtain a better rough alignment and as such, setting loose parameters also decreases computational cost. The maximum correspondence distance was set to 5 cm, the maximum number of iterations to 50 and a converge criteria of 1e-6 for fitness score and 1e-6 for RMSE. The resulting transformation is used to create an odometry edge and accumulated with the previous node's transformation matrix in order to encode it in the current point cloud's node.

Just as before, a pose graph optimization is performed on the final pose graph. The process is identical, solving the least square curve fitting problem with the Levenberg-Marquardt algorithm.

To get the most accurate alignment of the clouds, a fine-tuning of all of the transformations is performed. As such, a color-based ICP algorithm is applied, where the robust transformations calculated previously are used as the initial transformations required for this algorithm to work. Furthermore, the color information of the points is taken advantage of, which further improves the accuracy of the alignment.

Following the same steps as before, a pose graph was created for the refined registration. For this case, the way each refined registration was computed had no distinction between consecutive and non-consecutive local structures, meaning that for both, the same algorithm was applied. To fine-tune the registration, for each pose calculated previously between structures, an ICP algorithm was applied three times, once with a maximum correspondence distance of 5 cm for a maximum of 50 iterations, followed by a maximum distance of 2.5 cm for a maximum of 30 iteration, and lastly for with a maximum correspondence distance of 1.25 cm for a maximum of 14 iteration, where each used the transform obtained in the previous ICP result as the initial alignment, except for the first iteration. By reducing the maximum correspondence distance in each passage of the algorithm, it creates a tightly aligned point cloud, increasing the accuracy of the final model. As for the pose graph optimization, the same process was applied as previously.

When all point clouds are tightly aligned, TSDF volumetric integration algorithm is applied once again, now registering all the RGB-D frames of the entire scene, using the fine-tuned registration transformations. From this, a mesh file of type .ply is extracted, which is then converted into a point cloud file of the same file type, by extracting the mesh's vertices and vertices color, which later will be used to merge with the point cloud extracted from SfM.

4.5 Corrosion Classification

In this section, an explanation of the methods for representing the corroded parts of a ship in the 3D point cloud using the corrosion classification pipeline proposed in Sec. 3.4 is presented, as well as the process of obtaining the classification model.

4.5.1 Dataset Acquisition

Since the corrosion classifier is binary, two datasets were created, one where there were multiple different scenes with corroded objects and another with different scenes where no corrosion is present. To achieve a model with a high prediction accuracy rate, since this project focuses on the scanning of a ship, which is usually placed in an outdoor environment, the corrosion dataset requires to have at least some images of ships, metal surfaces, outdoor environments, and bright light scenes, all with corrosion present, while

the dataset with no corrosion must have some images with copper and red objects, as they can be confused with corrosion, as well as some other outdoor environment images.

To build the dataset, public datasets in Kaggle [68], as well as some copyright-free images from Pexels [69] were used, obtained by web scraping. In Figs. 4.6 and 4.7, a few images of the corrosion and no corrosion datasets, respectively, are shown. The corrosion dataset is composed of 1738 images and the dataset without corrosion is composed of 1400 images, resulting in a total of 3138 images.

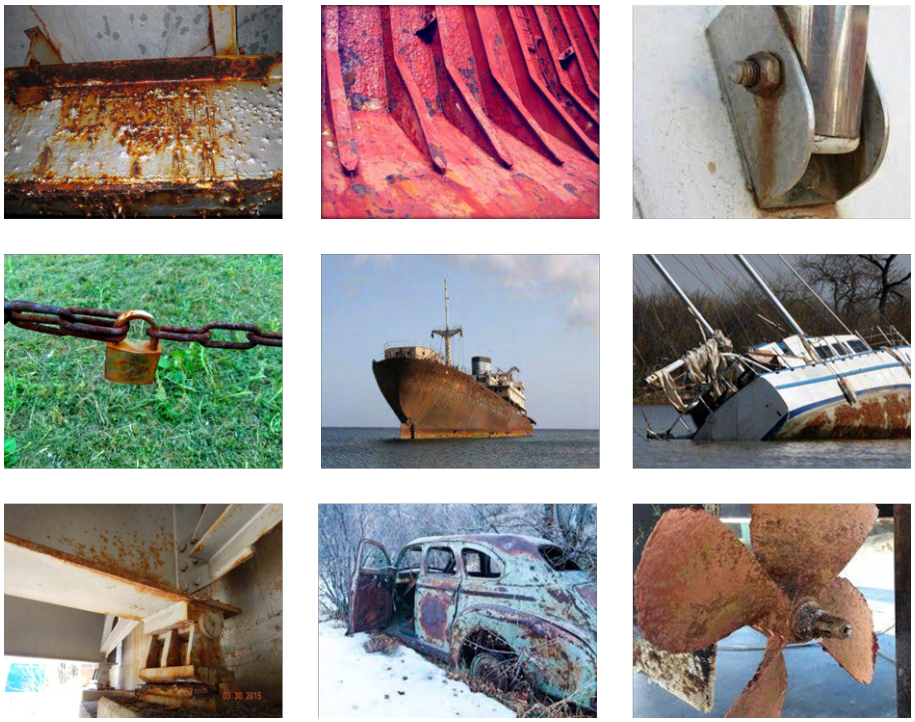


Figure 4.6: Examples of images from the corrosion dataset.

4.5.2 Dataset Processing

Preprocessing the dataset is important in order to achieve a more accurate model. Firstly, both datasets were loaded to Roboflow and split into different folders, where 70% of the images of each dataset are placed in the training folder, 20% in the validation folder, and 10% in the test folder.

Afterward, some image augmentations were performed on the training dataset (Fig. 4.8), such as horizontal and vertical flips, and 90° rotation clockwise, counter-clockwise, and upside down. Since corrosion doesn't have a defined shape or orientation, these augmentations make it suitable for many different types of orientations. Furthermore, a saturation between -25% and +25% was applied to make the model more robust to differences in color vibrancy, especially when there are completely different colors present in the image. Brightness changes between -15% and + 15% as well as exposure changes between -5% and +5% were applied. Both these augmentations help the model be more

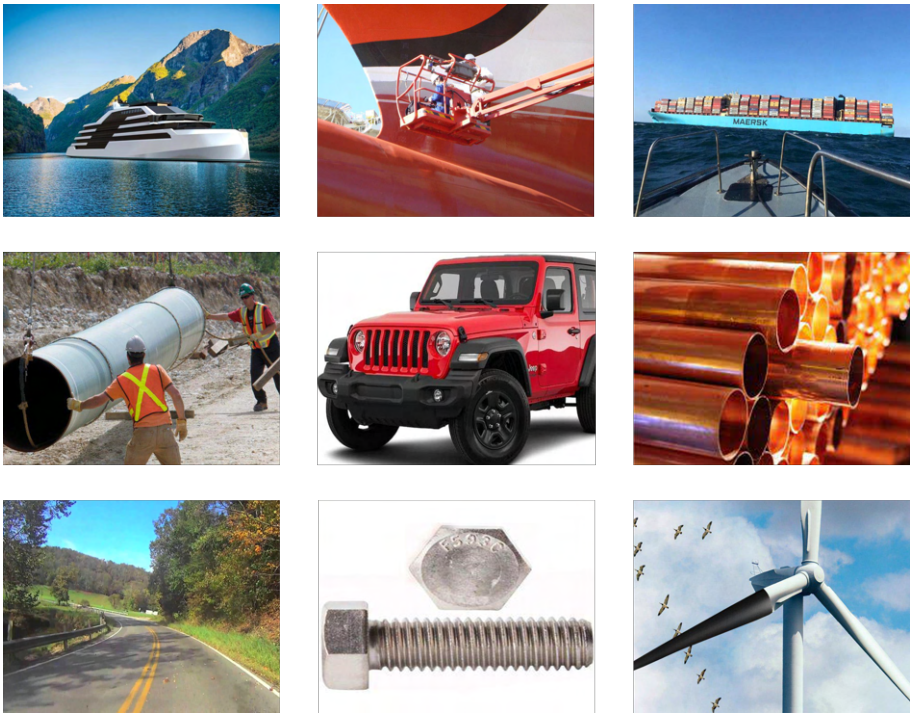


Figure 4.7: Examples of images from the dataset without corrosion.

robust to lighting changes, which is especially important in outdoor environments where the lighting isn't regulated. Lastly, a small amount of blur of up to 0.75 px was applied, which makes the model more resilient to different camera focus or Gaussian noise that could be present, primarily due to the resizing of the image regions to the training size. Resulting of all these transformations, the training dataset triplicated in size to approximately 6.6k images, combined from corrosion and no corrosion datasets.

Figure 4.8: Examples of augmented images in the training dataset.

After, the datasets were uploaded to Google Colab to exploit its GPUs for faster training and to make use of TensorFlow libraries to further process the images. To increase the speed of the training process and achieve peak performance, there should be an optimization of the input pipeline of each training step. A training step is composed of the following operations: opening data, reading data, and model training. If there was no optimization, while data for the training step is being opened and read, the model is idle and not training, and vice versa, when the model is training, the input pipeline of

the next step is idle. One way of optimizing is by prefetching the data, so that while the model is training the current training step, it is fetching data for the next step, reducing time expended in a step. Another way is by caching the images so that the images are kept in the memory after they are loaded, avoiding the necessity of opening and reading the images in every epoch. To test out the performance of these optimizations, a benchmark on the test dataset was performed for a dummy training loop of two epochs, where each training step had a duration of 1 ms. The results obtained are presented in Table 5.1. As shown, both methods reduce computation time, with cache having a bigger impact. Unfortunately, only the prefetching optimization could be applied, as the cache would cause the RAM of Google Colab to run out when applied in the training dataset.

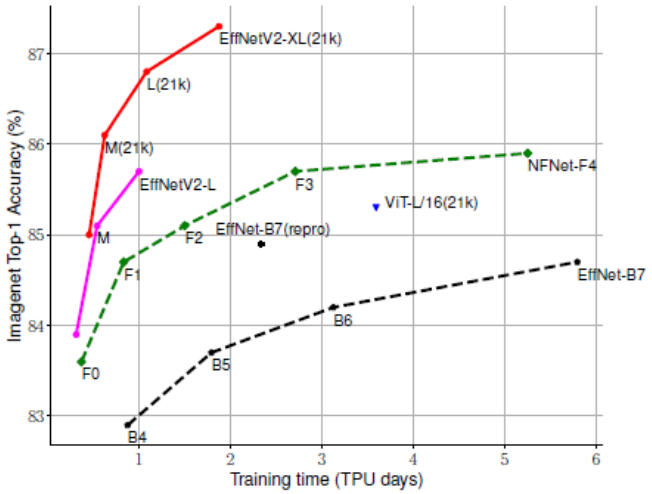
Table 4.1: Benchmark results from optimization processes.

Optimization Method	Execution Time
None	1.192727
Prefetching	1.185312
Caching	0.642868

4.5.3 Model Architecture

As mentioned previously, to create the classification model, transfer learning is exploited. Due to the smaller size model, faster training time, and higher accuracy (Fig. 4.9), EfficientNetV2 was chosen as the base model, which is a type of CNN, that was created by Google and trained on the ImageNet21k dataset. Since this model requires pixel values in the range [0, 1] and the input image size to be 260 x 260, a resizing and rescaling of the entire dataset was applied.

The final architecture is composed of a sequential model, using tf.keras.Sequential in TensorFlow, which contains an input layer, a base model, a dropout layer, and a dense layer. The input layer defines the input shape of the images and is set to 260 x 260 with a three-color channel. The base model is the EfficientNetV2, where its layers were frozen in order for the weights of this model to remain the same. The dropout layer helps prevent overfitting by randomly ignoring some inputs and this was set with a rate of 0.2. Lastly, the dense layer is where it is defined the number of classes to classify, which in this case is one, and added an L2 regularizer with a parameter of 0.0001. This regularizer updates the cost function of the model by randomly forcing the weights of the trainable layers to be small, close to but not exactly zero. Adding a regularizer, forces the model to generalize and hence, perform better, as only the most relevant features of corrosion are stored by the model while the rest are discarded. A summary of the model is shown in Fig. 4.10. As can be seen, by making the base model layer non-trainable, just the dense layer can be trained.



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

Figure 4.9: Training time and accuracy of EfficientNetV2 compared to other models [70].

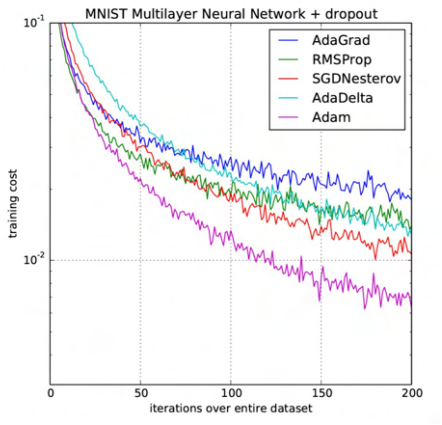
```
 Layer (type)                 Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)     (None, 1408)              8769374

 dropout (Dropout)            (None, 1408)              0

 dense (Dense)                (None, 1)                 1409

=================================================================
Total params: 8,770,783
Trainable params: 1,409
Non-trainable params: 8,769,374
_________________________________________________________________
```

Figure 4.10: Summary of the model created for corrosion classification.

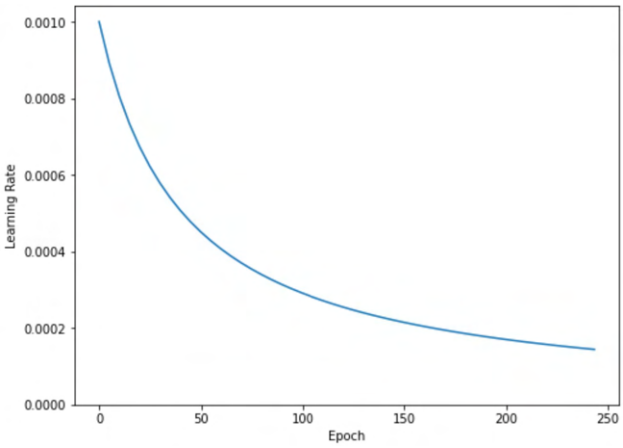
4.5.4 Training Process

Before starting the training process, a few functionalities had to be added first. A checkpoint callback was created, in which after every epoch during training, checks if it was the best model achieved so far, and if so, it saves it. To analyze if it should be saved, it monitors the binary cross-entropy loss, calculated by computing the cross-entropy of the true and predicted labels. The checkpoint allows to resume training at any given saved point, in the case of an unexpected interruption during the process.

