FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Shape Completion with a 3D Convolutional Neural Network for multi-domain O&M activities in offshore wind farms.

Ricardo Fernando Freitas Dinis

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master's Degree In Electrical And Computer Engineering

Supervisor: Andry Maykol Gomes Pinto

Co-Supervisor: Daniel Filipe Barros Campos

July 29, 2020

Resumo

A indústria de energia eólica offshore está a ganhar cada vez mais força no mercado da energia, trazendo novos desafios. Um destes desafios é a tarefa Operação e Manutenção (O&M), mais especificamente a inspecção onde as metodologias actuais são altamente dependentes do homem, perigosas e dispendiosas. Como tal, o desenvolvimento de sistemas autónomos procura a automatização destas tarefas para aumentar a sua precisão e segurança. e, mais especificamente, a inspecção.

Este trabalho centrou-se na utilização de um ASV equipado com sensores que lhe permitam fazer o levantamento do ambiente e criar uma representação do ambiente circundante considerando tanto os domínios aéreos como subaquáticos. Isto permitirá aumentar a consciência dos veículos nas operações marítimas, permitindo assim uma reconstrução mais rica e ainda mais segura para aplicações como a navegação a curta distância. Contudo, devido às limitações dos sensores no campo de visão, nem todos os dados podem ser observados principalmente na região da fronteira de superfície, criando representações incompletas. Além disso, diferentes tipos de estruturas podem ser encontrados no ambiente marinho com diferentes formas que podem pôr em risco a missão.

Com isto em mente, o objectivo deste trabalho é aproveitar os dados e conhecimentos disponíveis sobre as estruturas circundantes (turbinas eólicas) para inferir a informação em falta. A abordagem é filtrar os dados dos sensores, classificar a estrutura usando uma 3D Convolutional Network (CNN) e alinhar um modelo dessa estrutura com a representação incompleta. Este trabalho é implementado e testado num ambiente simulado, onde foram desenvolvidos modelos CAD para 12 estruturas 3D comuns.

Com esta abordagem foi possível classificar com uma certeza de mais de 86% utilizando um desvio padrão de ruído de 0,7. Quanto ao registo, foi obtido um alinhamento visualmente preciso entre o modelo e os dados do sensor.

Abstract

The offshore wind power industry is gaining increasing traction in the energy market bringing new challenges. One of these challenges is the Operation and Maintenance (O&M) task, more specifically inspection where the current methodologies are highly human dependent, hazardous and costly. As such, the development of autonomous systems seeks the automation of these tasks to increase their accuracy and safety. and, more specifically, inspection.

This work focused on the use of an ASV equipped with sensors that allow it to survey the environment and create a representation of the surrounding environment considering both aerial and underwater domains. This will allow to increase the awareness of the vehicles in maritime operations thus allowing a richer and even safer close-range navigation for applications such as environment reconstruction. However, due to the sensor limitations in the field of view, not all data can be observed mainly in the surface frontier region, creating incomplete representations. Moreover, different types of structures can be found in the sea environment with different shapes that may put the mission at risk.

With this in mind, the goal of this work is to leverage the available sensor data and knowledge about the surrounding structures (wind turbines) to infer the missing information. The approach is to filter the sensor data, classify the structure using a 3D Convolutional Neural Network (CNN) and align a template of that structure on the incomplete representation. This work is implemented and tested in a simulated environment, where CAD models for 12 common 3D structures were developed.

With this approach it was possible to classify with a certainty of more than 86% using a noise standard deviation of 0.7. As for the registration it was obtained a visually accurate alignment between the model and the sensor data.

Acknowledgements

This thesis was supported by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-030010.

Contents

List of Figures

List of Tables

Acronyms and Symbols

1D	1 Dimensional
2D	2 Dimensional
3D	3 Dimensional
ASV	Autonomous Surface Vehicle
CAD	Computer-Aided Design
CDBN	Convolutional Deep Belief Network
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
ICP	Iterative Closest Point
MSE	Mean Squared Error
NN	Neural Network
O&M	Operations and Maintenance
PSO	Particle Swarm Optimization
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue-Depth
RNN	Recurrent Neural Network
ROV	Remotely Operated Vehicle
STN	Spatial Transformer Network
TJ30	Twisted Jacket with a 30° twist

Chapter 1

Introduction

1.1 Context

Currently, energy production is moving away from hydrocarbon-based sources and toward renewable sources, due to environmental and financial motives. As a result, in 2017, almost 20% of the global energy production came from renewable sources and it is expected to reach 34% by 2050 [1], driven mainly by solar and wind energy.

An increasing interest is being placed on wind energy, specifically in offshore wind farms [2]. Their main advantages are that wind speeds tend to be higher and steadier than when compared to onshore facilities, thus creating more energy and being more reliable. These wind farms are built within some distance from the coast where the water depth can vary from 50 to more than 100 meters.

The depth factor presents a challenge since it is very hard to build robust structures that can extend to such depth and, as such, a variety of wind turbine designs had to be developed, including floating ones, like in Figure 1.1.

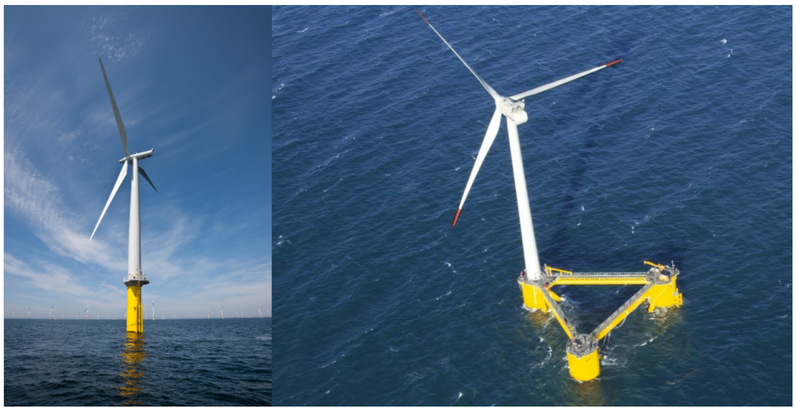


Figure 1.1: Picture of Monopile [3] and WindFloat [4] turbines.

The offshore wind farm environment is harsh and with several challenges and limitation for the infrastructure efficiency and reliability, such as:

- The wave action and storms can cause significant structural damage;
- The accessibility limitations for this environment makes it difficult to perform maintenance procedures;
- Regular inspection is highly dependent on weather conditions;
- The water turbidity limits the visibility for underwater inspection.

Thus, inspection is an important task in operation and maintenance (O&M) procedures on these wind farms to ensure the correct functioning and for the early detection of damage to the structures. One of the most commonly occurring problems is the underwater electrical infrastructure getting damaged, demanding regular surveys to ensure the cables conditions. As such, since the cables are placed at the sea bottom, the current inspection solutions are mostly manual, namely with divers and Remotely Operated Vehicles(ROV) equipped with sensors such as profiling sonars and cameras, which are hard to operate in an underwater environment - water turbidity lowers the camera range and floating sediments cause noise for sonars.

With the current developments of an Autonomous Surface Vehicle (ASV) equipped with a lidar sensor, stereo cameras, and a multibeam sonar to survey the environment below and above water [5] the survey tasks can be automated. This allows creating a 3D map of the multiple layers (emersed and immersed regions) of the offshore scenarios through data fusion methodologies [6]. Nevertheless, this results in missing information around the surface border (see Figure 1.2), i.e. the reconstructed model suffers from incompleteness due to field of view limitations on the ASV sensor payload.

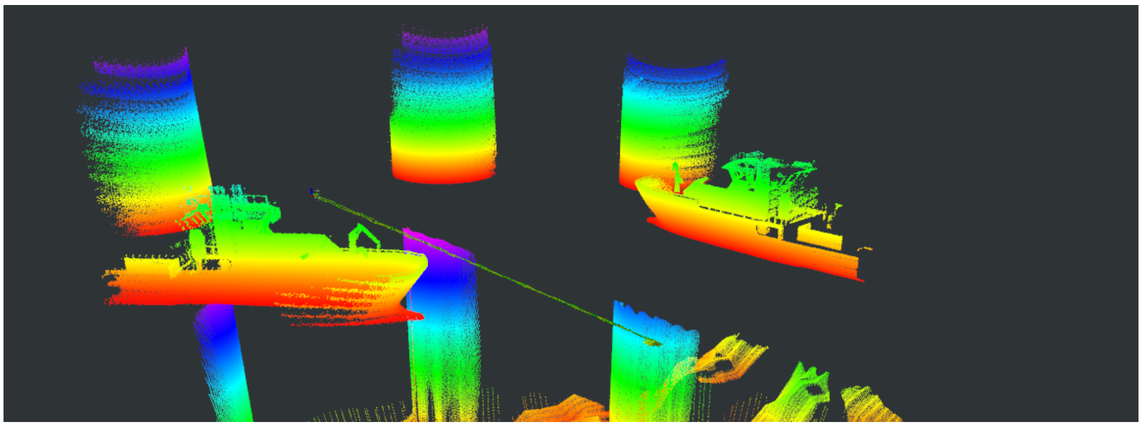


Figure 1.2: Image of scanned 3D map after performing Data Fusion. Here we can observe two vessels and the turbines tower with the missing region around surface border.

While the missing zones are not critical information for the inspection procedure itself, it is important knowledge to avoid collisions with nearby structures [7] and help with close-range navigation [8]. As such, in this work, a solution to complete the various scanned models will be studied to aid in the navigation system to avoid collisions.

1.2 Motivation

Current inspection methods are heavily man dependent, long, challenging, dangerous and highly conditioned by environmental conditions, mainly in Winter time, imposing a small operational temporal window to work. They are usually performed by rope access technicians, and scuba divers often during restrict weather windows and in extreme conditions [9]. In other cases, they resort to ROVs equipped with RGB or stereo cameras, for instance MARESye [10], that can only observe objects at very short distances. These prove to be quite tedious and lengthy tasks - one farm alone can contain hundreds of turbines [11] - while human error can cause severe consequences, from a safety and financial standpoint.

Therefore, the application of autonomous vehicles would be beneficial for the inspection of the wind farms since their introduction will:

- Reduce the risk of injury or loss of human life. Over 2018, G+ Global Offshore Wind Health and Safety Organization registered 117 work injuries, over 854 incidents [12] where most of them occurred on the maritime environment
- Increase of the lifetime of wind power plant, due to the possibility of performing structural inspections more regularly
- Add a more systematic approach to inspection can contribute to improving the overall farm profitability, reliability and sustainability

Nevertheless the vehicle, as depicted in Figure 1.2, presents incomplete information due to the field-of-view limitations of the provided sensors. As such, the completion of the shapes presented in scenario will provide richer information for collision avoidance, close-range navigation and inspection tasks.

Given that the offshore wind farms are composed by man made and know infrastructures it becomes feasible to augment the representation to reduce the observability limitations. To do this we need to: isolate the structure, recognize its type, and then use that information to fill the gaps in the structure.

1.3 Objectives

The main goal of this work is to implement a procedure to complete the scanned 3D environment to aid the robot's navigation system.

- Develop a classification methodology to identify distinct offshore wind farms structures using 3D information;
- Develop a shape completion methodology capable of tackling data imperfection, namely vagueness, ambiguity, and incompleteness;
- Define a modular multilayer architecture to classify and complete known offshore structures using 3D data.

1.4 Document Structure

This document is organized in 6 chapters. Chapter 2 depicts some of the different methods used in the fields of shape completion, 3D classification, and registration. Chapter 3, presents a description of the problem and the proposed method. Chapter 4 goes over the various implementation steps and gives a more detailed description of the work. Chapter 5 showcases the final results and validates the implementation. Finally, Chapter 6 gives a brief conclusion as well as presents future work.

Chapter 2

Literature Review

2.1 Shape Completion

3D shape reconstruction has been one of the most important aspects of computer vision. It gained a lot of momentum since the introduction of commodity cheap range sensors, such as the Microsoft Kinect. Though 3D reconstruction from this data can yield impressive results, they often result in incomplete models. This can be due to occlusion, data fusion issues, or sensor limitations. Even though in some cases, it is possible to solve this by taking more scans from different positions or using different sensors, in other cases, these solutions are non-viable or impractical so other software procedures are required.

Shape Completion algorithms aim to solve that situation by filling in the missing parts from partially reconstructed or corrupt models. These can range from small holes in a texture to missing high-level structures.

Despite the fact that humans are capable of comprehending and describing objects from incomplete models and 2D projections, it remains a challenge to emulate this ability on a computer. As such, it is common to observe some methods that require user interaction [13] instead of being fully automated.

2.1.1 Patching Methods

One of the most popular applications is to fill in small holes in CAD models, which is usually achieved by fitting new meshes according to local primitives, such as planes with methods like Laplacian Smoothing [14] and Poisson Surface Reconstruction [15]. Poisson Surface Reconstruction works by transforming the sampled points into a continuous 3D vector field and searching for the scalar function that best fits that field. The function is then used to estimate an isosurface that will cover the hole.

Though these methods can create good solutions for small holes, they are impractical for filling in higher-level structures, such as a table's missing leg or a plane's missing wing. Nonetheless, they are still useful to compliment others, for example, as a way to smooth or patch texture transitions between an incomplete model and a template.