In a further attempt to prevent overfitting, besides all the methods applied before, another callback function was created with another type of regularization known as early stopping. For an accurate model, one should train until it reaches a certain point where



(a) Comparison of Adam optimizer with other, in terms of computational efficiency [71].



(b) Learning rate function used to train the model.

Figure 4.11: Graphs relative to the optimizer selection process.

the chosen metric for monitoring has plateaued or even started degrading. When the early stopping callback is invoked, it stops the training process. For the scope of this project, the monitored metric is the binary cross-entropy of the validation dataset, with a patience of 2, meaning that after 2 epochs where performance has not improved, the training stops.

An important task required to do prior to training is setting a loss function, an optimizer and the desired metrics to be monitored, through the use of the compile function available in TensorFlow. The loss function is the same as the one on the checkpoint callback mentioned before, so binary cross-entropy loss is used. As for the optimizer, its objective is to adjust the weights of the model based on the loss function and data received. From the available ones in TensorFlow, the Adam optimizer was chosen, as it is the one with the less computational cost and low memory requirement, while at the same time achieving great results (Fig. 4.11a). Additionally, a scheduled learning rate was added for the optimizer which controls it by defining how much the model's weights can be tuned in response to the loss function. To do so, an inverse decay function was used as the scheduler, with an initial learning rate of 0.001, which updates the learning rate of the optimizer at every epoch. This function gradually reduces the learning rate during training by $1/2$ at 41 epochs, $1/3$ at 82 epochs, and so on, as depicted in Fig. 4.11b. By starting with a bigger learning rate, the training process is faster and a sufficiently accurate set of weights is obtained and by lowering the learning rate, the training speed decreases, but the weights are fine-tuned to get a model with a better accuracy. Lastly, for the compile method, the metrics set to be monitored during training are once more the binary cross-entropy loss, as well as the accuracy, as it measures how often predictions match the corresponding label of the images.

As for training the model *per se*, it was set to train for 10000 epochs with the callbacks previously explained set. The results of the training process are presented in Chapter 5. As a side note, all the values mentioned previously are the ones obtained after tinkering

and experimenting in order to achieve a higher accuracy of the model.

4.5.5 Fine-Tuning

For fine-tuning the model, the exact same process as before, for training, was applied, with the exception that the base model was made trainable, meaning its layers were unfrozen and its weights could be updated, as well as for the optimizer, a fixed value of 1e-5 was used instead of a scheduler, meaning the weights are updated in a smaller increments/decrements, making it more precise.

4.5.6 Image Splitting and Classification

The images taken from the DJI P4's camera have a resolution of 1600 x 1300. To divide the image into equal-sized regions with integer widths and heights, each image was split into 8 columns and 5 rows, resulting in a total of 40 regions of 200 x 260 pixels. The images could be split into more or even fewer regions, but since the model is trained for an image size of 260 x 260, a resizing of the regions is needed and the bigger the need for resizing, the more detail can be lost, or even introduce huge amounts of Gaussian noise, making the model not as accurate, thus by having regions with an approximate size as the one required, a lesser degree of resizing is needed.

For each region, an ID is assigned from 0 to 39, starting from the top left corner and incrementing until the bottom right corner of the image, as depicted in Fig. 4.12. Afterward, its pixels are normalized to an interval of [0, 1], the same way the model was trained for, and a prediction on each region is made. The results are stored in a dictionary where the key is the image path and the values are the region's IDs where corrosion was detected.

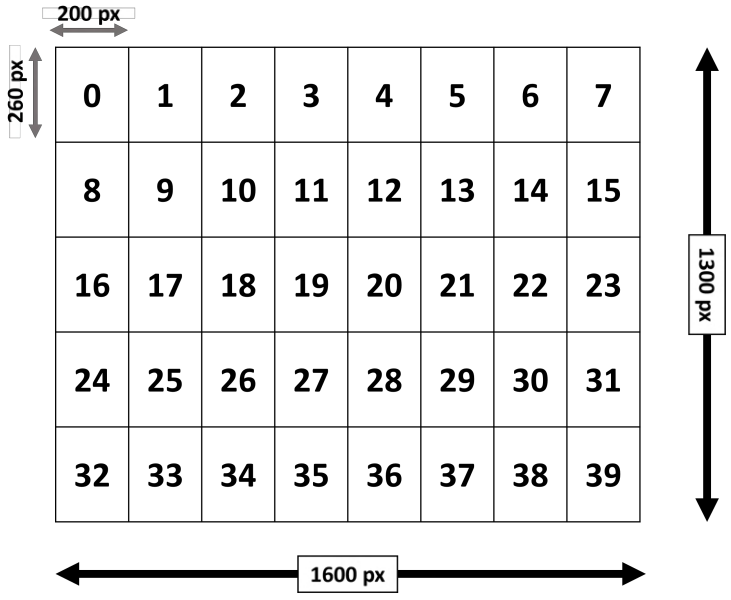


Figure 4.12: Overview of the different regions of an image with its respective ID.

4.5.7 Corrosion Representation

During the Structure from Motion step in the Meshroom pipeline, besides the point cloud being generated, a convenient JSON file is also generated, detailing all the points in the cloud and which pixels from which images generated each point. This facilitates the job of representing the corroded parts in the point cloud. Its structure is summarized in Fig. 4.13.

The most relevant field of the JSON file is "Landmark". Here, all points are listed, as well as their information. For each point, corresponding to one landmark, it is given an ID, detailed which describer type generated this point (SIFT, AKAZE, ...), the color of the point, its location, and a list of observations. These observations correspond to pixels and features that generated this point. Each observation has two IDs, one ID is the "observationID" which is equivalent to the ID of the image to which the pixel corresponds, and the other ID is the "featureID" which corresponds to the ID of the feature of that specific image that generated the point. Each observation also has the pixel location in the image (corresponding to the JSON field "x"), as well as a scale, which is always 0 in this case. In the "Views" field, all images information are exposed, namely its corresponding ID, file path, and metadata.

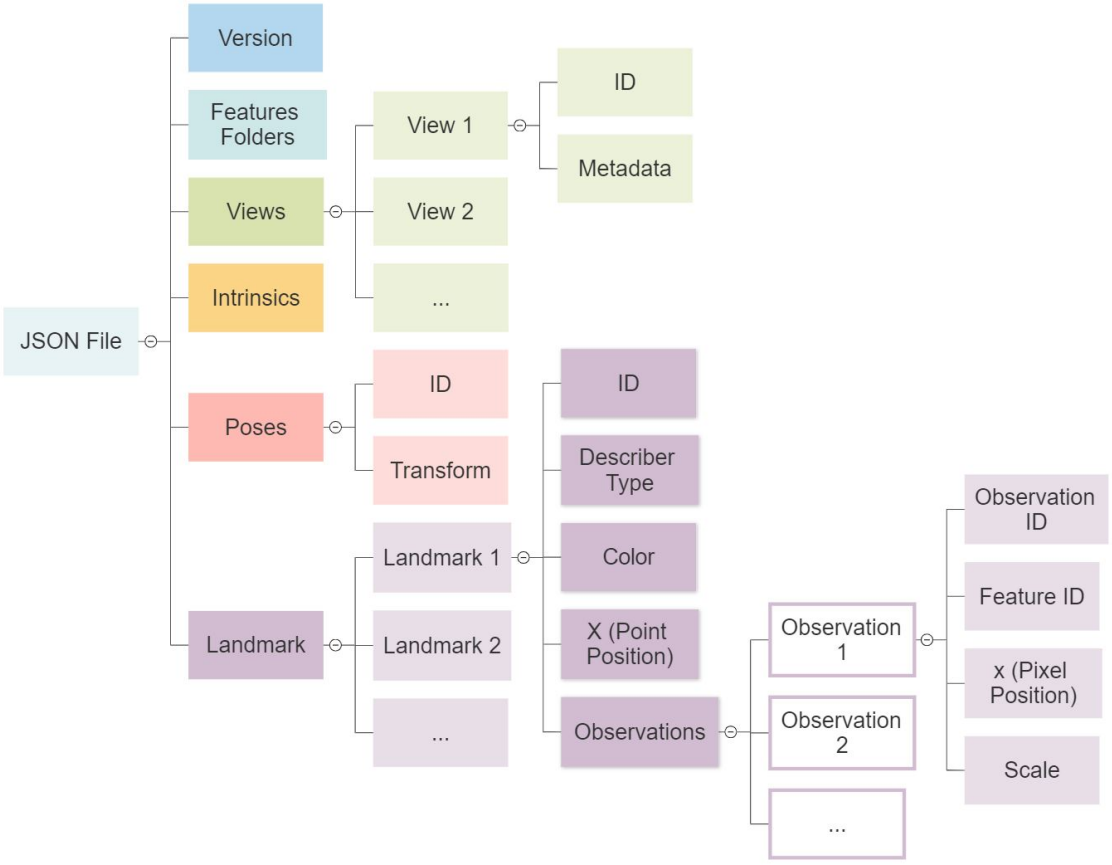


Figure 4.13: Generated SfM JSON file organization.

By going through the "Landmark" field, it can be observed if the point should be represented as corrosion or not. The algorithm for such is simplified in Fig. 4.14. First, an observation is retrieved and its associated "observationID" is extracted in order to match with the ID of an image in the "Views" field and retrieve the file path. Afterward, the pixel location is retrieved, and, based on this, the region ID corresponding to its location is calculated by determining the region ID (4.3).

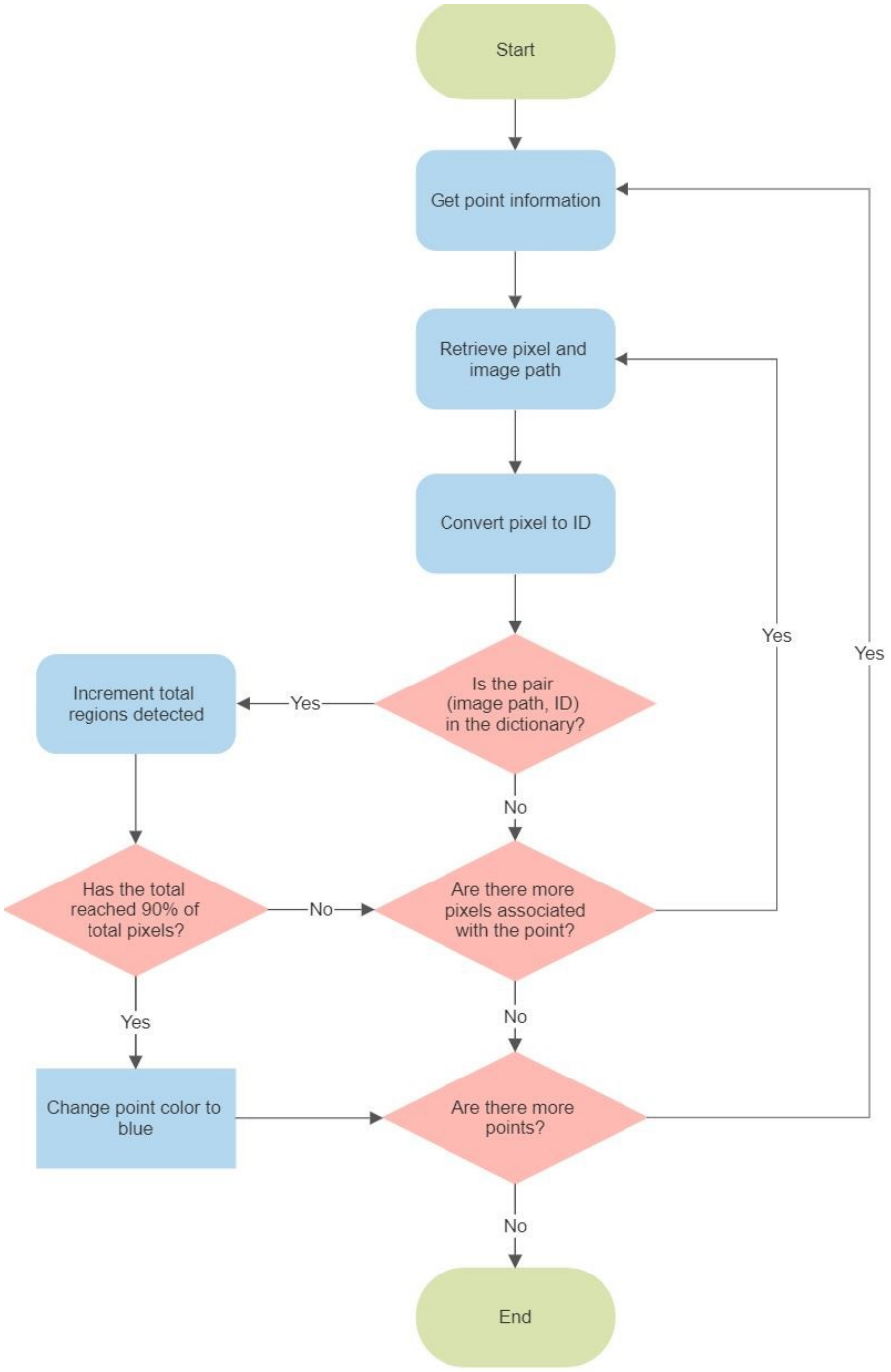


Figure 4.14: Algorithm for identifying points with corrosion and changing its color accordingly.

$$regionID = int(\frac{pixel_x}{\frac{width}{columns}} + \frac{pixel_y}{\frac{height}{rows}} * columns) \tag{4.3}$$

in which $pixel_x$ and $pixel_y$ correspond to the pixel coordinates, width and height correspond to the predefined region width and height, respectively, and rows and columns correspond to the number of rows and columns of regions present in a full image.