2.1.2 Symmetric Reconstruction

Another subset of methods consists of detecting structural characteristics on the incomplete 3D shapes, for example symmetries, and using them to predict the missing data [16, 17, 18], as demonstrated in Figure 2.1. Symmetry detection on incomplete models is in itself a challenging task that created its own bulk of research [19]. Even though this technique can achieve good results, it places obvious constraints on the objects' shape and what can be reconstructed.

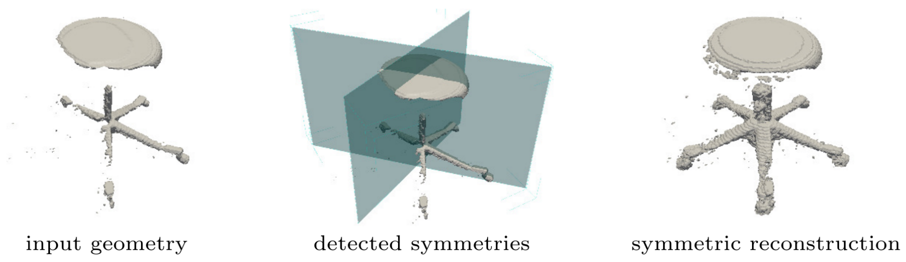


Figure 2.1: Symmetric Reconstruction example [16].

2.1.3 Database Priors

The idea of Database Priors methods is to find an identical CAD model and use it to cover the holes on the scanned model. Generally, this process has three main components: 1) Classify the input model to determine the respective CAD model in the database; 2) Align the 2 models within the 3D space so they overlap in the common features; 3) Blending the models to complete the input shape. It is also common to observe an additional post-processing step to obtain a smoother, more visually appealing result.

By leveraging templates, we can infer higher-level features and details that more generic mathematical models, such as Poisson Surface Reconstruction, have difficulty on, as can be observed in Figure 2.2.

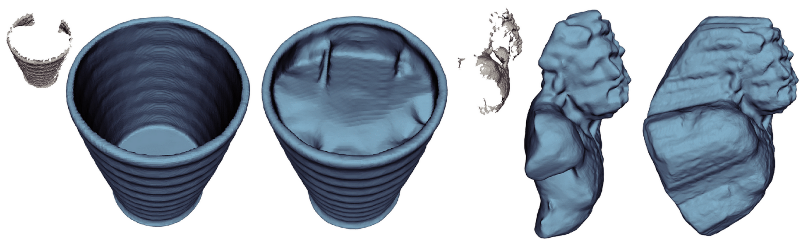
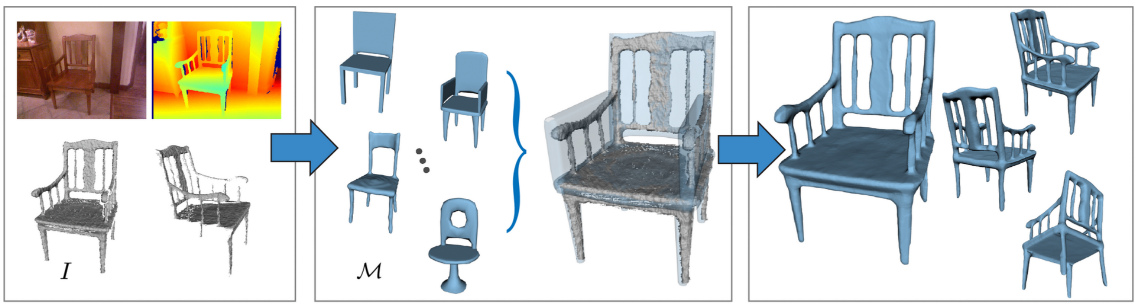


Figure 2.2: Results from [20] by matching templates (left) and by Poisson Surface Reconstruction.

The Database Prior approach relies on the assumption that the databases include identical objects to those being reconstructed. To work around this, some authors resort to performing deformation on the retrieved model before alignment. This way there is no need for an exact match and it is possible to cover a wider range of objects with a smaller database.

Pauly et al. (2005) [21] was one of the first to introduce this concept. They search for the correct model in the database with a combination of descriptive keywords provided by the user and shape similarity measurements based on point-wise squared distances. They then perform a deformation to the retrieved model to obtain better alignment find the optimal position by minimising a cost function. This function takes into account a distortion metric (the reasoning is that if only a small deformation is necessary, the model is more likely to provide a meaningful continuation in regions of missing data) and a shape matching penalty function, which measures the deviation between the input and the deformed model. Finally, they perform segmentation to decide which model should cover which zones and blend the models.

Li et al. (2017) [20] take a different approach to non-rigid alignment. They start by upright aligning both the template model and the incomplete model (to facilitate the initial alignment) and then perform rigid alignment by translating and scaling the template to fit the bounding box of the scan and exhaustively search for the orientation that minimises the corresponding points' distance. To obtain a better match, they perform embedded deformation [22] on the template model. They opted for a deformation energy function that includes matching error, shape smoothness and rigidity cost. They repeat this procedure for every selected model and keep the one with the best score. The result of this process is usually rough due to inevitable dissimilarities between the input and the deformed model, so they added an optimisation step where they patch the model to obtain a smoother final model.



(a) input object I (b) initial solution from the model collection $\mathcal{M}$ (c) our patch-based optimization result

Figure 2.3: Li's method pipeline [20].

Li et al. (2015) [23] propose the replacement of reconstructed, incomplete, models with hand-designed 3D ones, which allows them to skip the blending component. They employ a corner detection algorithm (by sampling non-planar points) from where they determine the key points from the object shape and additional descriptors (global primitives and local neighbourhoods). To match the models, they match the key points from both models by their descriptors and then compute a translation that minimizes the distance between them. They show good results for identical database and scanned models, but sometimes struggle to find the correct alignment on more cluttered scenes and when the models are not so identical with the ones in their database. Since they did not implement posegraph optimization, the authors suggest that as a possible avenue to improve the method. An example can be seen in Figure 2.4

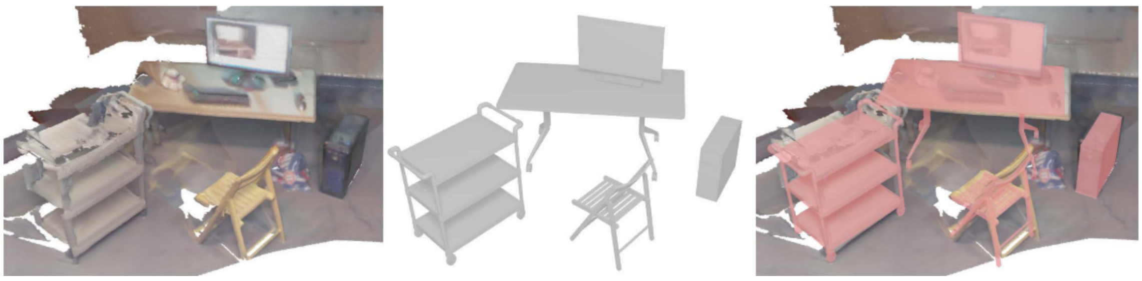


Figure 2.4: Example result from [23]: initial reconstructed scene (left), reconstructed scene after matching (middle) and superposition of both (right).

Gupta et al. (2015) [24] also proposes the reconstruction of a 3D scene purely from templates but they do not start from partially reconstructed model. Instead, their goal is to predict depth data from occluded parts of the objects, though the techniques used are still of relevance to this subset of methods. Their algorithm inputs an RGB-D image where they perform segmentation and detection of the desired objects. They propagate the segmented image through a CNN which outputs a coarse pose estimation which serves as initialization to the next step (note: this last procedure does not provide information about the model that would best fit the image). In this stage they select a couple of models and rotate them (with the orientation details yielded by the CNN), then they perform alignment optimization through the ICP algorithm. Since they are trying to match it with a 2D image, at each iteration, they project the 3D model on the 2D space and compute the respective depth values. Only then do they run a linear classifier based on a specific subset of features to select a 3D model they will best fit to the image.

Liu et al. (2019) [25] extract key points from the 3D model to perform identification and pose estimation. They use an unsupervised learning algorithm where they propagate the 3D model through a 1D Encoder CNN network (obtaining a feature vector) plus a decoder with reversed dimensions (to reconstruct the model). By reconstructing the model from their feature vector they can compare it with the original one to compute a loss function to backpropagate. The feature information allows them to search a database of models they already encoded and to obtain an accurate pose estimation in the 6 Degrees of Freedom and then patch the incomplete model.

While the idea of non-rigidly deforming models from a database improves shape coverage, these methods are still limited by the fact that global structure cannot be easily generalized, as such, a database with somewhat similar models will always be a critical requirement.

2.1.4 Deep Learning Approaches

The advances in Machine Learning and the availability of 3D shape databases inspired new approaches that, besides classification, use machine learning for generative tasks [26, 27, 28].

Wu et al. (2015) [29] was among the first to apply this concept. Their method consisted of creating a volumetric representation from a single depth map. They identify each voxel as object surface; free space; occluded, unknown space. The volumetric representation is the input to a Convolutional Deep Belief Network (CDBN) which recognizes the object category (chair, table, etc.) and which voxels are part of its surface. Additionally, it can suggest a new angle to capture a new image from, if it is unsure of the prediction. The goal is to classify the "unknown" voxels as either free space or part of the mesh, so the CDBN outputs, for each voxel, a probability value about whether it belongs to the surface or not. Figure 2.5 illustrates the method steps.

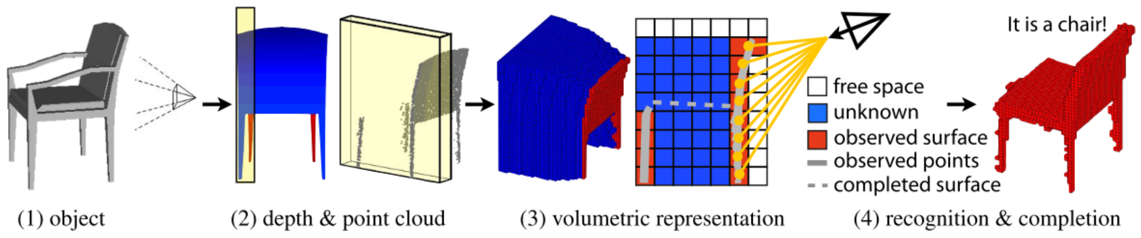


Figure 2.5: Illustration of the method in [29]

Dai et al. (2017) [30] introduces a different structure to solve this problem, depicted in Figure 2.6. They create an independent CNN which classifies the incomplete shape and a 3D encoder which creates a compressed stream of values. The results are concatenated and fed into a sequence of 3D up-convolutions to generate an intermediary prediction. They also introduce a post-processing step by leveraging a database prior. They learn a shape feature descriptor based on a 3D-CNN to retrieve the most similar CAD model. With the reconstructed shape and the higher resolution CAD model, they can perform a shape synthesis process to increase the resolution and local geometric detail of their prediction. At the time of publishing, this method outperformed every other state-of-the-art method it tested against (this includes Wu's method described above).

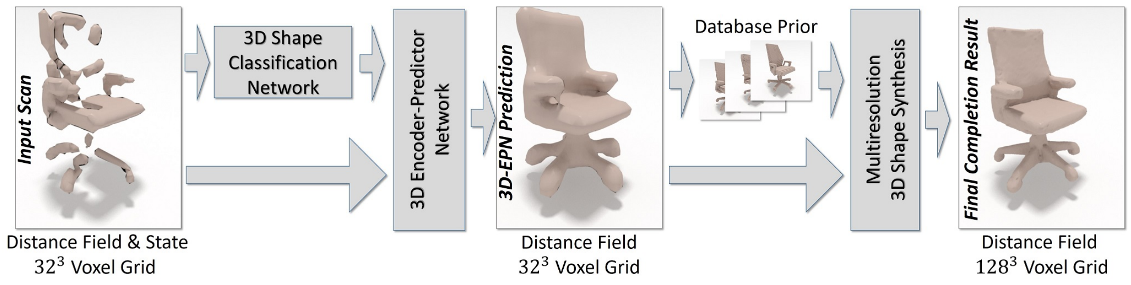


Figure 2.6: Method architecture from [30]

Wang et al. (2017) [31] They propose the implementation of a more end-to-end approach by applying the concept of Generative Adversarial Network (GAN) to complete corrupted 3D scans, followed by a Recurrent Neural Network (RNN) to increase the model resolution. GANs consist of 2 neural networks, a generator (which tries to generate a "real" sample) and a discriminator (which classifies the samples as real or fake). The generator follows a 3D Encoder-Decoder architecture while the discriminator uses the same encoder architecture with a fully connected layer at the end. For the RNN, since 3D CNNs require a lot of memory, they opted instead to use 2D convolutions and treat the 3D model as multiple slices of 2D images, for this, they claim to be more memory efficient than other 3D CNN based methods, which could make this method more suitable to map complex high-resolution indoor environments. Their testing shows that this network can correctly infer high-level structures without a classification step and also its potential use as a feature learner. Though they make reference to Dai's method above as the state-of-the-art method, they did not compare with their results.