Following this, the pair (image path, region ID) is searched in the dictionary calculated previously, containing the image path keys and the respective detected regions with corrosion as values. In case it is present, it means that said pixel is in a region where it was classified as corrosion, thus a counter is incremented. If the counter is superior to 90% of all the pixels that originated that point or, in other words, if 90% of the pixels are in a region where corrosion was detected, the point is identified as corrosion and, therefore, its color is changed to blue, with an RGB value of (0, 0, 255), and the algorithm proceeds to check another point if any are left. Case 90% is not achieved yet, it continues to process each pixel and if that percentage is not achieved by the end, the point is not identified as a corrosion point and its color remains the same. This high percentage allows it to be more robust to possible wrong classifications, as the classification model is not 100% accurate, as well as more precise in detecting zones in the ship with corrosion.

4.6 Merging Point Clouds

In this final section of the current chapter, the process of merging the SfM model with the corrosion highlighted, and the RGB-D model is performed, as well as an explanation of the adaptations it had to be done to the proposed method in Chapter 3 due to the construction work.

4.6.1 Principal Component Analysis

The first step in PCA is to project the point cloud in the eigenvector space. To do so, the following steps must be taken:

1. **Normalization:** all the points' locations are normalized in the interval [0, 1], so that the variance between each point is smaller, causing fewer sensitivity issues when performing the analysis.

2. **Calculate the mean for each dimension:** since the data is 3D, there are 3 dimensions, thus the mean for the x, y, and z-axis of all the points is calculated.

3. **Covariance matrix computation:** a 3x3 covariance matrix is computed to detect any relationship each dimension has with each other. The matrix will assume the

following form:

$$\begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

The formula to compute each element of the covariance matrix is given by (4.4):

$$cov(A,B) = \frac{\sum(a_i - \overline{a})(b_i - \overline{b})}{N-1} \tag{4.4}$$

where cov(A,B) is the covariance between A and B variables corresponding to a dimension, a_i and b_i are a data member of A and B, $\overline{a}$ and $\overline{b}$ are the mean of each corresponding dimension, and N the size of the point cloud.

4. **Compute eigenvalues and eigenvectors of the covariance matrix:** these eigenvectors correspond to the principal components mentioned in the previous section. These principal components will indicate the directions of the data with the maximum amount of variance. To get the eigenvectors, the 3 eigenvalues must be calculated first, in a manner shown by (4.5):

$$\left|M - \lambda I = 0\right| \Rightarrow a\lambda^3 + b\lambda^2 + c\lambda + d = 0 \tag{4.5}$$

where M is the covariance matrix, λ the eigenvalues, I the identity matrix, and a, b, c, and d the resulting variables from expanding the determinant. After discovering the 3 eigenvalues, the eigenvectors can be determined by solving (4.6) for each eigenvalue:

$$M \cdot \overrightarrow{u} = \lambda \cdot \overrightarrow{u} \tag{4.6}$$

where $\overrightarrow{u}$ is the eigenvector represented in the form of a 3x1 matrix.

5. **Create a feature vector:** to create a feature vector, firstly the eigenvectors are sorted in descending order based on their corresponding eigenvalues. The higher the eigenvalue, the more information the eigenvector carries, thus the more relevant it is. This part is where the dimensionality reduction comes into play where lower-ranked eigenvectors can be disposed of as they carry less information about the dataset. Since in this case, all the information pertaining to the location in the x, y, and z-axis are required, no eigenvector is dropped. Thus, the feature vector W is a matrix that has as columns the eigenvectors in descending order, meaning that the first column is the eigenvector with the highest eigenvalue, the second column the eigenvector with the second highest eigenvalue, and so on.

6. **Project the point cloud along the principal component axes:** in this last step, the point cloud is transformed from the initial axes to the axes formed by the principal components (4.7):

$$Y = W^{\intercal} \cdot P \tag{4.7}$$

where Y is the projected point cloud, $W^\intercal$ is the transpose of W, and P is the original point cloud.

Luckily, PCL has a function that automatically does this whole process. After applying this function to all models, the centroid of the point cloud as well as the new axes are determined which is used for transforming the data as it will be explained next. Note that all the models will be projected onto the same axes and, consequently, their centroid corresponds to the same point.

4.6.2 Transformation - Scaling, Rotation, and Translation

After determining the principal components of all the models, a scale of the SfM model to the same scale as the RGB-D models is performed. Initially, the idea was to determine the minimum and maximum point along each principal component axis on the SfM Model and the RGB-D model of the complete ship and determine the scale change for each axis in the SfM model accordingly, as given in (4.8):

$$\begin{cases} scale_x = \frac{\max(RGB-D)_x - \min(RGB-D)_x}{\max(SfM)_x - \min(SfM)_x} \\ scale_y = \frac{\max(RGB-D)_y - \min(RGB-D)_y}{\max(SfM)_y - \min(SfM)_y} \\ scale_z = \frac{\max(RGB-D)_z - \min(RGB-D)_z}{\max(SfM)_z - \min(SfM)_z} \end{cases} \tag{4.8}$$

where max(RGB-D) and max(SfM) are the points furthest away from the centroid for the RGB-D and SfM model, respectively, in the direction of the most relevant principal component axis, which in this case is considered to be the x-axis, and min(RGB-D) and min(SfM) the points furthest away in the opposite direction of the axis.

Since it was not possible to scan the whole ship with the RGB-D sensor, the scaling process had to be done manually. To scale the SfM model manually, a distance measurement between two of the support structures of the ship was taken in Meshlab for the SfM model and right-side RGB-D model. By dividing the measurement obtained from the RGB-D model by the measurement obtained from the SfM model, we get the scale needed for the latter. Thus, by multiplying every point in the SfM point cloud by this scale, results in same scale models.

To rotate and translate the model, part of it had to be done manually due to the problems mentioned above. As the model for the left side of the ship had a different direction along the x-axis than the remaining two models, it had to be rotated by 180° in the y-axis. Afterward, since all the models have a structure in common which is the back of the ship where the turbine is and it is located in the positive direction of the x-axis, all RGB-D models were translated along that axis until the maximum point had the same x position as the maximum point in the SfM model. The steps taken subsequently were performed manually in Meshlab. A rotation in the x-axis of both RGB-D models as well as a translation along the y- and z-axis was performed until it aligned with the SfM model.

4.6.3 Converting to Mesh

The final step in the workflow is to convert the point clouds into a mesh. The following steps were taken in Meshlab:

1. **Flatten point clouds:** the 3 point clouds were transformed into a single point cloud.
2. **Color point:** each point color from the SfM point cloud is transferred into the closest RGB-D point, with a maximum distance of 2% of the world size.
3. **Normal computation:** the point cloud normals are calculated to be used in the next step.
4. **Screened Poisson Algorithm:** this algorithm is used to create the final mesh. It uses the oriented point cloud to reconstruct its surface.
5. **Smoothing:** Laplacian smoothing algorithm is applied to smooth any irregular surfaces that may appear.

5 Experimental Results

The purpose of this chapter is to explain the results obtained from testing the proposed model, in terms of computational speed, quantitative accuracy analysis, and visual qualitative analysis of the 3D models.

5.1 Study Area

The ship used throughout this work for testing purposes is located in Arsenal do Alfeite, in Almada (Portugal). This ship was selected mainly due to the fact that it is located in a quiet place, it's by one of the RICS research office, where Prof. José Barata is the coordinator, and also due to not being used by either the workers in Alfeite nor the people working in the office.

5.2 Structure from Motion Benchmarking

In this section, it is presented the values given to each parameter of the Meshroom pipeline, the algorithms tested during the experimental trials, as well as the thought process behind the choices made to obtain the final point cloud of the ship. The software ran on a computer with an Intel(R) Core(TM) i7-6700HQ CPU processor at 2.60 GHz and NVIDIA GeForce GTX 960M GPU, with 16 GB RAM, running Windows 10 Home 64-bit operating system with OS build 19043.2006.

As seen previously, the Meshroom workflow starts with camera initialization. Here, multiple images were uploaded, taken on two different days where lighting conditions were different (a sunny day and a cloudy day). On the sunny day, 141 images of the ship were taken and on the cloudy day, 265 images were taken, resulting in a total of 406 images used for reconstruction. Some examples of the images are shown in Fig. 5.1. As these images illustrate, the ship still had the presence of shadows on the cloudy day, although these shadows were less heavy than on a sunny day. Nevertheless, these conditions have some impact on the accuracy of the reconstruction. Additionally, construction work was, unfortunately, being performed nearby the ship during the entire dissertation time, so the flight patterns of the UAV proposed in Chapter 4 to obtain the best results could not be conducted entirely, as the right and back side of the ship had many construction materials and vehicles obstructing the passage of the UAV.

(a)

(b)

Figure 5.1: Sample of the images taken. (a) On a cloudy day. (b) On a sunny day.

For feature extraction, to best analyze which algorithm to use, a benchmark of the ones available was performed in terms of execution time, to evaluate computational time, as well as the number of features extracted, which translates to accuracy, as the more features extracted, the more accurate is the model. To perform this benchmark, every parameter, except for describer type (feature extraction algorithm), was left on default, namely, describer quality and density remaining on the "normal" setting, as the Meshroom documentation mentions that it is only worth changing these parameters to a higher setting if the dataset has less than 300 images, as the time taken will be significantly longer, which is not intended. Furthermore, the maximum number of features detected is limited to 20000 features per image. In this benchmark, seven tests were performed: three for using a single extraction algorithm, three for using a combination of two algorithms, and one test for all three algorithms used together. The results are shown in Table 5.1.

As it can be seen in Fig. 5.2, as the number of features extracted increases, so does the execution time. Comparing the number of features extracted by a single algorithm with the number of features resulting from using two or three algorithms, it can be observed that the number is the same when summed up. Although, the same does not apply to execution time as the sum of the times of single algorithm tests compared to the ones where more than one algorithm was used is less, with the only exception being SIFT + DSP-SIFT. A possible reason for this is the increased number of processes in the computer, decreasing its computational speed.

To further assess the individual algorithms to choose to provide the best compromise between accuracy and speed, with accuracy having a higher priority, a quantitative

Table 5.1: Execution time and number of features extracted obtained from the benchmark tests.

Algorithms	**Execution Time (s)**	**Features Extracted**
One Feature Extraction Algorithm		
SIFT	1136.800	8120000
AKAZE	565.381	2734734
DSP-SIFT	638.240	3347168
Two Feature Extraction Algorithms		
SIFT + AKAZE	1942.037	10854734
SIFT + DSP-SIFT	1656.111	11467168
AKAZE + DSP-SIFT	1200.648	6081902
Three Feature Extraction Algorithms		
SIFT + AKAZE + DSP-SIFT	2940.183	14201902

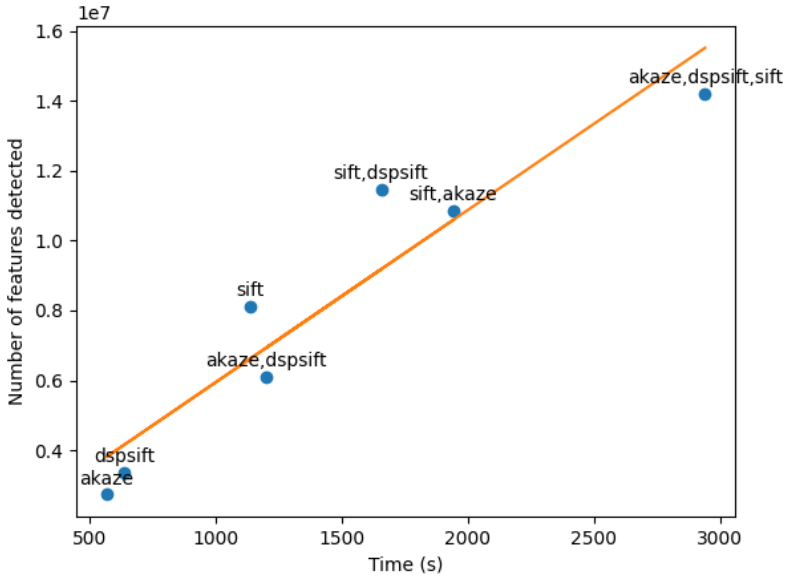


Figure 5.2: Graph of the number of features extracted in relation to execution time, in seconds. The orange line corresponds to the trendline.

study was performed based on the ratio of improvement of execution time and number of features extracted. Since SIFT was the algorithm that extracted the most number of features, all the results were compared with this algorithm, as demonstrated in Table 5.2.

By analyzing the table above, straight away the option of using simply AKAZE or DSP-SIFT can be eliminated as both reduce the number of features detected by more than 50% compared to SIFT. The same applies to using AKAZE + DSP-SIFT as besides taking longer to compute, it extracts fewer features. Since accuracy is the main factor, using just

Table 5.2: Table of the performance of the different algorithms and the merging of them, compared to SIFT results.

Algorithm	**Execution Time**		**Features Extracted**	
	Time (s)	%	Features	%
SIFT	1136.800	00.00	8120000	00.00
AKAZE	565.381	-50.27	2734734	-66.32
DSP-SIFT	638.240	-43.86	3347168	-58.78
SIFT + AKAZE	1702.181	+49.73	10854734	+33.68
SIFT + DSP-SIFT	1775.040	+56.14	11467168	+41.22
AKAZE + DSP-SIFT	1203.621	+5.88	6081902	-25.10
SIFT + AKAZE + DSP-SIFT	2340.421	+105.88	14201902	+74.90

SIFT is also not an option due to the remaining left options having a higher number of features extracted. For the remaining three options, a ratio between the percentage of features extracted in relation to SIFT and the percentage of execution time in relation to SIFT was performed, resulting in 0.68, 0.73, and 0.71 for SIFT + AKAZE, SIFT + DSP-SIFT and SIFT + AKAZE + DSP-SIFT, respectively, thus, since SIFT + DSP-SIFT presents a higher ratio, the algorithms SIFT and DSP-SIFT are the ones chosen to be used.