These approaches are generally more versatile than database dependent methods as they can infer high-level structural details without directly copying them from a prebuilt CAD model. This makes them ideal for objects with many variants or with highly deformable features [32]. Additionally, as some methods don not require model selection *a priori*, they can skip a problematic step of classification on incomplete models or multiple costly model alignment procedures. On the downside, training a neural network for generative purposes still proves to be a challenging task and the greater generalization possibilities is not necessarily an advantage when trying to complete models from objects with limited, known variations where using a template matching approach can grant some robustness to the method.

2.2 Classification on 3D Data

Techniques for analysing 3D shapes are becoming increasingly important due to the number of sensors capable of capturing such data, and in recent years, a wide variety of network architectures have been developed to achieve different goals. However, the number of available representations of this data, create some fragmentation in this area and a lot of work has evolved in parallel.

2.2.1 Volumetric Representations

The Volumetric Representations divide the space into cells representing an uniform sub-volume. These representations are directly affected by the model resolution, as can be observed in Figure 2.7, obtained from [33].

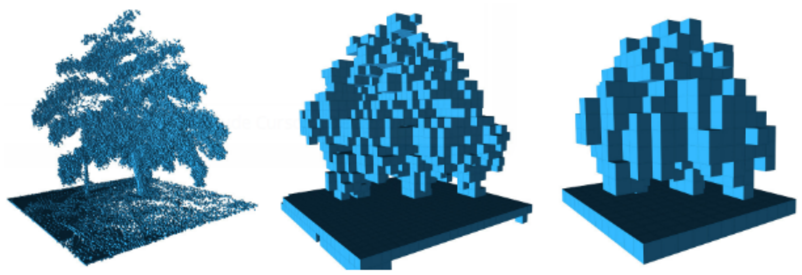


Figure 2.7: 3D map with different resolutions: 0.08m, 0.64m and 1.28m [33].

Voxel Grid

Voxel grids characterize 3D data as a regular grid in the 3D space (akin to how pixels represent a 2D image). In this representation, each voxel is classified as occupied and non-occupied and takes on a cubic shape. This makes it so the data on a specific region can be easily accessed, viewed and manipulated. The implementation is also simple as it can be implemented with just a 3D array.

There are also alternative implementations for voxel grids. For example, it is possible to model uncertainty in a voxel by assigning it the probability that it is occupied. Another method is to use Signed Distance Functions where each voxel can be assigned 0 (surface of an object), a positive value (known empty space), or a negative value (unknown).

Despite its simplicity, this representation is restrictive since it encodes all the space (occupied and unoccupied, occluded and visible) in a 3D map of equally sized sections. This creates a high memory storage requirement ($O(n^3)$), and makes their processing more demanding, especially for higher resolution models. To combat this, most voxel-based applications tend to use lower resolution representations, for example, 30x30x30.

The first Neural Network to process this type of data was 3D ShapeNet [29]. It exploited the 3D convolution, which works similarly to a 2D convolution with an added dimension. In the same publication, the authors also introduced the ModelNet dataset, which went on to become one of the most popular public benchmarks for 3D classification tasks.

Maturana et al. (2015) [34] introduced VoxNet, which managed to outperform 3D ShapeNet on the ModelNet dataset while requiring significantly fewer parameters (12 million to less than 1 million)

Other works have also explored deeper 3D CNNs, such as, VoxCeption Resnet [35], which was inspired by Inception architectures. This model contains 45 layers and, as such, requires a significant amount of data augmentation to avoid overfitting and computation to run.

Octree

An Octree is a tree-type data structure where each node has 8 or 0 (leaf node) children. One way this can encode a 3D model is by assigning the whole space to a node and subdividing it into 8 cubic regions between its child nodes. Each of these nodes is then classified as "empty", "full" or "partially filled" depending on their space's contents. When a node is classified as partially filled, it is further subdivided into new octants and spawns 8 new child nodes. As the empty and filled regions are already homogeneous, there is no need to detail them further so their nodes turn into leaf nodes. This process is run recursively until every node is classified as either "empty" or "full", as depicted in Figure 2.8.

Being able to represent adaptable size, uniform regions of the 3D space as one entity is the main reason why Octrees are much more memory efficient than voxel grids: a lot of the branches will not be explored to its full depth.

The most straightforward way to implement an octree in each node, the pointers to its child nodes, as in Figure 2.9. Though other implementations are also used which represent different memory and speed trade-offs [33]. This representation can significantly speed up 3D CNNs since all values within the same cell are known to be equal, a big part of the computations can be skipped, as shown in [37]. While this method can speed up processing and lower the memory requirements

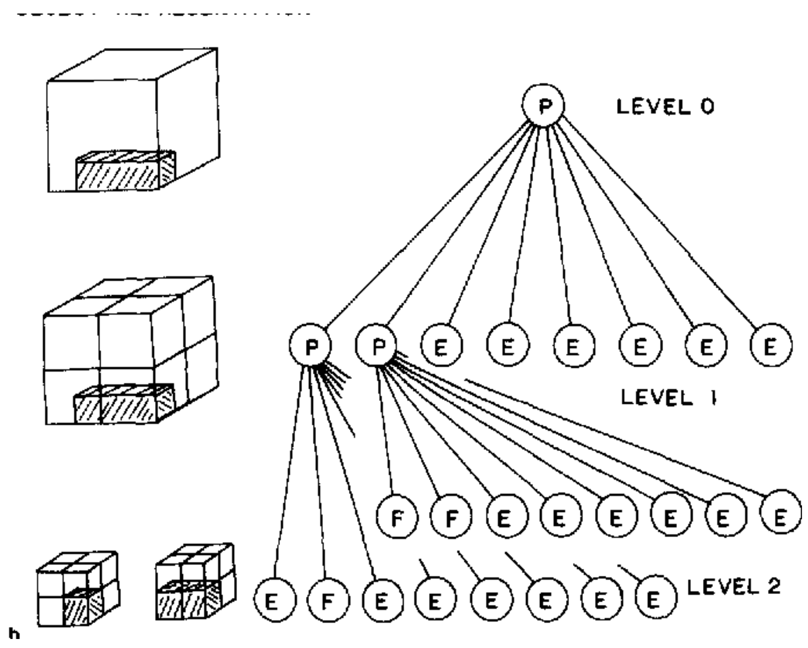


Figure 2.8: Octree encoding format [36].

```
struct OcTree{
float center[3];
float size[3];
OcTree *child[8];
int nr_points;
float **points;
};
```

Figure 2.9: Example implementation of an Octree node.

of a volumetric representation, the effects are less noticeable for lower resolution models, which have been the most commonly used for 3D CNNs.

2.2.2 Point Cloud

A point cloud is an unstructured group of 3D points that approximate the geometry of an object. Capturing models in this format is a relatively simple and cheap process, however, processing can be a challenging task. Since the points are stored in no particular order and there is no information about the connectivity between them, a certain degree of ambiguity about the object will be present.

The lack of order between points also creates a challenge for neural networks, as a change of order between 2 points will result in the same 3D structure, but different results on the first NN layer, therefore most works resorted to performing voxelization or other techniques to transfer point clouds to a different type of data before processing it on a neural network.

Qi et al. (2017) [38] pioneered deep learning on raw point clouds by introducing PointNet. It is composed by a Spatial Transformer Network (STN), an RNN and a symmetric function. The STN, based on 1D convolutions, processes all the data into one canonical form and learns the key points of the point cloud which approximately corresponds to the skeleton of the 3D object. Then, the RNN learns the point cloud like a sequential signal of points and by training the model with randomly permuted sequences, it becomes invariant to the input order of the point cloud. Lastly, the symmetric function is a max-pooling operation. This results in a feature vector, which is then sent to a different NN which can perform classification, segmentation, etc...

Projection

3D Data Projections can represent 3D data in a 2D structure. The type of stored features is dependent on the projection method. Spherical projections (Figure 2.10), for example, represents the 3D data on a 2D plane while being invariant to rotations around one axis. Naturally, such a conversion will cause some degree of information loss that can not be directly reconstructed and must be taken into account by the application. One major advantage is that, since the resulting structure is planar, it is possible to use the massive and widely available image data-sets for pre-training a classifier neural network [39] to then obtain better results for the projected 3D models.

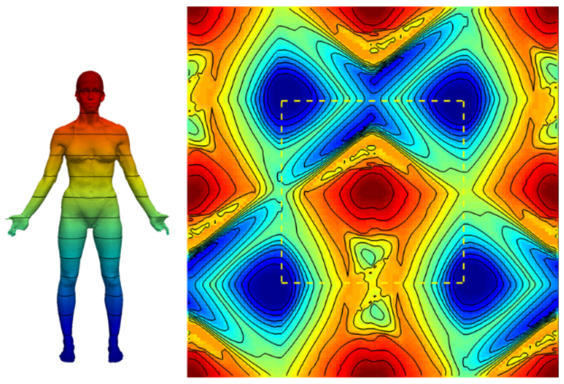


Figure 2.10: Result of a spherical projection of a 3D model [40].

2.2.3 RGB-D

RGB-D data is usually considered a 2.5D data type. It consists of an RGB image paired with a depth map (D). The normalization of cheap sensors like the Microsoft Kinect and its simplicity made this data type very popular for applications like scene reconstruction and pose regression.

Typical neural networks for this type of data use separate 2D CNNs for the RGB and the Depth layers before fusing their features and making a final decision.

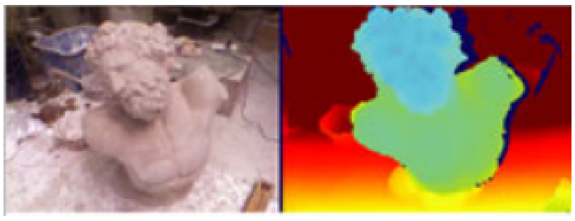


Figure 2.11: RGB-D example [20].

2.3 Registration

Another key problem is: given two sets of 3D data points, find the optimal rotation and translation that minimises the distance between them. Some methods have been proposed to solve this problem, such as: Iterative Closest Point (ICP) [41] and Chen Method.

2.3.1 Iterative Closest Point

Given a set of points from two images, m_i and p_i, in a common reference system, the ICP method searches for a geometrical transformation (described by a rotation R and a translation t), such that the mean square distance between the points in p_i and their closest points in m_i is minimised. Those parameters are found with iterative calculations until equation 2.1 converges.

$$f = \frac{1}{N_p} \sum_{i=1}^{N_p} \left\| \vec{m}_i - R\vec{p}_i - \vec{t} \, \right\|^2 \tag{2.1}$$

This method can obtain good results in the presence of Gaussian noise. Its main drawbacks are that it can converge to a local minimum and the quadratic time complexity $O(N^2)$ for N points.

Over the years researchers proposed modifications to the ICP algorithm:

- *Greenspan et al.* (2001) [42] applied the Nearest Neighbour problem to ICP to increase its efficiency. They pre-process the reference image and apply a spherical constraint to find which points are within the neighbourhood of each reference point. The constraint imposes that, if a point is closer to the reference $\vec{q}$, than the current estimate $\vec{p}_i$, it must be located within a sphere centred in $\vec{p}_i$ with radius $2\|\vec{q} - \vec{p}_i\|$ This pre-treatment reduces the computation of ICP drastically. After that, a triangular constraint is applied and the points are ordered according to their distance to the reference point to identify the correspondence.

- The Multi-Resolution Scheme ICP Algorithm[43] proposes applying down-sampling to reduce the number of points and to progressively increase the number of points in later iterations whenever the distances fall below certain thresholds. The author claims this method can lead to an increase of over 1300 times the speed of a non-accelerated ICP algorithm when coupled with a faster search algorithm.

- *Sharp et al.* (2007) [44] proposed matching points using invariant features, such as curvature and spherical harmonics. They achieve this by adding a weighted feature distance, as in equation 2.2.

$$d(p,q) = d_e(p,q) + d_f(p,q) \tag{2.2}$$

Where d_e represents the euclidean distance and d_f represents the distance in the feature space. This change decreased the probability of the ICP algorithm getting stuck in a local minimum, thus increasing the convergence rate.

- *Trucco et al.* (1999) [45] made use of the Least Median of Squares approach to make the ICP algorithm more robust to outliers. At the cost of higher complexity, this method can tolerate higher amounts of missing data and wrong measurements than regular ICP does, while maintaining similar accuracy on situations with Gaussian noise only.

- *Shen et al.* (2007) [46] extend ICP by including parameters that enable model deformations. This makes increases the shape correspondence and improves its potential for shape analysis.

2.3.2 Hybrid ICP Methods

One of the biggest issues with ICP is that it lacks any way to figure out whether it has been stuck in a local minimum. Due to that, it is very reliant on the initialization, i.e., the initial position before executing ICP. However, this initialization is not always possible to find. This inspired other works that tackle this issue by searching for the optimal initialization and make the registration process more robust.

Yang et al. (2016) [47] introduced GO-ICP. This method uses branch-and-bound to search the whole 3D space, repeatedly executing ICP to test which initialization yield a better final alignment.

Yu et al. (2014) [48] implements a similar procedure using Particle Swarm Optimization (PSO). Having the particles travel in a multidimensional space representing an initial transform, this method allows multiple initializations to be tested in parallel.