Following feature extraction comes image matching in the Meshroom pipeline. Since the images taken on both days were taken sequentially, sequential image matching was able to be tested. From all the methods explained in Chapter 4, Frustum was the only one that could not be tested out as the camera poses of the images were not available. Therefore, the remainder of the methods were analyzed, in which its results are presented in Table 5.3.

Table 5.3: Table of results of the different image matching methods.

Method	**Execution Time (s)**	**Image Pairs**
Vocabulary Tree	32.942559	12969
Sequential	00.624836	19025
Sequential and Vocabulary Tree	05.795092	12969
Exhaustive	02.016027	82215

As expected, the exhaustive method resulted in a larger number of image pairs compared to the remaining methods. It can also be observed a large reduction in execution time when combining the vocabulary tree with the sequential method when compared with just the vocabulary tree, while keeping the same number of image pairs. As this latter method is best suited for when there is a huge dataset, with thousands of images, if it were the case, it would probably be the preferred method in terms of execution time. Since the used dataset only has a few hundred images, exhaustive matching has surprisingly shown to be fairly quick and provides the most number of image pairs, thus this was the method chosen. Using just the sequential method would indeed lower computational cost

by 69.01% when comparing to the exhaustive matching, but also reduce the number of image pairs by 76.86% which will translate into a huge toll, in terms of accuracy, on the final model and, as mentioned previously, accuracy has a bigger influence in this project. It is also known that due to the fact that there is a larger amount of image pairs, the following steps inevitably will take longer to process, especially in feature matching, as the matching of each feature is performed on each image pair. Nevertheless, the reduction of 76.86% of the image pairs in sequential matching compared to exhaustive is too large to be considered a good trade-off.

In feature matching, three different approaches are tested: simple feature matching, feature matching with cross-matching, and feature matching with guided matching. It has already been explained that cross-matching and guided matching will provide better matches than if they weren't used, as the false matches would be better filtered, and that the computational cost of using either is far greater due to performing the computation twice. The objective of this study is to compare the results and evaluate if any of the methods are worth using in ship reconstruction. A test compromising of using both methods at the same time was left out as execution time would take far too long, inhibiting the computer that is processing the data from performing any other task during that time. In Table 5.4, the results of the tests are displayed. For a better understanding of these results, a bar chart was developed for a more visual approach (Fig. 5.3).

Table 5.4: Table of results of the different feature matching methods.

Matching Method	**Execution Time (h)**		**Matched Features**	
	Matching	Filtering	Before Filter	After Filter
Simple	11.11	1.04	30767642	8043183
Guided Matching	11.63	26.13	30767642	9308021
Cross Matching	35.48	0.43	11076902	6057881

As can be seen, the number of matched features before filtering is the same when using guided matching or when just using the simple method, which would be expected as they are running the same algorithm. This number is significantly lower in cross-matching as the features are only considered a match if the matching from image A to B is also a match when matching from B to A. Thus, it also makes sense for the filtering process to take less time to execute as this method is prone to have fewer outliers compared to the other two methods, requiring to filter out fewer matches, as observed by the 45.31% reduction of matched features compared to the 69.75% and 73.86% reduction of the guided matching method and simple method, respectively. Even though cross-matching yields better accuracy due to the thorough matching process when compared to the simple method, it takes roughly 3 times more to compute and yields fewer matches. The same happens to guided matching, even though it does provide more matches after the filtering process that could be useful if not all images were registered during the last step of reconstruction (the Structure-from-Motion). From a computational cost standpoint, cross-matching is not

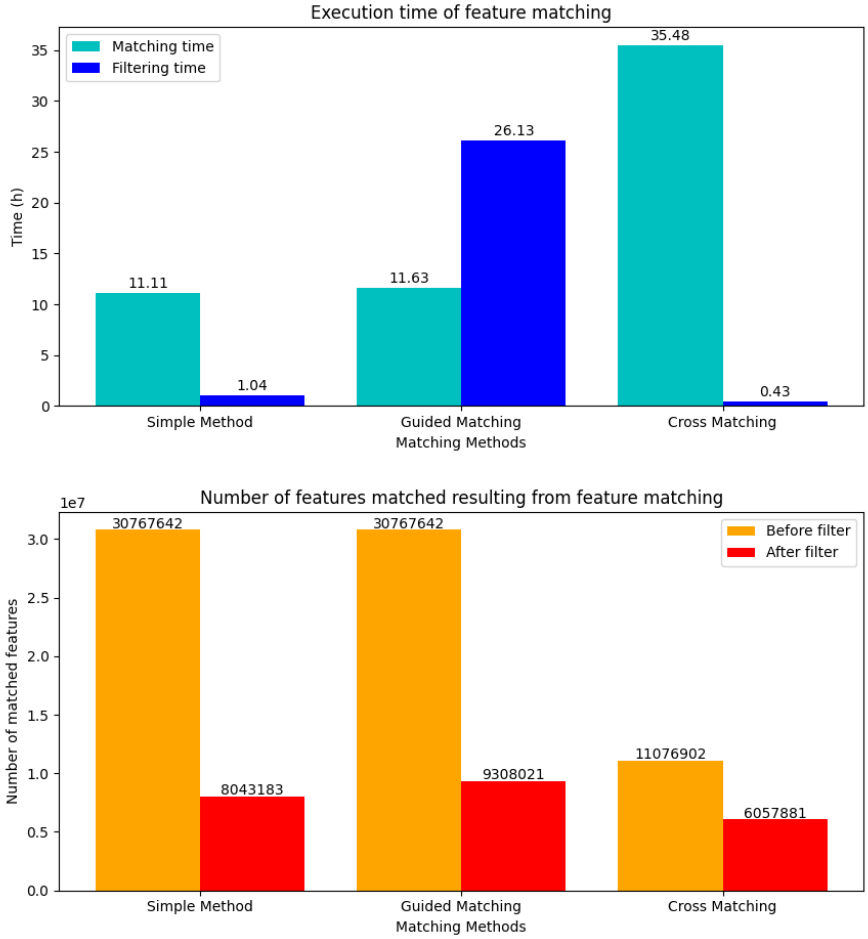


Figure 5.3: Bar graph of the number of features matched, prior and post filtering, as well as each execution time.

worth using, as well as guided matching. Thus, the simple method is the chosen method, and later on, after the last step, in case not all images are registered, guided matching will be preferred.

For the last step in the Meshroom pipeline, Structure-from-Motion was performed. As mentioned in Chapter 4, no specific test was performed on this step as Meshroom does not provide any relevant parameters to change. Therefore, just its performance is evaluated:

- **Cameras calibrated:** 406 / 406
- **Poses estimated:** 406 / 406
- **Number of points:** 569205
- **Execution Time:** 1923.38 s

From these results it can be noted that all input images were registered, meaning that applying guided matching during feature matching is not required, keeping total execution time lower. For a better understanding of the resulting point cloud, a visual inspection

is performed (Fig 5.4). As it can be seen, regions of the ship where heavy shadows are present are not properly reconstructed or not even reconstructed at all, leaving a "hole" in the point cloud, which is expected. Nevertheless, overall the ship is well reconstructed, presenting a lot of detail, especially on the parts that were well illuminated, with few point outliers surrounding the reconstructed ship.

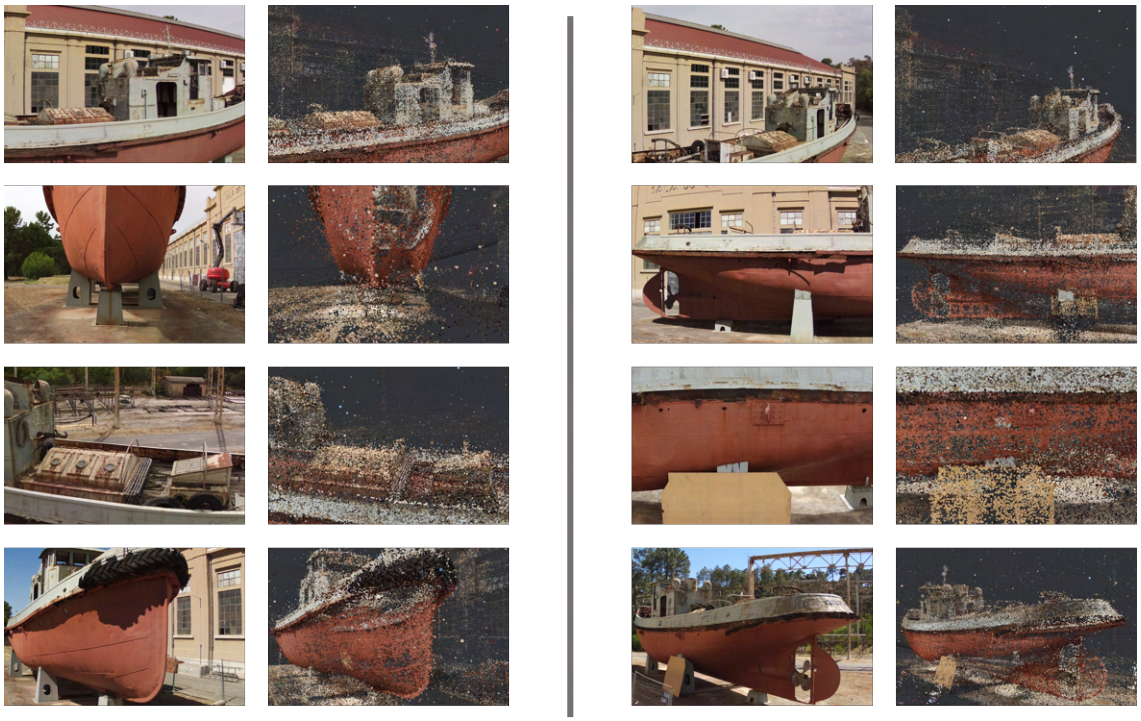


Figure 5.4: Showcase of the resulting point cloud in the same viewpoint as the input images.

5.3 Corrosion Classification and Representation Results

Throughout this section, an analysis of the results during the entire process of representing the corroded parts in the point cloud is presented. As mentioned previously, for this part Google Colab was used, in which it provided a Tesla T4 GPU.

Starting with the training process, the model was set to train for 10000 epochs. However, this process was stopped after 14 epochs due to the "early stopping" callback that stopped the training earlier, reaching an accuracy of 95.38% for the training dataset and 92.96% for the validation dataset. Using the obtained model, evaluating its performance on the test dataset achieved an accuracy of 93.27%, thus giving coherent results between the datasets. The evolution of the accuracy and the binary crossentropy of both the training and validation dataset throughout each epoch is shown in Fig. 5.5. The total training time was approximately 10 minutes. A sample of the classified images is presented in Fig. 5.6. From this figure, it is possible to observe that even though the model does a good job in

classifying, it is not 100% perfect, as the image in row one and column four was wrongly classified as having no corrosion present.

(a) Accuracy

(b) Binary Crossentropy

Figure 5.5: Graphs of the metrics' evolution during the training process for the training and validation dataset (orange line corresponds to training and blue line corresponds to validation).

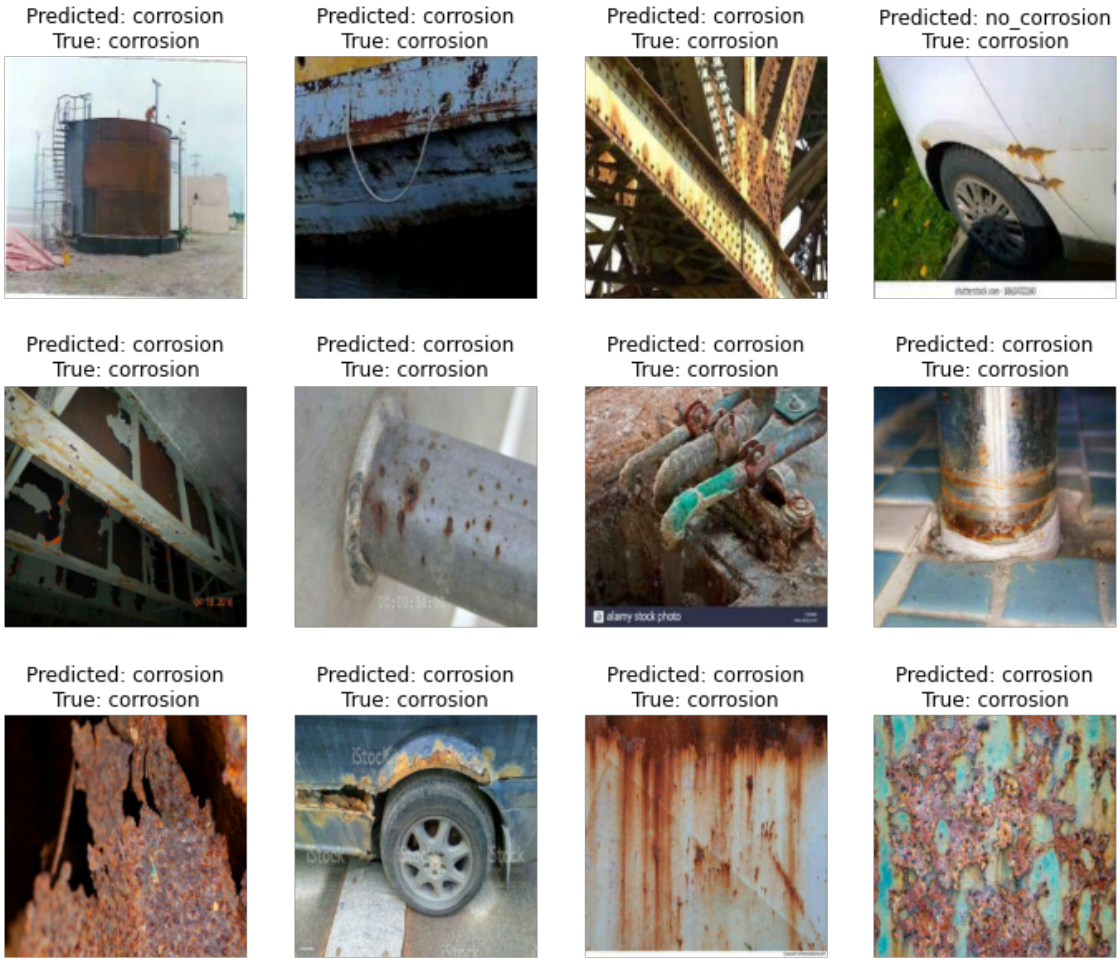


Figure 5.6: Batch of images from the test dataset classified by the model.

To further assess the accuracy of the classifier model, a binary confusion matrix was compiled for the entire test dataset, as depicted in Fig. 5.7. This matrix is composed of the

following elements:

- **True Positive (TP)**: labels predicted correctly as no corrosion. In this case, this resulted in 127 images correctly labeled (40.71%);
- **True Negative (TN)**: labels predicted correctly as corrosion. In this case, this resulted in 165 images correctly labeled (52.88%);
- **False Positive (FP)**: labels predicted incorrectly as no corrosion. In this case, this resulted in 9 images incorrectly labeled (2.88%);
- **False Negative (FN)**: labels predicted incorrectly as corrosion. In this case, this resulted in 11 images incorrectly labeled (3.53%);

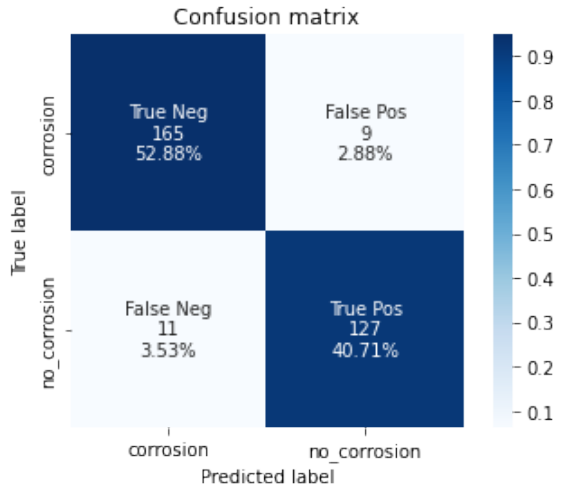


Figure 5.7: Confusion matrix resulted from the test dataset.

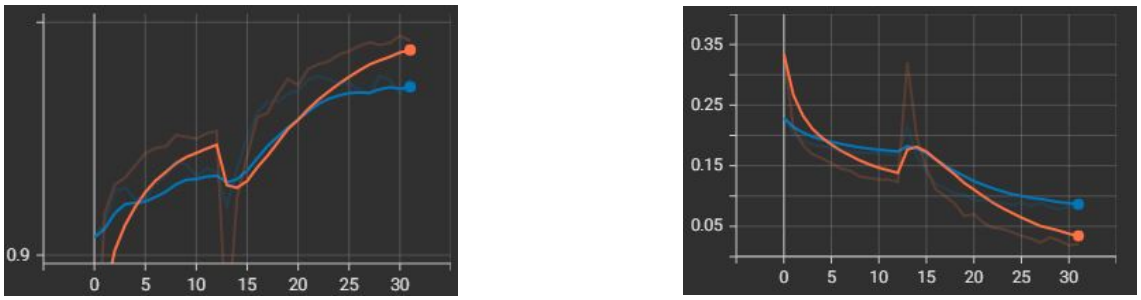
From these elements, more concise metrics can be extracted such as precision and recall. The precision (5.1) indicates the probability of a positive class to be predicted correctly, which in this case corresponds to the class with no corrosion. On the other hand, recall (5.2) indicates how well can the model correctly classify images with no corrosion given images in the positive and negative classes.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

The previous two equations result in a precision of approximately 93.38% and a recall of 92.03%. Therefore, the model has a significantly high capability of predicting correctly images where no corrosion is present, as well as when there are images of corrosion in the dataset.

To further improve the model, as mentioned in the previous chapter, fine-tuning is performed. The model started training from epoch 14, which was where it was left off in the first training, and trained until epoch 32, so a total of 18 epochs before the early stopping callback was invoked. Thus, in turn, achieved an accuracy of 99.18%, 97.44%, and 97.44% on the training, validation, and test dataset, respectively. It can be seen an improvement in all of the accuracies when compared to the previous results. The total training time took roughly 30 minutes, therefore an average of 100 seconds per epoch, whereas before it took an average of 38.6 seconds per epoch. This increased processing time is expected as the optimizer was set with a significantly lower value, consequently, the weights were updated with smaller changes and slower. The evolution of the accuracy and the binary crossentropy of both the training and validation dataset throughout each epoch is shown in Fig. 5.8.



(a) Accuracy (b) Binary Crossentropy

Figure 5.8: Graphs of the metrics' evolution during fine-tuning for the training and validation dataset (orange line corresponds to training and blue line corresponds to validation).

The same sample of images in Fig. 5.6 were run through this newly updated model, and its results are shown in Fig. 5.9. As can be observed, the image that was previously incorrectly classified is now correctly classified as corrosion.

As for the binary confusion matrix (Fig. 5.10), the results were as follows:

- **True Positive (TP)**: resulted in 135 images correctly labelled as no corrosion (43.27%);
- **True Negative (TN)**: resulted in 170 images correctly labelled as corrosion (54.49%);
- **False Positive (FP)**: resulted in 4 images incorrectly labelled as no corrosion (1.28%);
- **False Negative (FN)**: resulted in 3 images incorrectly labelled as corrosion (0.96%);

From this, it shows a clear improvement in the correctly labeled images, as TP and TN percentage has increased, as well as FP and FN percentage has decreased. The new precision is now 97.12% and the new recall is at 97.83%.

Afterward, the 406 images were divided into regions and each was classified accordingly, as explained in Chapter 4. This entire process took 1451.03s (approximately 24

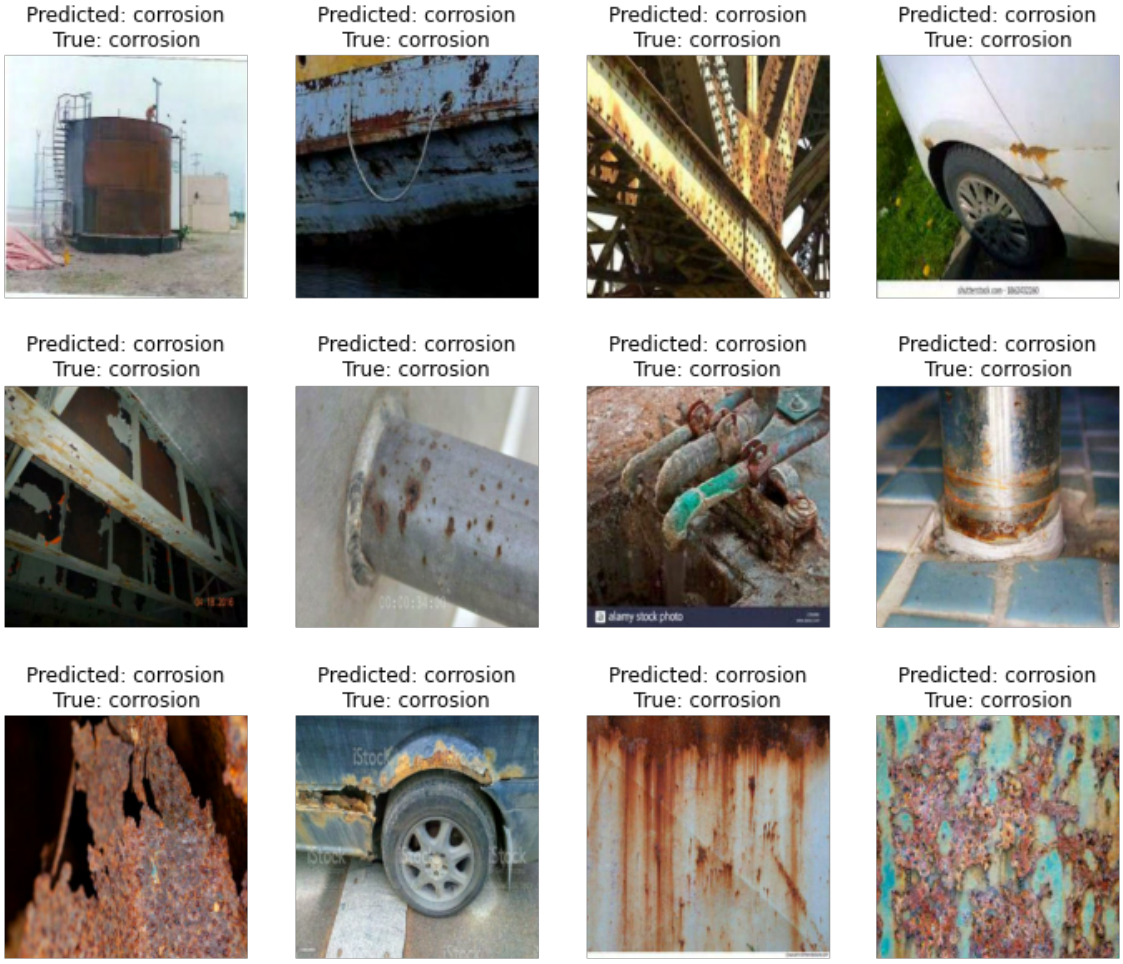


Figure 5.9: Batch of images from the test dataset classified by the fine-tuned model.

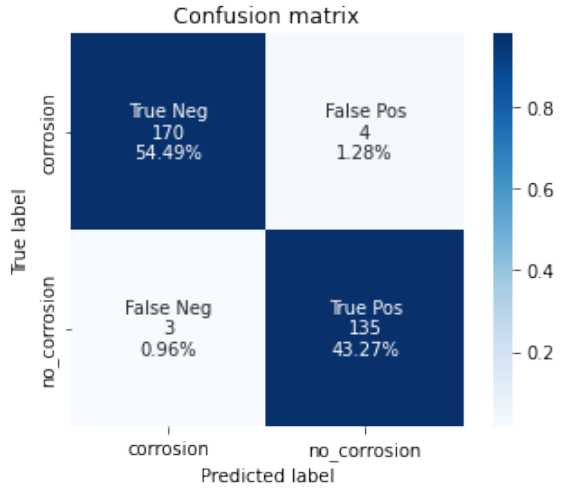


Figure 5.10: Confusion matrix resulted from the test dataset after fine-tuning the model.

minutes), thus 3.57s per image and roughly 0.09s to crop a region, normalize it, and classify. Fig. 5.11 presents an example.

Figure 5.11: Image divided into regions and labeled accordingly.

Lastly, to change the color of the points in the cloud, the algorithm presented in Fig. 4.14 is tested for different percentages besides 90% to verify and confirm its plausibility. For such, it was tested by changing the points' color at 70%, 80%, 90%, and 100% of the pixels in the regions classified as corrosion. The quantitative results are shown in Table 5.5.

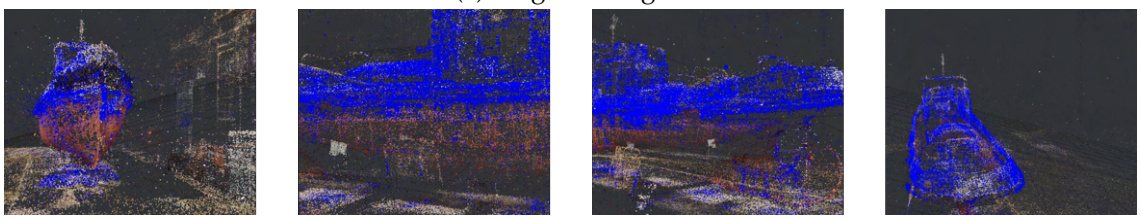
Table 5.5: Table of results of the different image matching methods.

Percentage	**Execution Time (s)**	**Points Changed**
70%	138.93	128480
80%	135.49	119013
90%	138.17	108289
100%	139.46	104634

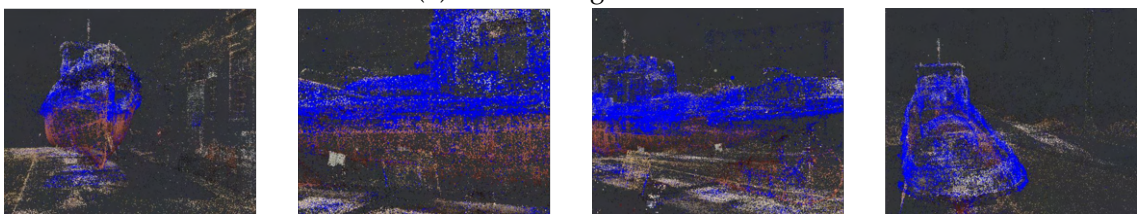
Analyzing these results, it is possible to verify that the number of points that are changed decreases as the required percentage to be considered corrosion increases, which would be expected. As for execution time, when 100% of the pixels that originated the point is located in a region classified as corrosion, it takes the longest even though fewer points are changed. This is also expected as all the pixels need to be checked, while the other cases most likely won't require to do so. Summarizing, from a perspective

of computational efficiency, 80% would be the optimal. Although, in a perspective of accuracy, 100% would be best as it requires that all pixels are in a region classified as corrosion, but it would also require the model to be perfect with 100%, which no model has ever been able to achieve such accuracy. Thus, using 90% of the points would be a good compromise between both. Nevertheless, to further confirm this choice, a qualitative study was performed, by visually inspecting the results shown in Fig. 5.12.