These methods proved to be significantly more robust than regular ICP. The main trade-off is computational demand. Since they require executing ICP and transforms multiple times, the execution times can add up quickly.

2.3.3 Deep Learning Methods

Recent works have focused on applying deep learning methods to solve the registration problem. *Aoki et al.* (2019) [49] combines PointNet [38] with the LK algorithm. The architecture is shown in figure 2.12. It computes global feature vectors for the input point clouds and then, iteratively, computes incremental transforms to the source point cloud until they reach below a certain threshold. During training, the loss function is the difference between the estimated rigid transform and the ground truth transform.

Groß et al. (2019) [50] also proposed a NN based registration method, but with emphasis on time performance. They also utilize PointNet [38] to extract feature vectors from the source and

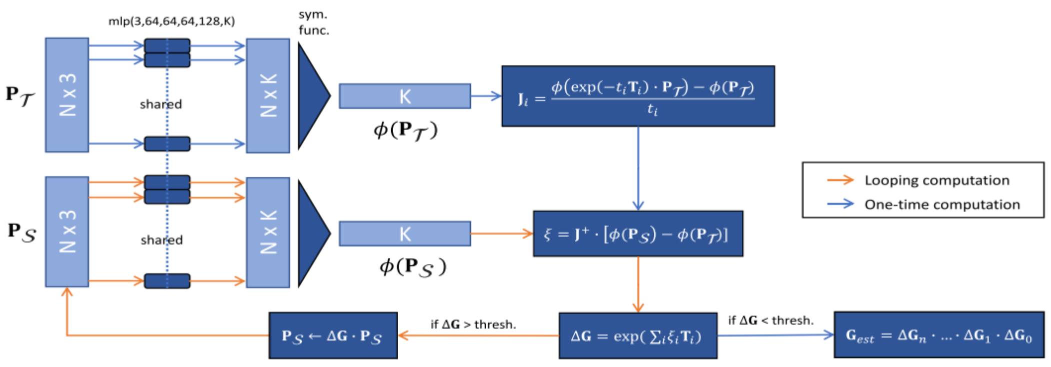

mlp(3,64,64,64,128,K)
sym. func.
$\mathbf{P}_{\mathcal{T}}$
N x 3
shared
N x K
K
$\phi(\mathbf{P}_{\mathcal{T}})$
$\mathbf{J}_i = \frac{\phi(\exp(-t_i \mathbf{T}_i) \cdot \mathbf{P}_{\mathcal{T}}) - \phi(\mathbf{P}_{\mathcal{T}})}{t_i}$
Looping computation
One-time computation
$\mathbf{P}_{\mathcal{S}}$
N x 3
shared
N x K
K
$\phi(\mathbf{P}_{\mathcal{S}})$
$\xi = \mathbf{J}^{+} \cdot [\phi(\mathbf{P}_{\mathcal{S}}) - \phi(\mathbf{P}_{\mathcal{T}})]$
$\mathbf{P}_{\mathcal{S}} \leftarrow \Delta\mathbf{G} \cdot \mathbf{P}_{\mathcal{S}}$
if $\Delta\mathbf{G}$ > thresh.
$\Delta\mathbf{G} = \exp(\sum_i \xi_i \mathbf{T}_i)$
if $\Delta\mathbf{G}$ < thresh.
$\mathbf{G}_{est} = \Delta\mathbf{G}_n \cdot \ldots \cdot \Delta\mathbf{G}_1 \cdot \Delta\mathbf{G}_0$

Figure 2.12: PointNetLTK [49] architecture. The PointNet-type Neural Networks receive as the input P_τ (template), and P_S (source) and outputs their global feature vector (ϕ). These are used to compute a Jacobian (J), a twist (Ξ) and an incremental transform (δG), which is then applied to P_S. This process is repeated until δG reaches below a threshold.

template point clouds, but, rather than using an iterative approach, the resulting vectors are fed into a multi-layer NN that computes the transformation matrix.

2.4 Multi-sensor Data Fusion

Multi-sensor data fusion is a technology that aims to combine data from multiple sources to form a unified picture. It is an important and problematic step for systems that can not directly fetch all the required data at once or from the same source. Figure 2.13 shows an overview of the possible types of data fusion issues.

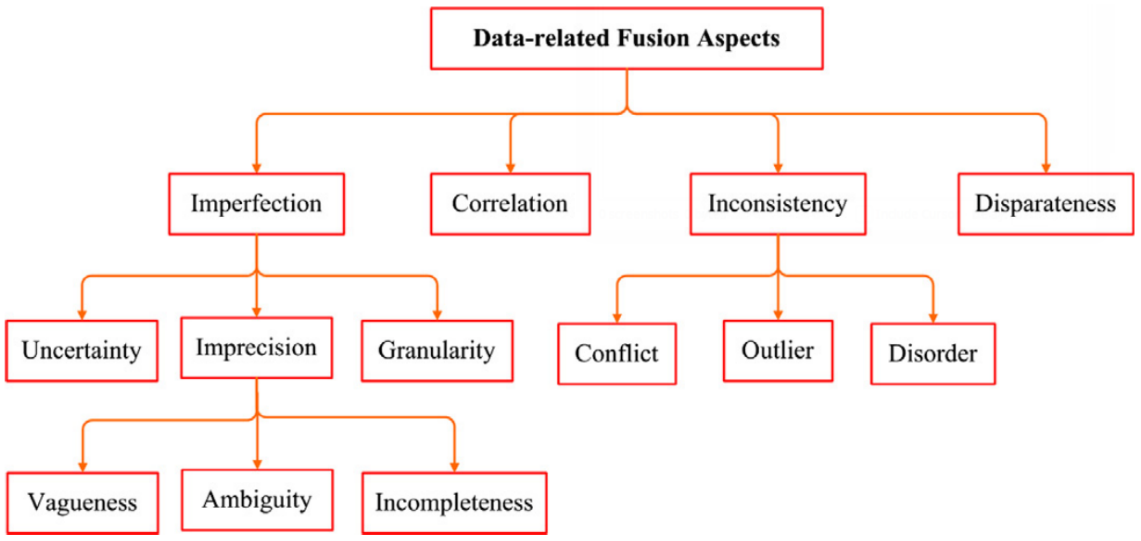


Figure 2.13: Taxonomy of data fusion methodologies [51].

One of the most pertinent issues to performing Data Fusion is the imperfection of data gathered by different sensors. The acquisition is always affected by some degree of uncertainty and imprecision which varies depending of many different factors. As such, data fusion algorithms must be able to express those factors.

The objective of this work can be categorized as solving an "Incompleteness" aspect of a Data Fusion application, as the combination of the sensors data contains missing information from the surface border region.

Chapter 3

Problem Formulation

3.1 Problem Description

Autonomous Surface Vehicle (ASV) needs to interpret the surrounding environment to correctly plan its routes and avoid collision. To that end, the ASV is equipped with sensors that yield information on nearby structures and terrain. For instance the INESC TEC Zarco ASV [5] perceives and navigates the scenario using several sensors, namely a L1+L2 RTK GPS (Swift Navigation Piksi Multi) and an IMU (Xsens MTi-30) for navigation, and a 3D lidar and a Multibeam Sonar for scene representation. The perception sensors in use are:

- The 3D lidar (Velodyne VLP-16) captures point clouds on a 360° horizontal and 30° vertical field of view, representing the environment above water. More detailed information in Table 3.1

- The Multibeam Sonar (Blueview M450-130) maps the underwater environment. It emits a series of beams across a 2-dimensional field of view to capture depth information at each angle. More detailed information in Table 3.1

Table 3.1: Sensor characteristics.

Attribute	Lidar	Sonar
Horizontal FoV(°)	360	130
Horizontal Resolution(°)	0.1	0.18
Vertical FoV(°)	30	-
Vertical Resolution(°)	2	-
Optimal Range(m)	100	150
Output Rate(Hz)	4	25

This sensors payload allows the reconstruction of the scenario as seen in the work of Campos et al. (2020) [6], however, it will not be sufficient to completely depict the scenario. In particular, the surface frontier region, depicted in Figure 3.1, is hard to capture, as the wave action can cause noise, occlusions and perspective shifts that can change the frontier region and the sensors in-vehicle positions and the vehicle non-holonomic motion (linear and angular control) does not provide full coverage of the environment. Since offshore wind farms provide a structured man-made environment it is possible to use the knowledge about the structures to complete the observed 3D information.

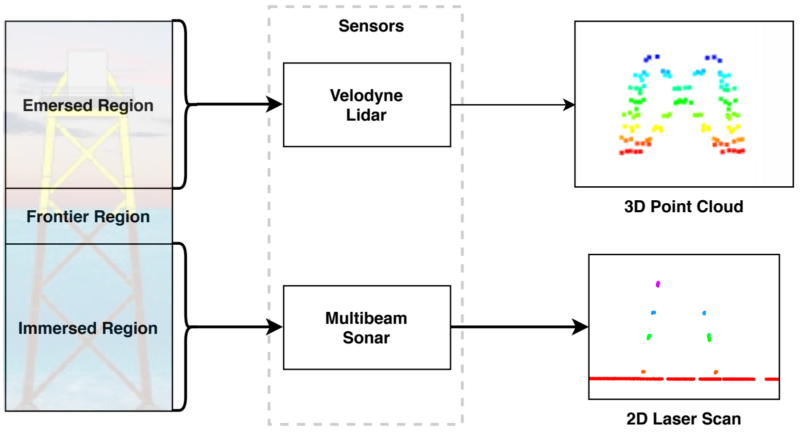


Figure 3.1: Sensor output data type.

3.2 Proposed Approach

The proposed approach falls under the Database Prior type, i.e. we rely on the assumption that pre-made models can represent the real world structures accurately enough (see Section 2.1.3). As is depicted in Figure 3.2 first, there is a preprocessing step, where the input data is transferred to the same modality (point cloud) and filtered. This filtering step is important to remove spurious and potentially misleading information from the sensor. Then the emersed region point cloud is converted to a voxel grid and ran through a classifier to infer the structure type. With this information, we can retrieve an appropriate template to align on the sensor data. For the registration step, data from both above water and underwater sensors are fused to get a better alignment. The registration step outputs a transformation matrix which is applied to the template point cloud to align it with the sensor data. Finally, a subtraction operation is implemented, to remove the template information where there is already sensor information.

The focus is on performing the completion on a single iteration, so no feedback mechanisms were added. This means that, when running the procedure repeatedly, it does not take previous results into account.

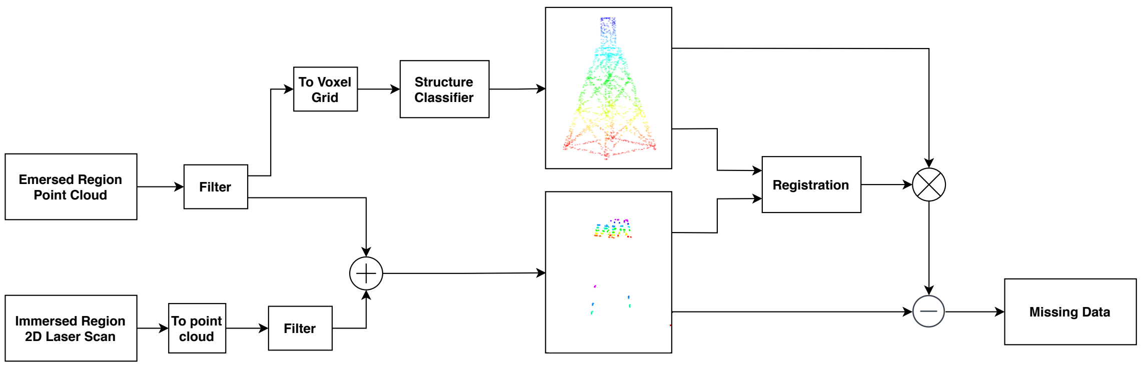


Figure 3.2: Method Overview.

3.3 Environment

This work was developed and tested on a 3D realistic simulated environment with marine scenario dynamics and sensor noise models. Moreover this scenario is composed by models of real offshore infrastructures (see Section 3.4) and of the Zarco ASV [5]. This work was developed using four main tools:

- ROS [54] provides an interface to the simulation tools, and programming and visualization utilities, including RVIZ which is used to showcase the final results;
- Gazebo is a simulator that includes a physics engine to simulate the environment and ASV. It also implements the used velodyne lidar;
- VRX [52] expands gazebo for the maritime environment. It adds wave models and a set of oceanic environments;
- UUV [53] is a set of packages to expand Gazebo for underwater scenarios. It implements the referenced sonar.

3.4 Turbine Models Design

While the Monopile type is by far the most common, other types are getting increased importance and the ASV needs to be prepared for such encounters. Thus, new 3D models of other turbines were developed in SOLIDWORKS, as shown in Figure 3.4(the rest of the models can be viewed in the Annex). Since there is no standard when it comes to dimensions or small variations, to get somewhat accurate measurements, the models were drawn based on research papers in their field, as depicted in Table 4.2.