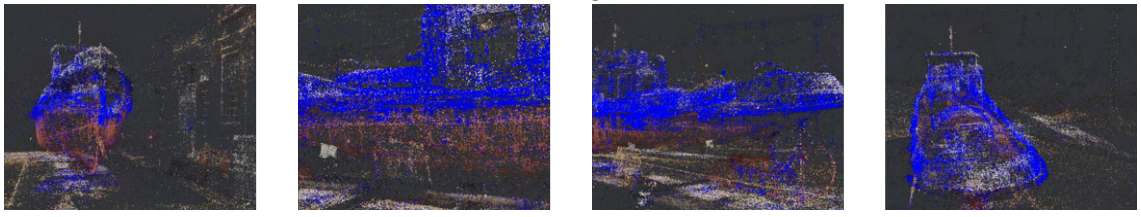
(a) Original images.



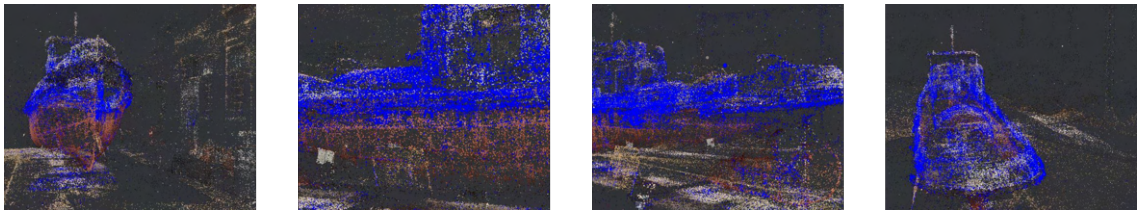
(b) Points changed at 70%.



(c) Points changed at 80%.



(d) Points changed at 90%.



(e) Points changed at 100%.

Figure 5.12: Comparison of the different point clouds when color is changed at different percentages.

For the point cloud with a threshold of 70% (Fig. 5.12b), compared with the original images (Fig. 5.12a), it can be seen that more parts of the ship are incorrectly changed color

compared with the remaining thresholds, especially in the bottom red areas of the ship where there are no corrosion have their corresponding points changed to blue. As for the threshold at 80% (Fig. 5.12c), this problem is attenuated. Although, in the white part at the back of the ship where there is no corrosion, more points are identified as corrosion present when compared to 90% and 100% (Figs. 5.12d and 5.12e). Comparing the results from the thresholds at 90% and 100%, visual differences cannot be seen. Concluding, the 90% threshold has proven to be a decent choice.

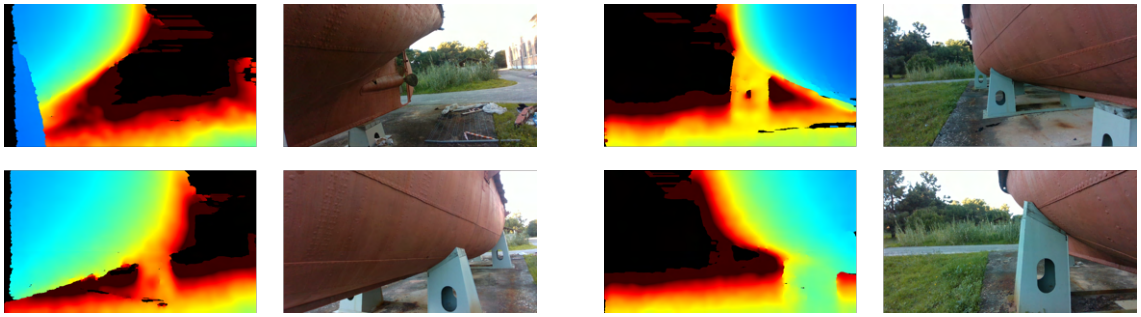
5.4 RGB-D Reconstruction Results

In this section, the computational speed and accuracy of the 3D reconstructed ship, as well as some other relevant parameters, are presented. The reconstruction program was written in Python language and ran through the Visual Studio Code software, using the same computer as indicated in Sec. 5.2.

As mentioned previously, only the parts of the ship where the SfM model would benefit from improvements were created, as adaptations had to be done to this project due to the construction work materials that obstruct the passage while walking and recording the ship. From inspection of the model, it can be seen that three parts of the model could be improved: the front-right side where there's a gap, the support structures all around the ship, and the back side, especially on top of the turbine. Therefore, to solve these issues, two models were created, a partial reconstruction of the right side of the ship, as well as a partial reconstruction of the left side that includes those problematic areas.

While recording the ship, it was obtained 766 RGB-D frames for the right side, therefore, resulting in 8 local geometric structure point clouds, where the first 7 are composed of 100 frames and the last of 66 frames. As for the left side, 907 frames were extracted, resulting in 10 local structure point clouds, again with the first 9 resulting from 100 RGB-D frames and the last by 7. The average time it took to extract each frame was around 0.12s. A sample of RGB-D frames is shown in Fig. 5.13. As it can be seen in Fig. 5.13a in the top depth image, there are artifacts on the upper right side of the image. This is due to the susceptibility to noise caused by sunlight that the depth sensor in the device D435i has. Precautions were taken to diminish these artifacts, such as adjusting the sensor's parameters like exposure and laser power, as well as performing the scanning later in the day when the sunlight was less bright, which dramatically improved the recording's quality, but still did not remove completely these artifacts. The problem with these artifacts is that they decrease the accuracy during the registration process and add unwanted structures to the point cloud that are non-existing in real life, as will be seen later on.

Following the extraction of frames, a pose graph was created and optimized and based on the resulting poses, a TSDF integration was performed. The execution time and the number of nodes and edges in each pose graph (robust and optimized) for each local structure corresponding to the left side is presented in Table A.1 available in Appendix A, as well as an average of each parameter, omitting for the last structure due to its



(a) (b)

Figure 5.13: Sample of RGB-D images of the ship. a) From the right side. b) From the left side.

particularity, which roughly represents the data as a whole. The results for the right side are not presented since they are redundant in terms of the results obtained and will not be shown throughout this entire section, where only the final model will be depicted. Thus, from now onwards, all results pertain to the left side of the ship.

As mentioned previously, seeing that the code allows for parallelization, 8 threads were running simultaneously to compute a structure, as it was the maximum number of processors available in the computer. This allowed for the total process to take 1253.462s to process, or in other words, roughly 21 min. To test the difference in total computational cost, the local structures were computed once again using only 1 thread. The results can be seen in the graph of Fig. 5.14. It is clear that there is a huge time efficiency when using multiple processors simultaneously, taking approximately 4.3 times less than using a single processor. This concludes that more processors equals to less computation time, up to a point, where more does not bring any greater efficiency and increases the cost of the hardware needlessly. Although, one other thing can be observed. By summing up all execution times shown in Table A.1, it amounts to approximately 7809.309s, when in fact using a single thread, it resulted in a computational time of 5392.032s, thus 30.95% faster. This is due to fewer computer resources being used simultaneously (which happens when using parallelization), thus each fragment takes less time to be computed.

One other factor to notice is the number of edges in the pose graph prior to and post the optimization process. There is a reduction, on average, of 56.73% of the edges, meaning that more than half of the transformations calculated at first are incorrect and incoherent with reality. This shows the importance of optimizing the pose graph for accurate results.

Proceeding to the registration of all local structures, from attempting to register 45 pairs (resulting from registering all structure point clouds with each other), only 9 were successful in obtaining the robust transformation, which corresponded to the point cloud pairs that were consecutive in space to one another. A sample of the process of robustly aligning the first two local structures is depicted in Fig. 5.15. As can be seen in the mentioned figure, in the raw point clouds, the individual local structures present some artifacts caused by the sensor, but nonetheless, the program performs well in registering

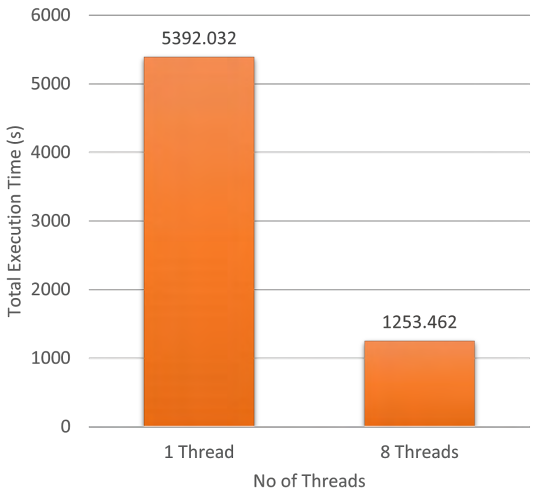


Figure 5.14: Comparison of total execution time between single thread and 8 threads.

both point clouds.

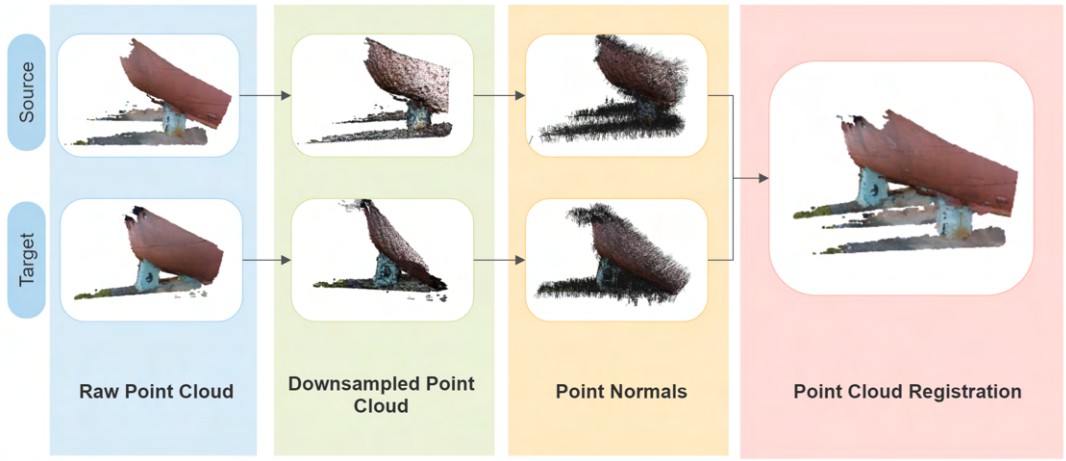


Figure 5.15: Results from registering the first two local geometric point clouds.

The average execution time for determining the robust transformation of a pair of point clouds, as well as refining it, is shown in Table 5.6. Once again, a comparison between a single thread and 8 threads is performed. As can be expected, using a single thread performs significantly faster for registering a single point cloud pair, by an average of 5.6 times, for reasons mentioned previously. Although, overall execution time is decreased by an average of approximately 1.3 times when using multiple threads, as seen in Table 5.7, which is a significant reduction in efficiency when compared to creating the local geometric structure point clouds, which was 4.3 times faster. This shows that for registering the local structures, using multiple threads is not as important for time efficiency as for creating the structures itself. Furthermore, it can be observed that the number of edges did not decrease after the optimization process, which shows that the method to determine the transformation between the source and target point clouds performs accurately. This can

also be due to the fact that there were only odometry edges and no loop closure edges due to the lack of overlap between non-consecutive point clouds. Table 5.7 does not show TSDF integration results for the robust alignment as it was only performed after refining the transformations.

Table 5.6: Table showcasing the average execution time for registering a pair of point clouds using a single thread and 8 threads.

Method	**Average Execution Time per Pair (s)**	
	1 Thread	8 Threads
Robust	05.868	38.061
Refine	17.524	82.654

Table 5.7: Table of results of the creation of all local geometric structure point clouds for the left side of the ship.

		Robust	**Refine**
Pose Graph	Execution Time (s) with 1 Thread	272.215	166.093
	Execution Time (s) with 8 Thread	226.508	127.396
	No. of Nodes	10	10
	No. of Edges	9	9
Optimization	Execution Time (s)	0.001	0.000
	No. of Nodes	10	10
	No. of Edges	9	9
TSDF	Execution Time (s)	-	315.887

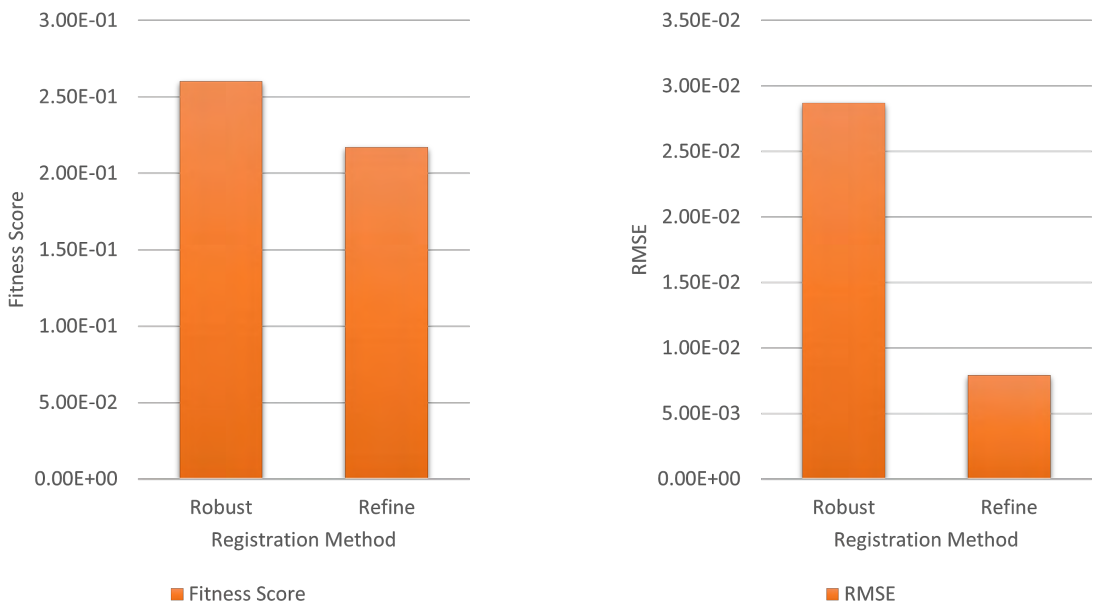
In Fig. 5.16, a visual comparison between the robustly aligned point cloud pair and the refined one is performed. Visually, not much difference can be perceived. Although, the definition of the edges of the ship is slightly sharper on the refined model than on the robust model.

To further analyze the transformations accuracy, the average fitness score and the RMSE (Fig. 5.17) were investigated, which measures the area of the source and target point cloud that is overlapping and the RMSE of the inliers, respectively. As it can be observed, the robust aligned cloud has a slightly better average fitness score than the refine aligned point cloud, which is not the best case as it should have been worse. Nevertheless,

the difference is not much and since it is an average, an outlier can skew this value. Although, from the RMSE, it is possible to observe a large decrease, which shows that the transformation estimation errors were largely reduced after the refining process.

(a) (b)

Figure 5.16: Registration of the first point cloud pair of the left side of the ship. a) Fitness Score. b) RMSE.



(a) (b)

Figure 5.17: Average evaluation metrics of the registration process of the left side of the ship. a) Fitness Score. b) RMSE.

The final model for the right side and left side of the ship is presented in Fig. 5.18. The model pertaining to the right side is composed of 20083404 points and the left side is composed of 20922009 points. Comparing these models with the one obtained from SfM, there is a clear difference in the density of the model. The SfM model is composed of 569205 points, which includes the ship, the structures surrounding it, and the ground, while the model from RGB-D of either side has at least 35 times more points and only has

a part of the ship and ground.

(a) (b)

Figure 5.18: Left and right side model of the ship obtained from RGB-D images.

Despite the model looking quite accurate, in Fig. 5.18a, the closest support structure of the ship has some misalignment. The front of the real ship was where it was most illuminated during the recording process, which is where this structure is located, thus the noise caused by this stronger illumination created some artifacts in the depth image, as shown at the beginning of this section, which most likely caused the registration to be not as accurate as the rest of the ship. In addition, it can be observed that some points are outliers, but can be easily eliminated with filtering. Therefore, to clear up these outliers and remove the unnecessary ground plane, Meshlab was used to apply filtering processes and obtain a more clean and accurate model. Since this process is more for a visually appealing model presentation and not the model reconstruction *per se*, the exact steps for cleaning it will not be explained. Just for the interest of the reader of the tools used, manual removal of the ground points was performed and the Meshlab filter tools such as "Close Holes", "Remove Outliers", and "Laplacian Smooth" were used. The final models are presented in Fig. 5.19.

(a) (b)

Figure 5.19: Left and right side model of the ship after filtering.

5.5 Merging and Meshing Results

As can be predicted, due to the many manual tasks done for this part of the project, neither time efficiency nor increased precision could be achieved, which was one of the main objectives. Thus, these metrics were discarded when analyzing the process and the focus was shifted into more of a visual analysis.

In Fig. A.1 in the Appendix A, it is possible to compare the point clouds belonging to the SfM model, without the corrosion highlighted, and to the model after aligning all point clouds together. As can be seen, by merging the point clouds, it increased its density on some zones, with a focus on the support structures where the density was not as elevated as the sides of the ship and, furthermore, fixed occluded areas of the ship that were not caught, especially on the back side of the ship where the turbine is located, as it can be observed in the second row of the image. Comparing the number of points of the filtered models (without surrounding structures or ground), the SfM point cloud is composed of 286718 points, while the merged point cloud is composed of 1666165 points, which equates to roughly a 481% increase in point number.

After applying the meshing process, the final ship with the corroded parts highlighted in blue is obtained and its results can be visualized in Fig. 5.20. This process, in which the surface of the ship was reconstructed, derived a total of 2388229 faces. By visually analyzing the mesh model, it is possible to observe that, overall, it represents closely the real-life ship. In the merging process, the ship antenna lost some detail and there's a slight color shift but, nevertheless, the lacking areas of the SfM ship were largely improved with the RGB-D model, except for the front support due to reasons explained in Sec. 5.5 and outside the author's control, thus resulting in this fairly accurate model.

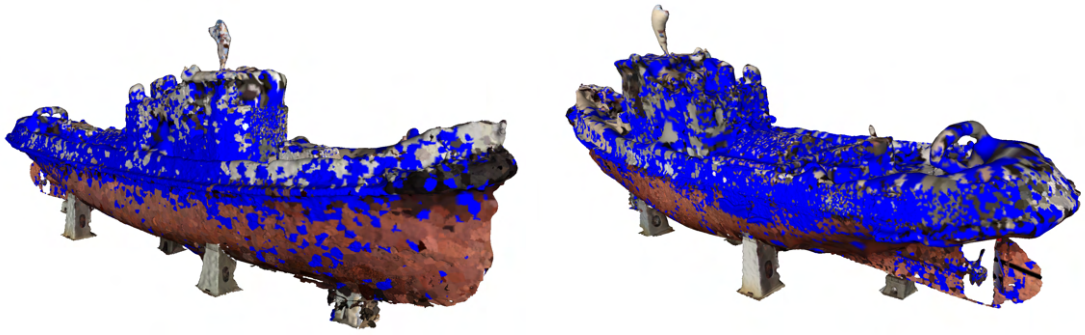


Figure 5.20: Final mesh model of the ship, from different viewpoints.

6 Conclusion

6.1 Conclusion

With the growing interest in computer vision, sensors such as RGB-D and common RGB cameras have been evolving, resulting in better technology and lower prices. 3D reconstruction has also been an area of increased interest, as it is useful in many topics such as gaming, city mapping, navigation, etc. With this evolution in sensors, it translates directly to 3D reconstruction, accuracy and efficiency-wise. Furthermore, the increased usage of systems that exploit machine learning to perform what was once a human-driven job has also facilitated such jobs to be performed more efficiently. A human inspector surveying the ship and searching for corroded parts can be quite time-consuming and inefficient.

This dissertation proposes a 3D reconstruction method that fuses aerial RGB data with terrestrial RGB-D data to reconstruct a ship. Additionally, the system integrates a corrosion classification algorithm that highlights the corroded parts of the ship in the 3D model. To develop this elaborate system, it was divided into four smaller steps: reconstruct the ship using just the RGB sensor, create a corrosion classification model and highlight the areas corroded in the RGB-based model, reconstruct the same ship using RGB-D sensor, and lastly merge both created models. As was mentioned throughout this dissertation, due to unexpected construction work nearby the ship, some proposed methods were limited and had to be adapted and readjusted to the new conditions.

Structure from Motion was the algorithm used to reconstruct the ship using the incorporated RGB camera in the UAV. The entire workflow was performed using Meshroom software and a benchmark of the different algorithms for each step in the workflow was conducted. It was concluded that for the best tradeoff between accuracy and computational speed, using both SIFT and DSP-SIFT for feature extraction, the sequential method for image matching, and the simple method for feature matching, was the best choice, which resulted in a visually equal model of the real-life ship, with some parts lacking points due to being occluded.

To detect areas of the ship that were corroded, a corrosion classification model was developed in Google Colab. The model after fine-tuning achieved an accuracy of 97.44% on the test dataset, a precision of 97.12%, and a recall of 97.83%, which demonstrates that this model has a high capability of classifying corrosion images correctly. Since the

ship images that were fed to the classification model were the ones taken for the SfM model and these images capture the entire ship in a single image, a small portion of the ship that could be corroded would translate to the whole ship being identified as corroded. Therefore, each image was divided into 40 equally divided regions and fed to the classification model. If 90% of the total dataset image regions relating to a specific part of the ship were labeled as corroded, the SfM model would change its points to blue to highlight the corroded part, making this somewhat a fusion of corrosion classification and detection. The algorithm proved to be efficient and fairly accurate in classifying the regions accordingly.

The RGB-D model complemented the SfM model in terms of adding information about the occluded zones. Two distinct 3D models were created with the RGB-D data, one pertaining to the right side of the ship, obtained from 766 RGB-D frames, and another pertaining to the left side, using 907 frames. The process of integrating and registering all frames taken with the sensor and converting the images into a point cloud exploited the functions available in the Open3D library. Since the sensor used to capture the images is very susceptible to light noise, a small zone of the 3D point cloud models was not accurately portraying the real-life ship, more concretely the front support structure, but overall were quite similar.

Lastly, the merging process for the SfM model with corrosion highlighted and both RGB-D models was proposed. Since the models were reconstructed using different technologies and, therefore, have different densities and noise, the common feature-based registration for all the points clouds was not an option. Using PCA to project the clouds into eigenvector space and applying a fusion of automated and manual tasks to register all the models proved to achieve the desired result, demonstrating the advantage of using different types of sensors from different viewpoints, complementing one another and solving issues that one or the other may have. In this case, the SfM model captured the top of the ship where the RGB-D models could not capture, the RGB-D models capture the bottom of the ship which was occluded in the SfM model, and even increased significantly the density of the support structures which were very poor in the SfM model.

6.2 Future Work

This project has a good base for ship multimodel 3D reconstruction and corrosion detection but there are a few components that would benefit from further testing, improvements, or even added functionalities. The obvious first task would be re-testing the proposed model presented in this document after the construction work has finished to validate the results without interference. Without these interferences, the final model would most likely improve its accuracy relative to the real-life ship and, furthermore, the manual tasks performed could be automated as proposed, thus improving reconstruction speed as well.

A study on the difference in accuracy and detection speed of corrosion of a multispectral camera and a common RGB camera could be performed. Due to the lack of available

multispectral image datasets of corrosion, this could not be performed on this project as it would be time-consuming, but a further study on this area can be quite advantageous to choose the better sensor for the job when surveying the ship.

Another important upgrade that could be performed would be switching from corrosion classification to corrosion detection. This would lead to a more accurate highlighting of the corroded parts in the ship that were represented in blue to more concise areas of the ship which, again, would improve overall accuracy.

Incorporating a GPS RTK, which has an accuracy of 1 cm, with the RGB-D sensor could also be explored. Instead of using feature-based registration, adding the coordinates of each frame to its metadata along with the orientation of the sensor, which Realsense D435i provides, and converting all into UTM to transform each RGB-D reading to the correct pose in the coordinate system, registration could be performed and with possible added benefits of decreased computational cost.

Lastly, it would be interesting to add the ability to autonomously survey the ship. Instead of using a remote control to move the UAV around the ship, the system would be able to detect the ship and its location and circle it while recording it from different viewpoints.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf.

[2] L. Sun et al. "Real-Time Fusion Network for RGB-D Semantic Segmentation Incorporating Unexpected Obstacle Detection for Road-Driving Images". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5558–5565. DOI: 10.1109/LRA.2020.3007457.

[3] L. Li, K. Ota, and M. Dong. "Humanlike Driving: Empirical Decision-Making System for Autonomous Vehicles". In: *IEEE Transactions on Vehicular Technology* 67.8 (2018), pp. 6814–6823. DOI: 10.1109/TVT.2018.2822762.

[4] S. Gatesichapakorn, J. Takamatsu, and M. Ruchanurucks. "ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera". In: *2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*. 2019, pp. 151–154. DOI: 10.1109/ICA-SYMP.2019.8645984.

[5] H. Zhang, L. Jin, and C. Ye. "An RGB-D Camera Based Visual Positioning System for Assistive Navigation by a Robotic Navigation Aid". In: *IEEE/CAA Journal of Automatica Sinica* 8.8 (2021), pp. 1389–1400. DOI: 10.1109/JAS.2021.1004084.

[6] X. Qi et al. "Object Semantic Grid Mapping with 2D LiDAR and RGB-D Camera for Domestic Robot Navigation". In: *Applied Sciences* 10.17 (2020-08), p. 5782. ISSN: 2076-3417. DOI: 10.3390/app10175782. URL: http://dx.doi.org/10.3390/app10175782.

[7] Y. Liu et al. "RGB-D Human Action Recognition of Deep Feature Enhancement and Fusion Using Two-Stream ConvNet". In: *Journal of Sensors* 2021 (2021). ISSN: 16877268. DOI: 10.1155/2021/8864870.

[8] A. Barkoky and N. M. Charkari. "Complex Network-based features extraction in RGB-D human action recognition". In: *Journal of Visual Communication and Image Representation* 82 (2022), p. 103371. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2021.103371. URL: https://www.sciencedirect.com/science/article/pii/S1047320321002455.

[9] Framos. *DEPTH SENSING TECHNOLOGIES OVERVIEW*. https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies, Last accessed on 2023-03-27.

[10] M. Jaboyedoff et al. "Use of LIDAR in landslide investigations: A review". In: *Natural Hazards* 61 (2012-03), pp. 5–28. DOI: 10.1007/s11069-010-9634-2.

[11] C. Qi and D. Pinliang. *LiDAR Remote Sensing and Applications*. CRC Press, 2018, pp. 1–200. ISBN: 978-1-4822-4301-7.

[12] A. Al-Rawabdeh, F. He, and A. Habib. "Automated feature-based down-sampling approaches for fine registration of irregular point clouds". In: *Remote Sensing* 12 (7 2020-04). ISSN: 20724292. DOI: 10.3390/rs12071224.

[13] P. B. L. Bahman Ghiassi. *Long-term Performance and Durability of Masonry Structures*. 1st ed. Woodhead Publishing, 2019, pp. 1–404. ISBN: 9780081021101. DOI: 10.1016/C2016-0-03710-5.

[14] Open3D. *Point cloud outlier removal*. http://www.open3d.org/docs/latest/tutorial/Advanced/pointcloud_outlier_removal.html, Last accessed on 2022-02-01.

[15] R. Schnabel, R. Wahl, and R. Klein. "Efficient RANSAC for point-cloud shape detection". In: *Computer graphics forum*. Vol. 26. 2. Wiley Online Library. 2007, pp. 214–226.

[16] M. Weinmann. *Reconstruction and analysis of 3D scenes: From irregularly distributed 3D points to object classes*. 1st ed. Springer International Publishing, 2016, pp. 1–233. ISBN: 9783319292465. DOI: 10.1007/978-3-319-29246-5.

[17] C. Harris and M. Stephens. "A Combined Corner and Edge Detector". In: British Machine Vision Association and Society for Pattern Recognition, 1988-04, pp. 23.1–23.6. DOI: 10.5244/c.2.23.