Foundations for offshore wind turbines can be divided into two main categories: floating and fixed foundation. Among the fixed type, it was considered the Monopile and jacket foundations, the latter of which presents a range of varieties: four-legged, three-legged(tripod), and twisted.

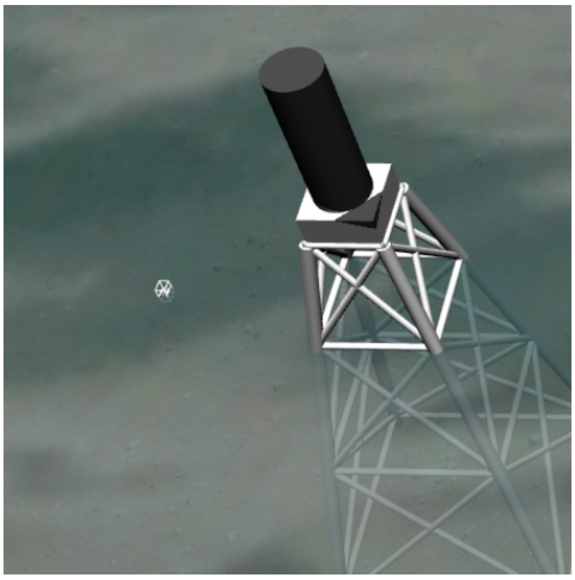


Figure 3.3: Simulator with setup world, including the ocean environment, the ASV(marked with the white box), and a turbine with a jacket foundation.

Table 3.2: Used turbine models.

	Type	Sub-types	Source
Fixed	Monopile	1	[55]
	Jacket	8	[56]
Floating	WindFloat	1	[57]
	DeepCWind Project	1	[58]
	Confloat	1	[59]

Most of the floating concepts are still in early development/prototyping stages, as such detailed descriptions or measurements are hard to come by. The WindFloat concept is perhaps the most mature of this type, being currently implemented on the Portuguese coast in Viana do Castelo by EDP in the WindFloat Atlantic project [60]. The DeepCWind project is not a type on its own, but it contains a generic foundation that is used by other types, such as Vertiwind and Compact Semi-Sub.

Due to the FoV limitations of the ASV sensors the aerial components such as the rotator blades and complete turbine were ignored. This reduces the computational demand on the simulation and focuses only on the perceived components.

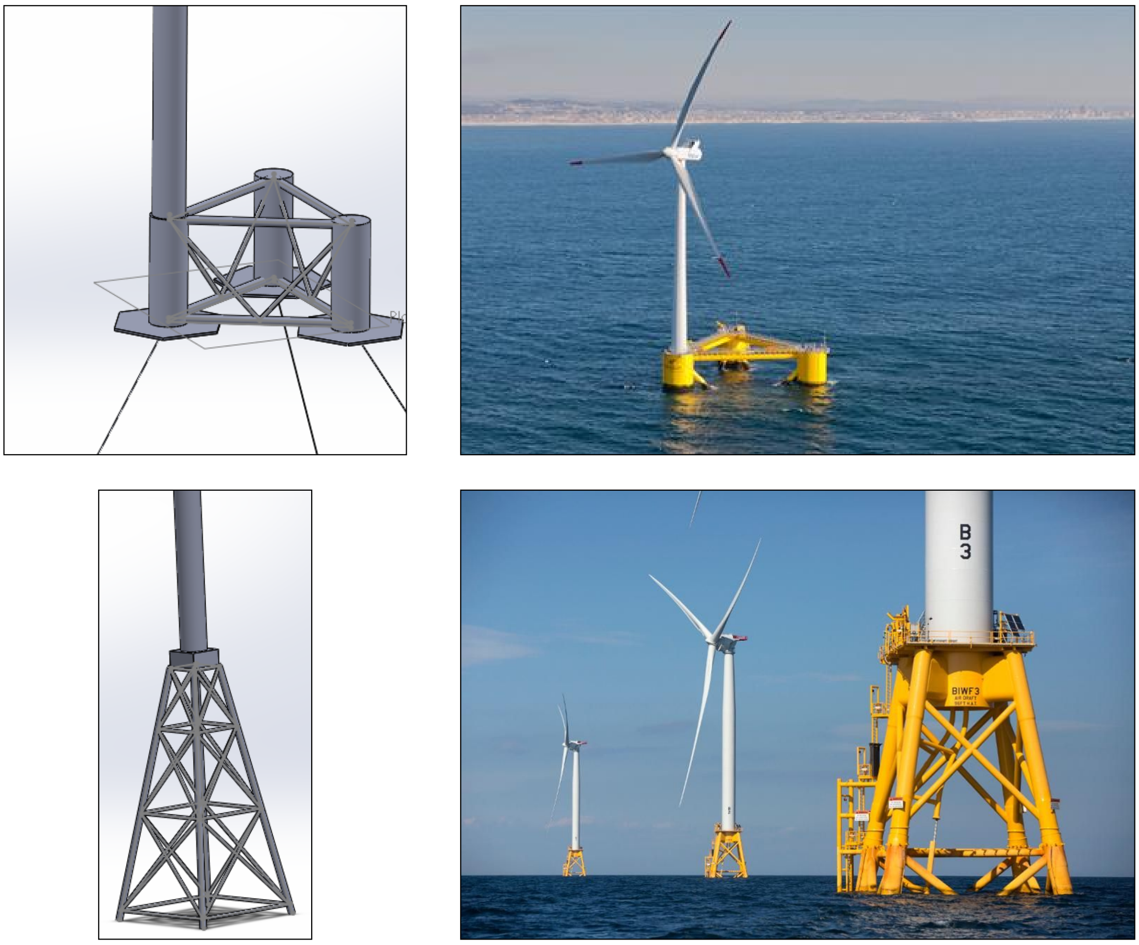


Figure 3.4: CAD model from the WindFloat (above) and Jacket type 'X'(below) structures.

Chapter 4

Methodology

4.1 Filtering

The sensors raw data suffers from noisy and spurious measures that can affect the classification accuracy. Moreover irrelevant measurements for offshore structures inspection are present such as the water surface on the Velodyne and the sea bottom on the multibeam sonar, as is depicted in Figure 4.1. Thus a passthrough filter was applied to remove every point under a certain height threshold. This data is filtered out by removing every point under a certain height threshold.

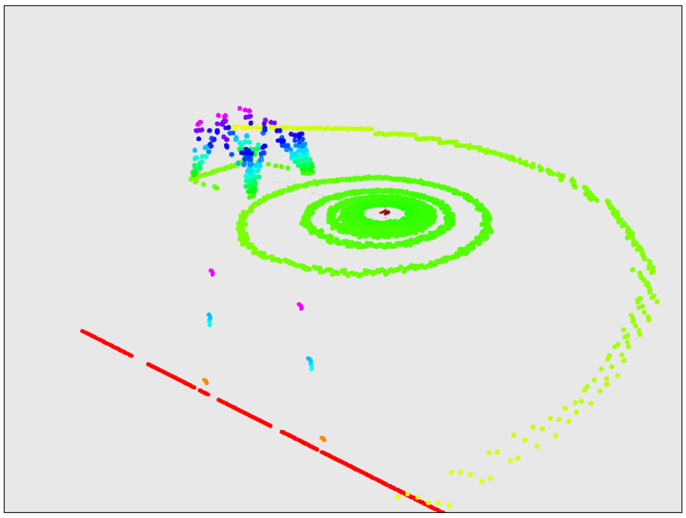


Figure 4.1: Raw sensor output visualization.

Furthermore several far away structures can be detected by the sensors making the classification task harder. Thus a later clustering step was implemented using the Euclidean clustering extraction algorithm proposed in [61] and depicted in Algorithm 1.

This method was preferred over K-means because it does not require setting a number of clusters *a priori* and, due to the FoV limitations and data sparseness, the number of nearby structures is unknown. Instead, it requires setting a "closeness" parameter, which defines how close a point must be from others to be considered a part of their cluster.

Algorithm 1 Euclidean Cluster Extraction Algorithm [61].

```
1: procedure EuclideanClustering(Point Cloud $P$)
2:     create a kd-tree representation for the input point cloud dataset $P$
3:     set up an empty list of clusters $C$, and a queue of the points that need to be checked $Q$
4:     for every point $\boldsymbol{p}_i \in P$ do
5:         add $\boldsymbol{p}_i$ to the current queue $Q$
6:         for every $\boldsymbol{p}_i^k \in Q$ do
7:             search for the set $P_i^k$ of point neighbours of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$
8:             add every not processed neighbour $p_i^k \in P_i^k$ to $Q$
9:         end for
10:        add $Q$ to the list of clusters $C$, and reset $Q$ to an empty list
11:    end for
12:    the algorithm terminates when all points $\boldsymbol{p}_i \in P$ have been processed and are now part of
       the list of point clusters
13: end procedure
```

The clustering process presents two main advantages, namely:

- allows additional filtering: by setting a lower limit on the number of points a cluster must have to be considered, we can filter undesired captured points, for example, caused by a seagull flying within sensor range or just an outlier point in the middle of nowhere
- detects nearby structures by assigning a minimum number of points and height requirements

4.2 Classification

This section describes how a shape classifier was trained to distinguish between the different structures, where the input is a point cloud obtained from a Velodyne sensor on a single capture, meaning that the sonar results were not considered for this module.

4.2.1 Data Generation

Ideally, the validation set should be acquired from real-life data, however during the development of this dissertation no wind farm dataset was available. As such, both the train and validation sets are captured using realistic environment hydrodynamics, vehicle kinematics and sensor models to mimic the real world from the gazebo simulator creating a synthetic dataset. The advantage of using a simulated environment is that it is possible to automatically generate large amounts of labelled data with a variety of different conditions. This is exploited by creating an automated procedure to capture samples from multiple viewpoints. To generate variability between the test and the train set, different viewpoints and noise configurations were used. The described scheme is depicted in Figure 4.2.

Using ROS API for gazebo, the ASV can be teleported to a chosen pose with a predefined attitude, from where the point cloud will be captured. To obtain the train data, the simulated ASV is teleported in circular trajectories around the structure (see Figure 4.3a). The positions represent

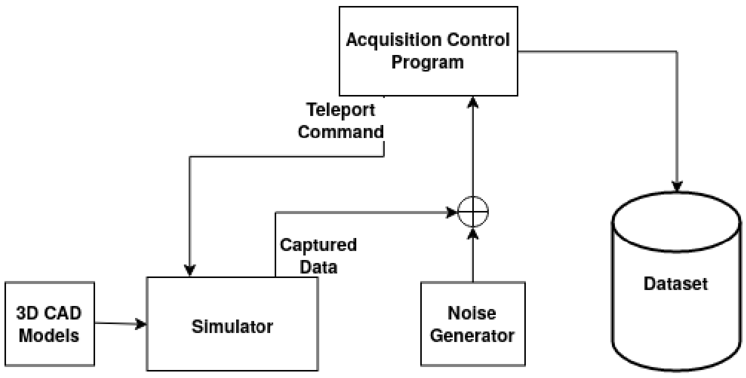
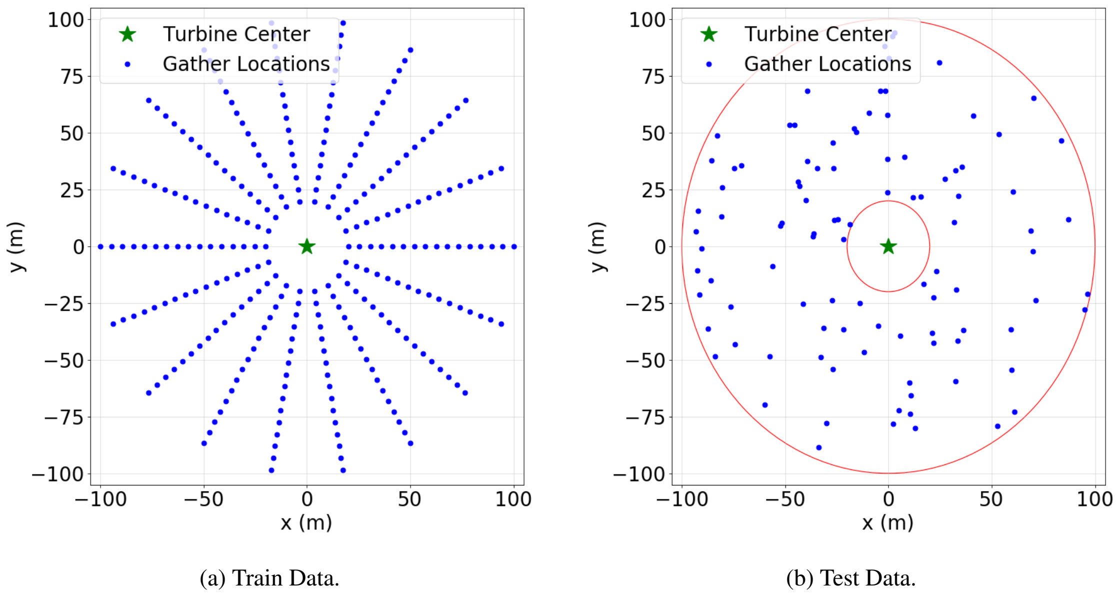


Figure 4.2: Data generation scheme. The acquisition control program and the noise generator are created for this application. The simulator is gazebo and the 3D CAD Models are the ones described in 3.4.

steps of 20° in the circumference and each circle has an increased radius of 5 meters. The test set positions, are generated randomly, following a uniform distribution for angle and distance (see Figure 4.3b). For both sets, distances from 20 to 100 meters to the structure centre were considered (as depicted by the red circles in Figure 4.3b). These were derived from the sensor range (100m) and the fact that, when viewed from a low distance, the scans will be will only represent a small portion of the structure, which can generate ambiguity.

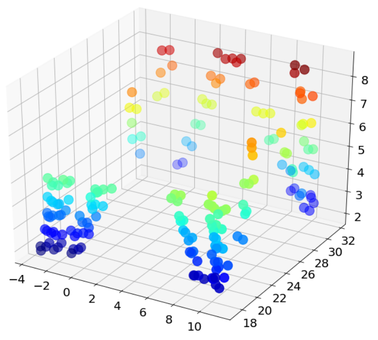


(a) Train Data.

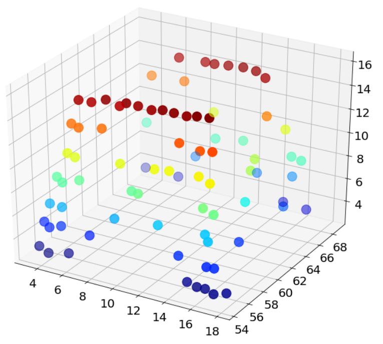
(b) Test Data.

Figure 4.3: Example ASV teleport locations in the data gathering process for a single turbine and noise configuration.

As we can observe in Figure 4.4, higher distances can affect the sample by making the point cloud sparser and taller. Figure 4.5, shows the effect of different perspectives, namely it differs in which face gets more detail and what parts of the structure are occluded.

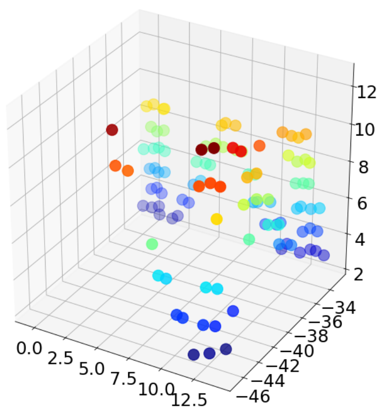


(a) 25 meters from the center.



(b) 60 meters from the center.

Figure 4.4: Resulting point clouds sampled by the Velodyne Lidar from different distances.



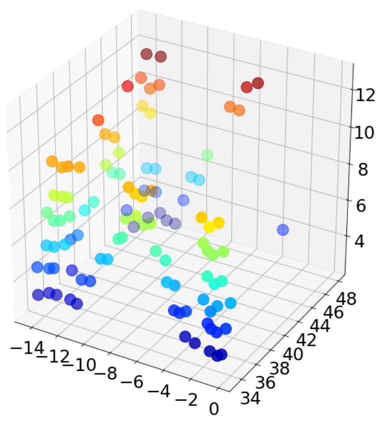
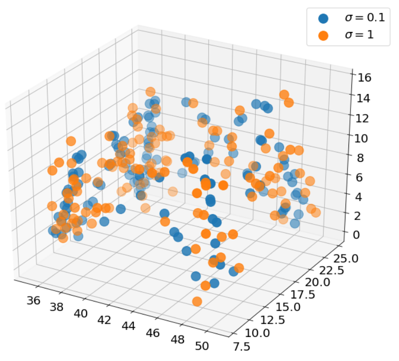


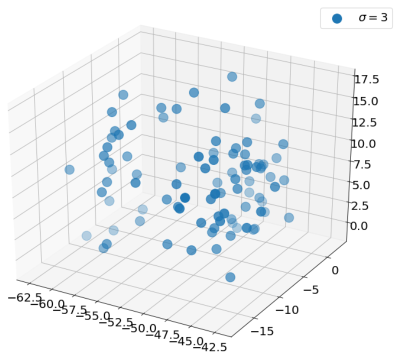
Figure 4.5: Resulting point clouds sampled by the Velodyne Lidar with perspective variations.

To approximate the simulation data to what real life acquisition would yield, different configurations of Gaussian noise are added to the training and testing data (see Figure 4.6). The noise is modelled as a normal distribution of mean μ and standard deviation σ. For each point in the point cloud, three different values are generated and added to the x,y, and z field.

While all the train data will be concatenated to a larger dataset, the test data will be used as different smaller sets. This way we can observe how the classifier results vary with distinct conditions. Additionally, noise can work as an augmentation technique, for example, making the classifier more robust to random shifts in voxels, which is generally a fragility of 3D CNNs.



(a) Lower noise values.



(b) Extreme noise value.

Figure 4.6: Resulting point clouds sampled by the Velodyne Lidar with different σ values.

The resulting dataset is characterized in Table 4.1. Due to the large number of noise configurations for the test set, it ended up being larger than the train set. However, it will not be considered one large unified set, instead, it will be used as a group of smaller sets to study the classifier's behaviour under different noise conditions.

Table 4.1: Train and test set characterization for 12 different wind turbines.

	Train Set	Test Set
View Points per Turbine	288	100
Noise σ	0.1, 0.5, 1.5	0.1, 0.7, 1.3, 2, 3
Noise μ	0	-2.5, 0, 2.5
Noise Configurations	9	15
Total Set Size	10368	18000

4.2.2 Model Selection

To select the model, both its potential results and size were taken into account. Architectures like the 45-layered VRN [35] can be very computationally demanding which makes them prohibitive with the available resources and with prejudice to a real time application. Additionally, by using a smaller model there is a lower risk of overfitting, which is helpful due to the limited insight the synthetic data provides. On table 4.2, we can see a comparison between different classifiers. ModelNet10 [29] is used as a benchmark since it contains a similar number of classes to the problem at hand (12 different turbines).

While OctNet [37] can achieve good results and claim a fast execution, the difference is not that noticeable when compared to a 3D CNN with low-resolution voxel grids and the implementation becomes significantly more complex. PointNet [38] could allow to directly process the input without any type of data conversion, however, it is a limiting factor that it requires the input point

Table 4.2: Comparison between popular models for 3D data on the ModelNet10 [29] dataset. Values marked with * vary with resolution.

Model	Input Type	Parameters (millions)	Accuracy
VoxNet [34]	Voxel Grid	0.89	92%
LightNet [62]	Voxel Grid	0.3	93.94%
ShapeNet [29]	Voxel Grid	12.4	83.5%
PointNet [38]	Point Cloud	3.5	-
OctNet [37]	Octree	*	88-91*%

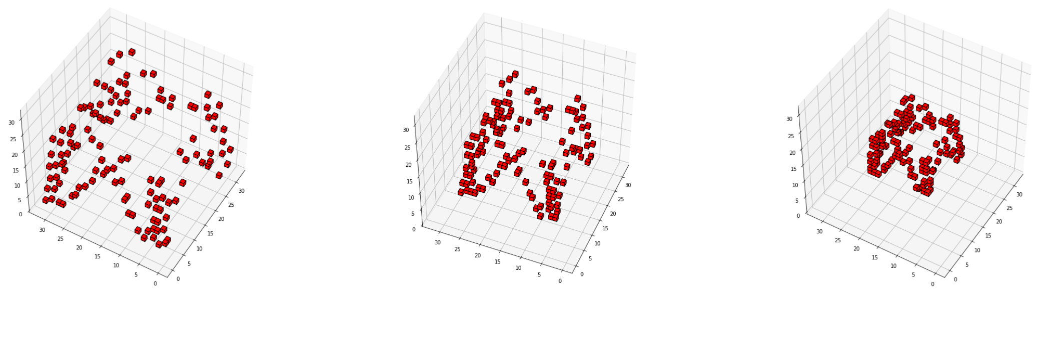
clouds to have the same number of points. This could be solved by up-sampling the point clouds, but that option was not explored during this work.

For the rest of this work, VoxNet [34] and LightNet [62] will be considered since they provide the best accuracy while requiring a small number of parameters to optimize.

4.2.3 Voxelization

To make the data compatible with the volumetric CNNs, a voxelization step is necessary. In some implementations, each voxel in the grid contains a probability of the space being filled or the number of points inside a voxel. While that information can be useful, binary voxelization can achieve similar performance on classification tasks [62]. For simplicity and efficiency, it was opted to use binary voxelization, i.e., each voxel is marked as 1 if it is part of the surface and 0 if it is empty.

Both VoxNet and LightNet use 32^3 grids as input, so this is also the chosen dimension in this implementation. The sparsity of the input point clouds can make it so the filled voxels end up very distanced, which can difficult the earlier convolutional layers ability to extract discriminative features. To counter this, the point clouds are first voxelized into a 24^3 grid and then zero-padded to a 32^3 grid. The effect of this procedure can be observed in Figure 4.7.



(a) Without padding (b) 24^3 voxel (c) 16^3 voxel

Figure 4.7: Resulting voxel grids of the same point cloud after padding to 32^3. All the grids have the same resolution.

4.2.4 Training

There are many training variables that can influence a machine learning model. Not only there are various tunable hyper-parameters, but choices like: optimizer, pre-training, and data augmentation can have significant unpredictable impact. So while a model "A" could achieve better results in general, it is possible that a model "B" could outperform it with a specific combination of optimizer and augmentation. However, it is unfeasible to test every possible combination, especially with limited computing and with prejudice to a real time application, so some assumptions must be made. During this section, different training conditions will be tested.

The approach is to, first, decide on a common training procedure, and parameters. After, experiment with different neural network architectures to detect any clearly advantageous results. Then, different optimizers will be tested. Lastly, the effects of transfer learning to enhance performance will be studied.

Due to its consistently good results, the Adam optimizer will be used with the default recommended parameters($\alpha = 0.001, \beta_1 = 0.99, \beta_2 = 0.999$). The training is done using mini-batch, of size 32. The loss function is cross entropy loss, as shown in equation 4.1. A dropout of $p = 0.3$ is implemented for the first convolution layer and $p = 0.2$ for the deeper ones.

$$L(p,y) = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{4.1}$$

Early stopping is implemented to halt the training before overfitting worsens the results. This means that, during training, a validation set is periodically tested on (5 times per epoch) to ensure the resulting loss keeps decreasing. For that purpose, a set with the noise configuration of $\mu = 0$, $\sigma = 1.3$ was used. The tolerance is set to roughly one epoch and a half, i.e. the training stops when the best validation score has not been surpassed 8 times in a row.

Since all the data is automatically generated, there are no cases of skewed classes. Meaning we can use accuracy as the main metric and forego additional ones like precision and recall.

Data Loading

The data loading method is depicted in Figure 4.8. Each of the sample's point cloud is stored on an individual file. These are organized in folders identifying their label and noise configuration. At the start of the training procedure, the folders are scanned for existing files, which are opened and read as required. This means that, only the ones being used as input are stored in memory. After reading, any augmentation operation is performed while in the point cloud state. Only then, is the data voxelized and fed to the model.

Model Comparison

Firstly, it must be decided which model to use. This is done by testing the different models to find out which one generalizes better for this problem. A customized VoxNet with added batch normalization layers (similar to [63]) is also experimented with. The training is done with early

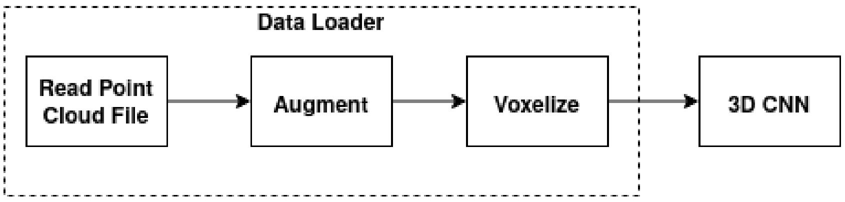
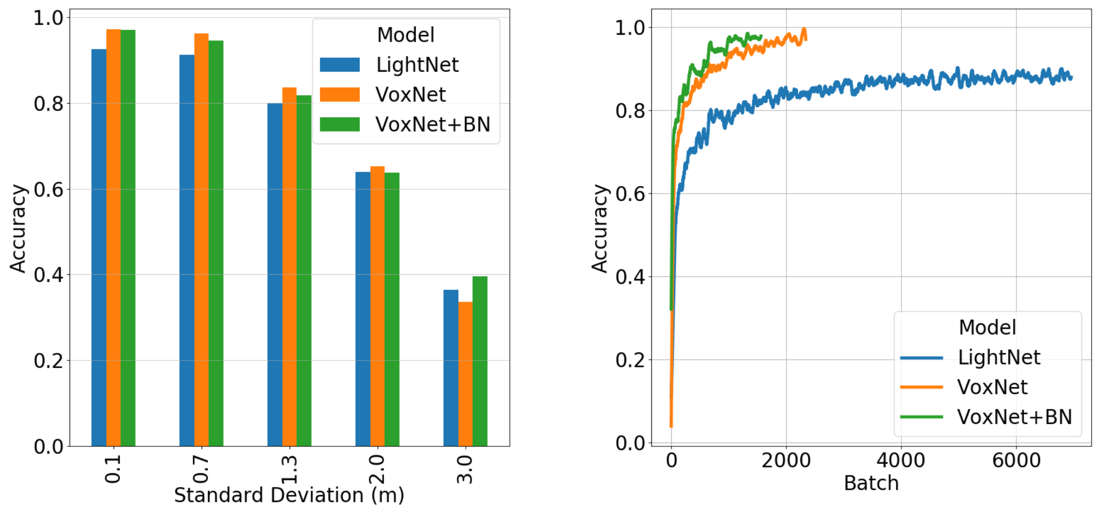


Figure 4.8: Data loading sequence during training.

stopping, as described above, using the Adam optimizer with the initial learning rate 0.001(divided by 10 every 10 epochs) and batch size 32.