[18] I. Sipiran and B. Bustos. "Harris 3D: A robust extension of the Harris operator for interest point detection on 3D meshes". In: vol. 27. 2011-11, pp. 963–976. DOI: 10.1007/s00371-011-0610-y.

[19] Z. Yu. "Intrinsic shape signatures: A shape descriptor for 3D object recognition". In: 2009, pp. 689–696. ISBN: 9781424444427. DOI: 10.1109/ICCVW.2009.5457637.

[20] P. C. Library. *Fast Point Feature Histograms (FPFH) descriptors*. https://pcl.readthedocs.io/projects/tutorials/en/latest/fpfh_estimation.html?highlight=Fast%20Point%20Feature%20Histogram, Last accessed on 2022-02-02.

[21] X.-F. Han et al. "A comprehensive review of 3d point cloud descriptors". In: *arXiv preprint arXiv:1802.02297* 2 (2018).

[22] F. Tombari, S. Salti, and L. D. Stefano. *Unique Signatures of Histograms for Local Surface Description*. 2010. URL: http://www.vision.deis.unibo.it.

[23] S. Liu et al. *3D Point Cloud Analysis - Traditional, Deep Learning, and Explainable Machine Learning Methods*. Springer International Publishing, 2021. ISBN: 978-3-030-89179-4. DOI: 10.1007/978-3-030-89180-0. URL: https://link.springer.com/10.1007/978-3-030-89180-0.

[24] A. Nguyen and B. Le. "3D point cloud segmentation: A survey". In: *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE. 2013, pp. 225–230. ISBN: 9781479912018.

[25] ShapeNet. *Large-scale 3D Datasets*. https://shapenet.org/, Last accessed on 2022-01-30.

[26] ModelNet. *Princeton ModelNet*. http://modelnet.cs.princeton.edu/, Last accessed on 2022-01-30.

[27] A. Koutsoudis, F. Arnaoutoglou, and C. Chamzas. "On 3D reconstruction of the old city of Xanthi. A minimum budget approach to virtual touring based on photogrammetry". In: *Journal of Cultural Heritage* 8 (1 2007), pp. 26–31. ISSN: 12962074. DOI: 10.1016/j.culher.2006.08.003.

[28] D. Ferdani et al. "3D reconstruction and validation of historical background for immersive VR applications and games: The case study of the Forum of Augustus in Rome". In: *Journal of Cultural Heritage* 43 (2020-05), pp. 129–143. ISSN: 12962074. DOI: 10.1016/j.culher.2019.12.004.

[29] S. Zhao et al. "Structural health monitoring and inspection of dams based on UAV photogrammetry with image 3D reconstruction". In: *Automation in Construction* 130 (2021-10). ISSN: 09265805. DOI: 10.1016/j.autcon.2021.103832.

[30] C. Balletti et al. "3D reconstruction of marble shipwreck cargoes based on underwater multi-image photogrammetry". In: *Digital Applications in Archaeology and Cultural Heritage* 3 (1 2016), pp. 1–8. ISSN: 22120548. DOI: 10.1016/j.daach.2015.11.003.

[31] D. G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[32] H. Bay, T. Tuytelaars, and L. V. Gool. *SURF: Speeded Up Robust Features*. Vol. 110. Springer, Berlin, Heidelberg, 2008, pp. 346–359. ISBN: 978-3-540-33833-8. DOI: 10.1007/11744023_32.

[33] J. P. Mueller and L. Massaron. *Deep Learning*. Vol. 1. John Wiley Sons, Inc, 2019, pp. 1–334. ISBN: 978-1-119-54304-6.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[35] C. Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[36] C. Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[37] M. Tan and Q. V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[38] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[39] K. He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[40] J. Liang et al. "A new power-line extraction method based on airborne LiDAR point cloud data". In: 2011. ISBN: 9781457709692. DOI: 10.1109/ISIDF.2011.6024293.

[41] L. Cheng et al. "Three-dimensional reconstruction of large multilayer interchange bridge using airborne LiDAR data". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8 (2 2015-02), pp. 691–708. ISSN: 21511535. DOI: 10.1109/JSTARS.2014.2363463.

[42] B. Wu et al. "A graph-based approach for 3d building model reconstruction from airborne lidar point clouds". In: *Remote Sensing* 9 (1 2017). ISSN: 20724292. DOI: 10.3390/rs9010092.

[43] Y. Pan et al. "Three-dimensional reconstruction of structural surface model of heritage bridges using UAV-based photogrammetric point clouds". In: *Remote Sensing* 11 (10 2019-05). ISSN: 20724292. DOI: 10.3390/rs11101204.

[44] Y. Qu, J. Huang, and X. Zhang. "Rapid 3D reconstruction for image sequence acquired from UAV camera". In: *Sensors (Switzerland)* 18 (1 2018-01). ISSN: 14248220. DOI: 10.3390/s18010225.

[45] Z. Xu et al. "Tridimensional reconstruction applied to cultural heritage with the use of camera-equipped UAV and terrestrial laser scanner". In: *Remote Sensing* 6 (11 2014), pp. 10413–10434. ISSN: 20724292. DOI: 10.3390/rs61110413.

[46] B. T. Bastian et al. "Visual inspection and characterization of external corrosion in pipelines using deep neural network". In: *NDT and E International* 107 (2019-10). ISSN: 09638695. DOI: 10.1016/j.ndteint.2019.102134.

[47] Y. J. Cha et al. "Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types". In: *Computer-Aided Civil and Infrastructure Engineering* 33 (9 2018-09), pp. 731–747. ISSN: 14678667. DOI: 10.1111/mice.12334.

[48] J. L. Schönberger and J.-M. Frahm. *Structure-from-Motion Revisited*. URL: https://github.com/colmap/colmap..

[49] X. Huang et al. "A Coarse-to-Fine Algorithm for Registration in 3D Street-View Cross-Source Point Clouds". In: *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2016, pp. 1–6. DOI: 10.1109/DICTA.2016.7796986.

[50] M. .-. C. Cheng. *Principal Component Analysis (PCA) Explained Visually with Zero Math*. https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d, Last accessed on 2023-03-14.

[51] dji. *P4 Multispectral Specs*. https://www.dji.com/pt/p4-multispectral/specs, Last accessed on 2022-08-10.

[52] Intel. *Intel® RealSense™ Depth Camera D435i*. https://www.intelrealsense.com/depth-camera-d435i/, Last accessed on 2022-08-11.

[53] DJI. *P4 Multispectral - Plant Intelligence for Targeted Action*. https://www.dji.com/pt/p4-multispectral, Last accessed on 2022-08-10.

[54] Google. *Welcome To Colaboratory*. https://colab.research.google.com/, Last accessed on 2022-09-05.

[55] Google. *Tensorflow*. https://www.tensorflow.org/, Last accessed on 2022-08-11.

[56] Roboflow. *Give your software the sense of sight*. https://roboflow.com, Last accessed on 2022-08-12.

[57] AliceVision. *Meshroom*. https://github.com/alicevision/Meshroom, Last accessed on 2022-08-31.

[58] P. C. Library. *PCL*. https://pointclouds.org/, Last accessed on 2022-08-12.

[59] Q.-Y. Zhou, J. Park, and V. Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[60] Intel. *Intel RealSense SDK 2.0*. https://www.intelrealsense.com/sdk-2/, Last accessed on 2023-02-13.

[61] P. Cignoni et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by V. Scarano, R. D. Chiara, and U. Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[62] D. G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. 2004, pp. 91–110.

[63] J. Dong and S. Soatto. "Domain-size pooling in local descriptors: DSP-SIFT". In: vol. 07-12-June-2015. IEEE Computer Society, 2015-10, pp. 5097–5106. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7299145.

[64] P. F. Alcantarilla, A. Bartoli, and J. Nuevo. "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces". In: (2013).

[65] S. Choi, Q.-Y. Zhou, and V. Koltun. "Robust reconstruction of indoor scenes". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5556–5565. DOI: 10.1109/CVPR.2015.7299195.

[66] J. Park, Q.-Y. Zhou, and V. Koltun. "Colored Point Cloud Registration Revisited". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 143–152. DOI: 10.1109/ICCV.2017.25.

[67] D. Werner, A. Al-Hamadi, and P. Werner. "Truncated Signed Distance Function: Experiments on Voxel Size". In: *Image Analysis and Recognition*. Ed. by A. Campilho and M. Kamel. Cham: Springer International Publishing, 2014, pp. 357–364. ISBN: 978-3-319-11755-3.

[68] Google. *Kaggle*. https://www.kaggle.com/, Last accessed on 2022-08-28.

[69] Pexels. *The best free stock photos, royalty free images videos shared by creators.* https://www.pexels.com/, Last accessed on 2022-08-28.

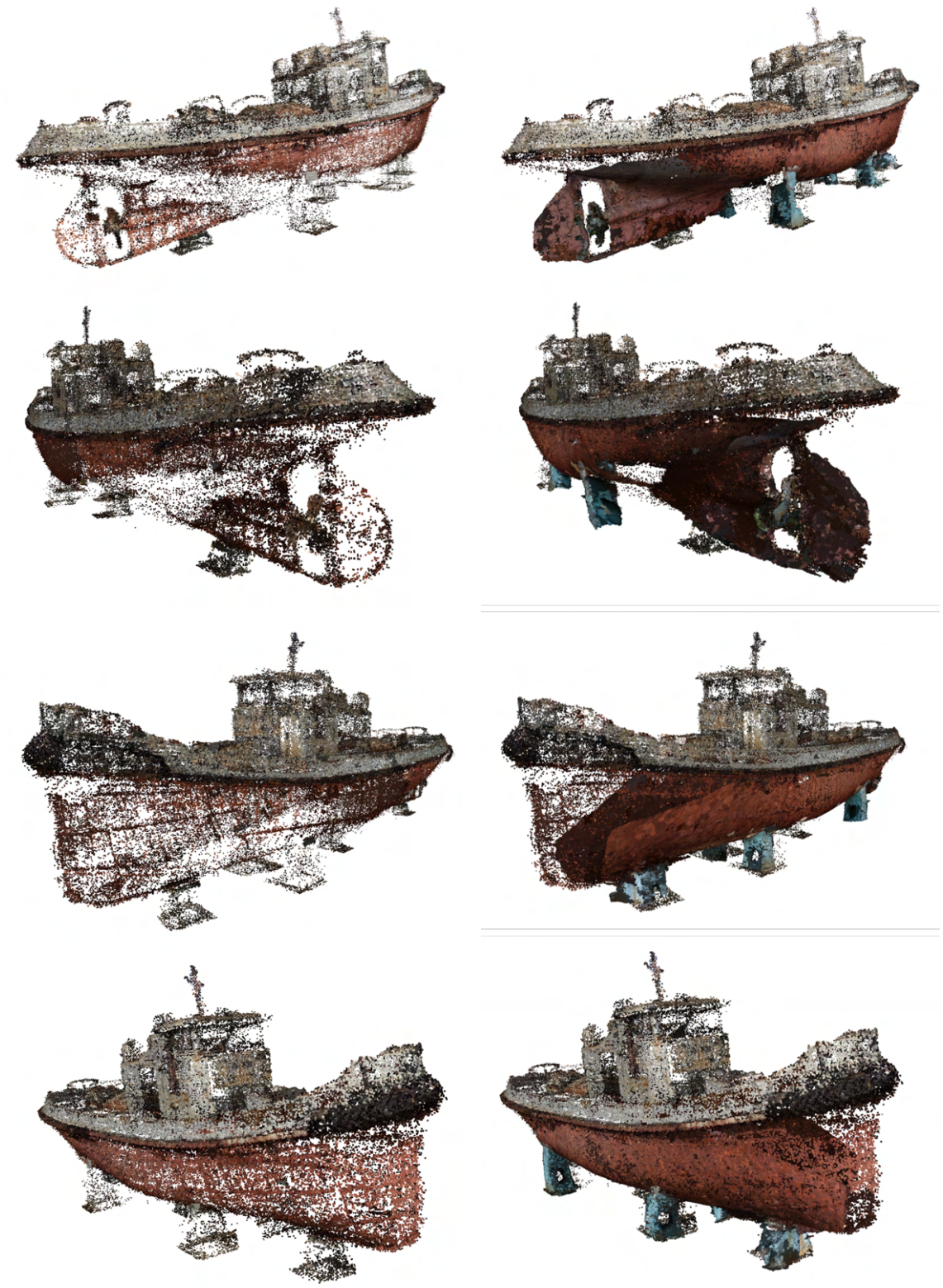
[70] Medium. *Review — EfficientNetV2: Smaller Models and Faster Training*. https://medium.com/aiguys/review-efficientnetv2-smaller-models-and-faster-training-47d4215dcdfb, Last accessed on 2022-09-01.

[71] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference for Learning Representations* (2015-12). URL: http://arxiv.org/abs/1412.6980.

A Appendix

Table A.1: Table of results of the creation of all local geometric structure point clouds for the left side of the ship.

Fragment No.	**Pose Graph**			**Pose Graph Optimization**			**TSDF Integration**
	Execution Time (s)	Nodes	Edges	Execution Time (s)	Nodes	Edges	Execution Time (s)
0	854.854	100	289	0.101	100	108	58.682
1	929.902	100	289	0.115	100	106	38.608
2	905.416	100	289	0.150	100	120	43.948
3	946.328	100	289	0.199	100	115	23.297
4	887.127	100	289	0.211	100	139	45.845
5	877.984	100	289	0.142	100	110	50.379
6	818.913	100	289	0.145	100	129	47.284
7	787.866	100	282	0.263	100	109	50.794
8	377.204	100	167	0.108	100	138	28.217
9	025.732	7	7	0.003	7	7	10.223
Average per fragment (except last)	820.622	100	275	0.159	100	119	43.006



(a) (b)

Figure A.1: Before (a) and after (b) merging RGB-D generated point cloud with SfM generated model, from different viewpoints.