The results are shown in Figure 4.9 where the base VoxNet architecture clearly outperforms the others. It should be noted that, while the custom VoxNet model got slightly worse results for the majority of the noise configurations as depicted by the green bar in Figure 4.9a, it achieved those in significantly less iterations than the other models, as it can be seen in Figure 4.9b (green line).



(a) Test accuracy results per noise configuration.

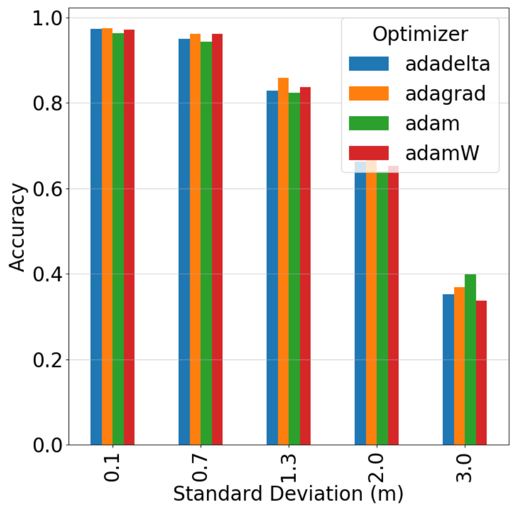
(b) Train accuracy evolution.

Figure 4.9: Accuracy of different models when trained on the generated dataset, that was presented on section 4.2.1.

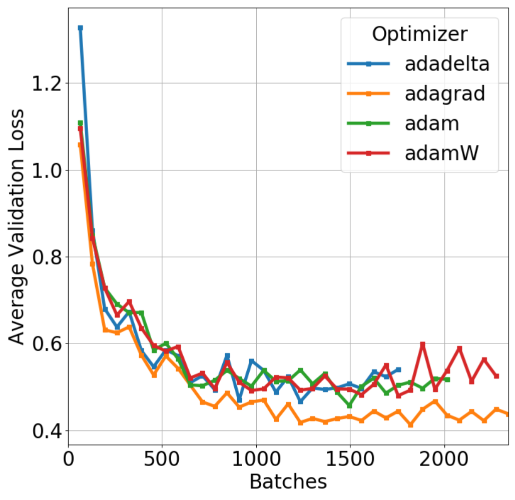
Optimizer

Having selected the model, additional sets of tests were performed to find out which optimizer can lead to better results. Optimizers with adaptative learning are beneficial for sparse data types [64], such as voxel grid. Therefore, only those types will be considered.

The compared optimizers are: adam, adagrad, adadelta, and adamw. Their results can be viewed in Figure 4.10, where adagrad gets better results both in accuracy (orange bars in Figure 4.10a) and loss (orange line in Figure 4.10b).



(a) Accuracy per noise configuration.



(b) Training loss evolution.

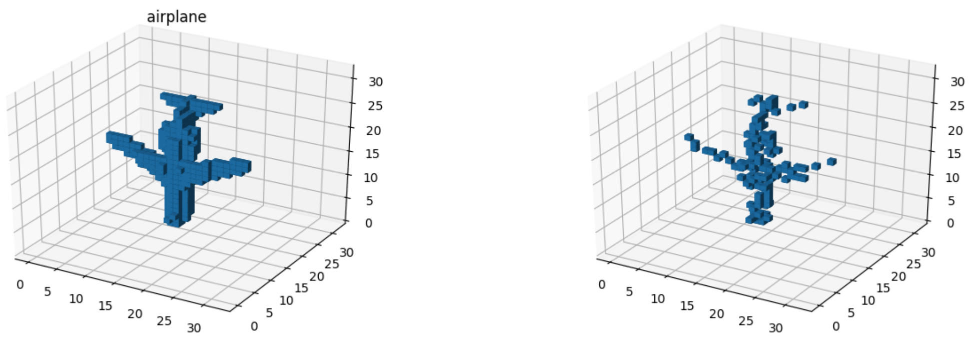
Figure 4.10: Optimizer comparison.

Transfer

To further improve the results, transfer learning is explored with two different datasets. The Sydney Urban Objects Dataset [65] includes 588 labelled objects, split across 26 labels, from outdoor environment. Like in our case, the dataset include incomplete point cloud scans taken from a Velodyne sensor. As this set is relatively small, it was used in its entirety for training, so a validation set was not used.

ModelNet40 [29] is a larger version of the ModelNet10 dataset, but as the name implies, it contains 40 classes instead of 10. It totals 12,311 manually designed CAD models, stored in a polygon format, so a sampled version was used (10,000 points per sample). The training in this dataset is done with rotation augmentation around the z axis. Additionally, to approximate it to our dataset, the point clouds are resampled to a lower, randomized, amount of points. The effects of these procedures can be observed in Figure 4.11.

After training on these sets, the model was then retrained on the turbine dataset, without layer freezing, and validated on the same sets as before. The results are depicted in Figure 4.12. It can be observed that the ModelNet pre-training yielded better results in general, while the Sydney dataset resulted in worsening results as the noise increases. To be noted that the results after pre-training on the Sydney dataset are slightly better for $\sigma = 0.1$, which can be due to the fact that its samples are cleaned up and barely present any noise.



(a) Regular voxelization (b) With augmentation

Figure 4.11: Resulting voxelization of the ModelNet models, with and without augmentation.

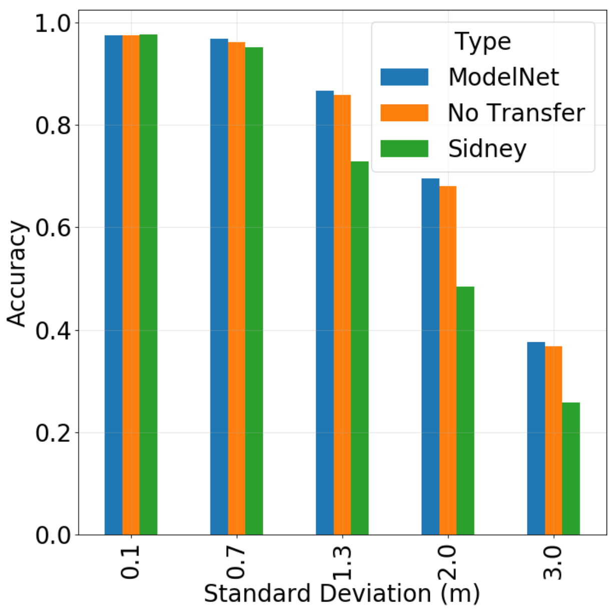


Figure 4.12: Validation results after performing transfer learning on different datasets.

4.3 Registration

This section is dedicated to the final step of the shape completion procedure. The goal is to obtain the missing structure when given an incomplete point cloud of an object (P_s) and its respective template (P_t). The approach is to, first, align the models, perform registration, and then there is a final step where the point clouds are subtracted to obtain the incomplete data from the template model.

4.3.1 ICP with Centroid Alignment

Initialization

One of the most influential steps in ICP [41] registration is the initial alignment. It is important to set a reasonable estimation to ensure it converges correctly instead of getting stuck in a local minimum.

There are multiple possible approaches to estimate this initial alignment depending on the available information. The first approach will be to perform a translation on P_t such that its centroid is aligned with that of P_s.

ICP Alignment

ICP is then employed to adjust the initial alignment. When applied, it outputs a rotation R and translation T matrix that minimizes the mean squared distance of each point in a source point cloud to their correspondent one on a target point cloud. For this implementation, point correspondences are obtained by searching for the current closest point.

By setting P_s as the source cloud it is possible to find a transform that fits in P_t. However, the goal is to align the template point cloud (P_t) on the incomplete cloud (P_s), not the other way around. On the other hand, by setting P_t as the source cloud, and P_s as the target cloud, a lot of the points will not have a correct correspondent on P_s. As a result, ICP will end up attempting to minimize the distance between points that should, ideally, be far apart and result in a larger error, as shown in Figure 4.13.

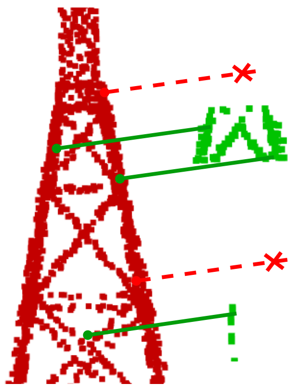


Figure 4.13: Ideal point correspondences from the template (left) to the incomplete (right) point cloud.

To overcome these issues, ICP is performed on P_s but rather than applying the resulting transform on it, their inverses are applied on P_t. The procedure is depicted in Algorithm 2.

While this method could achieve decent results in certain cases, it proved to be way too unreliable. Particularly, it struggled when the models face different directions and with the inconsistency of the sampled point clouds (e.g. lack of underwater data and number of occluded points in the back side), which affected the centroid position, as shown in Figure 4.14.

Algorithm 2 Registration Procedure.

```
1: function REGISTRATION($P_s$, $P_t$)
2:     $C_t \leftarrow$ COMPUTE_CENTROID($P_t$)
3:     $C_s \leftarrow$ COMPUTE_CENTROID($P_s$)
4:     $P_t$' $\leftarrow$ TRANSLATION($P_t$, $C_s - C_t$)
5:     $RT \leftarrow$ ICP($P_s$, $P_t$')
6:     $P_a \leftarrow$ TRANSFORM($P_t$', $RT^{-1}$) return $P_a$
7: end function
```

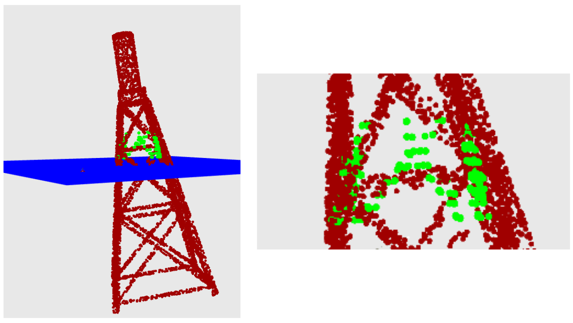


Figure 4.14: Failed alignment attempt. The blue plane represents the water level.

4.3.2 ICP with Particle Swarm Optimization

The previous method showed clear limitations, such as the risk of getting stuck in a local minimum, which can be too detrimental for the result, so there is a need for a more robust approach. The main limitation arises from the initial alignment. Not only it does not take orientation into account as the fact that the sensors typically yield more points from the emersed region than from the immersed region can also be a prejudicial factor. In some practical applications, this issue is solved by having a human manually perform the initial alignment. However, in this case, this process should be fully automated.

The idea is to run shorter ICP routines (limited to 2 iterations) for multiple different initial positions to find out which one leads to a smaller alignment error. For this purpose, a PSO-ICP algorithm, akin to the work proposed by Yu *et al.* in 2014 [48], was implemented. In this implementation, each particle represents an initial alignment transform for the template model. They travel in a 6-dimensional space, representing the 6 degrees of freedom (x, y, z, roll, pitch, and yaw). From the particle's position, a transformation matrix is computed (using the formulas in Figure 4.15), and then ICP is applied to fit the models and extract an error. Since repeatedly applying transformations on large point clouds can be computationally demanding, a similar approach to Section 4.3.1 is followed, with the addition that the initial alignment is also done in reverse, i.e., the inverted transformation matrix is applied on the incomplete point cloud.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z * R_y * R_x \qquad T = \begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix}^T \qquad RT = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

Figure 4.15: Transformation matrices for the 6 degrees of freedom. The angle in the rotation matrices R_x, R_y, and R_z represent roll, pitch, and yaw, respectively, while Δx, Δy, and Δz represent the translation along their axes

The particle updates equations are implemented as in equation 4.2. Where v_i represents the velocity on iteration i, x the particle position, x^l, x^g the local and global best, ω the inertia weighting, c_1 and c_2 the local and global acceleration coefficients, and r_1, r_2 are random values generated every iteration.

$$\begin{aligned} \mathbf{v}_{i+1} &= \omega \mathbf{v}_i + c_1 \mathbf{r}_1 \cdot \left(\mathbf{x}_i - \mathbf{x}_i^l \right) + c_2 \mathbf{r}_2 \cdot (\mathbf{x}_i - \mathbf{x}^g) \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{v}_i \end{aligned} \tag{4.2}$$

The proposed PSO-ICP algorithm flowchart can be observed in Figure 4.16, where the chosen termination criteria is the number of iterations.

PSO Parameters

One aspect of the PSO algorithm is the number of tunable hyperparameters. These include the number of particles, number of iterations, position bounds, inertia, maximum velocity, global, and local acceleration coefficients. The last four were tuned with iterative experimentation, and the first three will be analysed next.

The number of iterations and particles represent the biggest trade-off in computation time and result quality. By setting a large number of particles, there is a higher probability of them finding the global optimum. On the other hand, a large number of particles does not necessarily yield better significantly results as most, if not all, of them will converge to the same region. A short number of iterations carries a higher risk of the particles not finding a good solution. It relies more on the particle initialization and presents a higher risk of getting stuck in a local minima, which leads to more inconsistent results (see Figure 4.17a). By contrast, a larger number of iterations yield diminishing, or even null, gains and a high execution time, as depicted in Figure 4.17b.

Bounds were imposed on each particle's available space in the form of maximum and minimum value for each variable. By doing so, we can ensure that a lot of unreasonable values do not get evaluated, which helps to save computation time and to have the particles converge in fewer

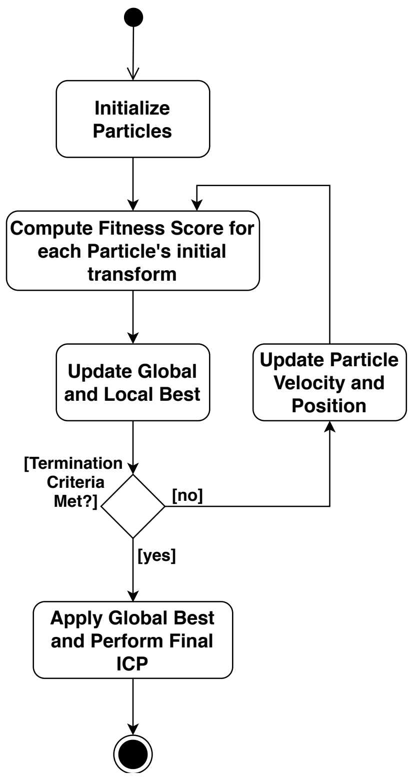

Initialize Particles
Compute Fitness Score for each Particle's initial transform
Update Global and Local Best
Update Particle Velocity and Position
[Termination Criteria Met?]
[no]
[yes]
Apply Global Best and Perform Final ICP

Figure 4.16: PSO-ICP algorithm state diagram. As implemented, the termination criteria is just the number of iterations.

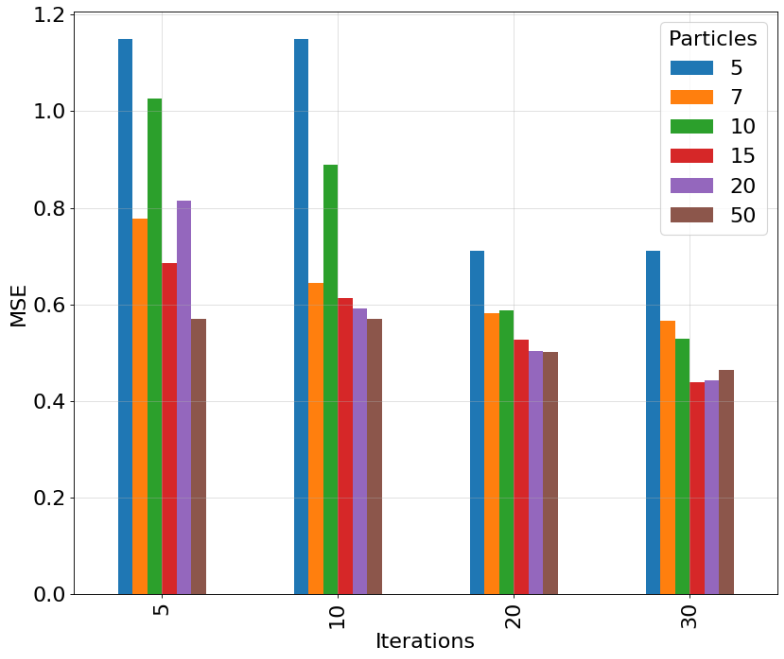
iterations. Though it is beneficial to have tighter bounds, there is also the risk of the ideal solution being left outside of them. This possibility is especially dangerous when taking into account the variety of dimensions that different 3D models can have. To tackle this, the bounds for the translation components (x,y) are computed based on the template's bounding box dimensions.

We can expect more points on the side facing the sensor, meaning that the centroid will drift towards it. With this in mind, it makes sense to use a shorter bounding distance between the Velodyne's position and the observed point cloud, as is depicted in Figure 4.18. As the turbine can be facing any direction relative to the origin, the bounds for yaw are set to the values of a full rotation($[-\pi,\pi]$). Since the type of structure being observed is known to always be upright(with some possible inclination due to the terrain, damage over time, etc...), roll and pitch are assigned tighter bounds($[-0.1,0.1]$).

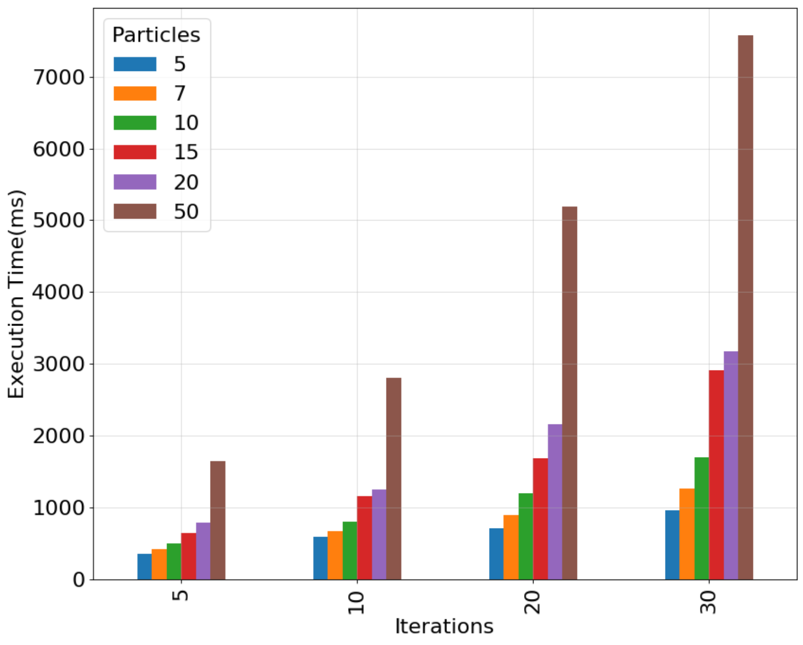
The final parameters are presented in Table 4.3.

Observations

During testing, this approach proved able to consistently find reasonable alignments for every used structure. The ability to explore different initial alignments in parallel, even if the next iteration leads to a larger error (see Figure 4.19), helps to avoid getting stuck in a local minimum. Additionally, by relying on ICP less to perform the main transformation and more as a way to



(a) Resulting Mean Squared Error. The error is measured in meters.)



(b) Execution Time.

Figure 4.17: Execution time and resulting error with different combinations of particle and iteration amounts. These results are obtained from the same single test case with a common random initialization seed.

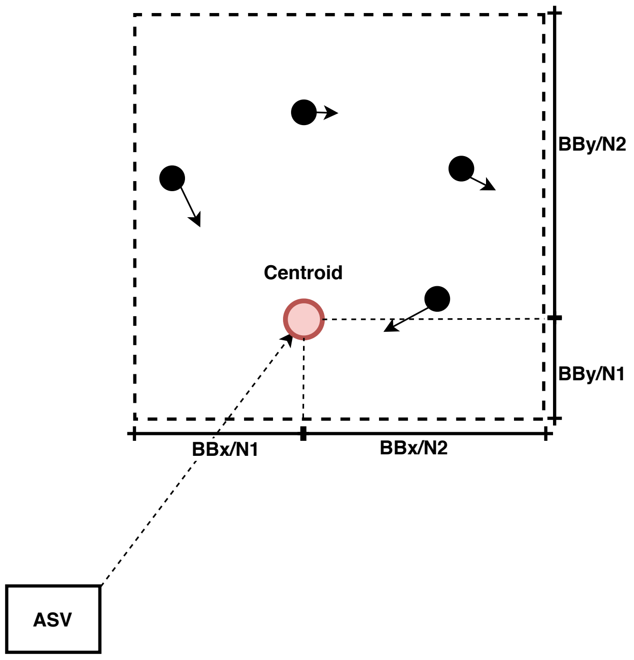


Figure 4.18: Particle Value Bounds for the x and y variables. BBx and BBy represent the bounding box width and height of the template model. N1 and N2 are factors to tighten these bounds.

Table 4.3: PSO Parameter values. N1 and N2 are the gains represented in Figure 4.18.

Variable	Value
Particles	7
Iterations	20
Global Coef.	1.5
Local Coef.	2
Inertia	0.79
N1	4
N2	2

measure an error and to make small adjustments allows us to restrain some transform components, such as high roll and pitch values.

The main downside to this approach is the execution time. Though some adjustments were made to decrease it, the time to finish all the iterations can hang around 1 second on the tested hardware.

4.4 Point Cloud Subtraction

The final step is to remove the overlapping regions from the template point cloud. This is achieved by searching for, and removing, every point in the template cloud within a certain radius from each point in the template cloud from the sensor point cloud. A direct implementation of this method can be computationally demanding, especially for a template cloud with a large number

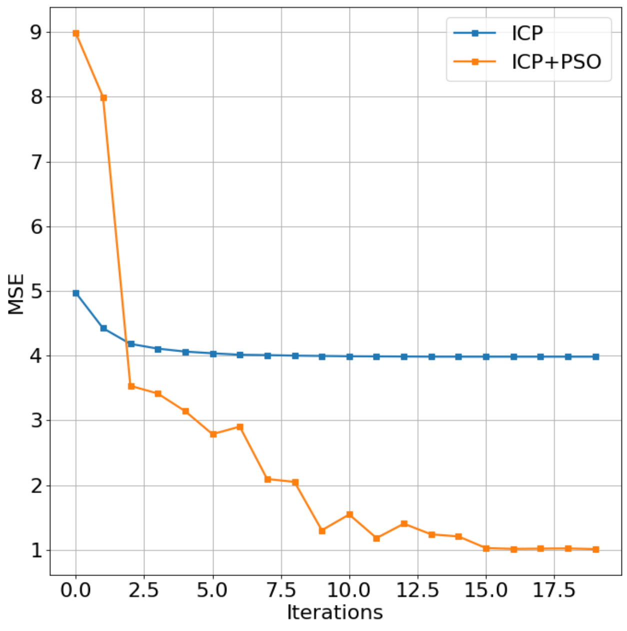


Figure 4.19: Mean squared error evolution with both approaches. The value for each ICP+PSO iteration represents the minimum MSE of every particle.

of points. As such, the approach is to convert the template cloud to a KdTree and perform radius search [66]. The procedure is depicted in Figure 4.20.

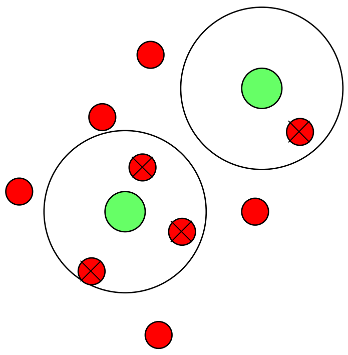


Figure 4.20: Subtraction Operation. The green points are the sensor data, while the red ones originate from the template cloud. The ones marked with a 'X' are removed after the operation.

The removal radius is a tunable parameter for this procedure. Figure 4.21 shows the effect of larger and smaller values after performing the PSO-ICP registration.

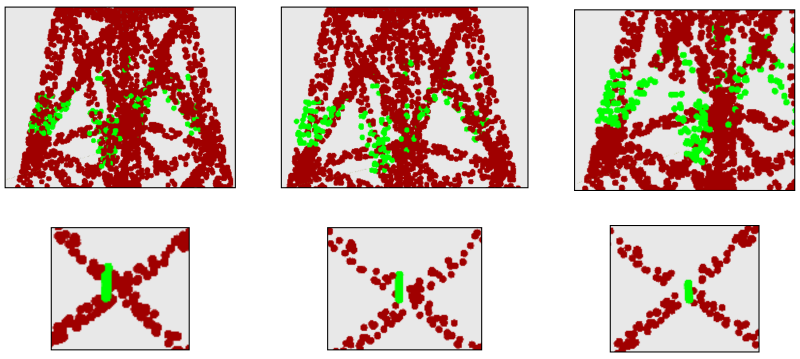


Figure 4.21: Results with different ranges ($0m$, $0.5m$, and $1m$ represented from the left to right images, respectively).

Chapter 5

Results

5.1 Classifier

In this section the results from the classifier module are presented. The validation datasets generated in Section 4.2.1 are used to evaluate the effect of noise and perspective on the final results.

5.1.1 Noise Effect

It can be seen in Figure 5.1 that, unlike the noise standard deviation, the noise mean has little to no influence on the result. This can be attributed to the voxelization, because, while the effect of noise mean is to shift the whole point cloud in a diagonal direction, it also moves its entire bounding box in that direction, and the relative position between points remains unchanged. This means that the resulting voxel grid will end up the same. The small variations between the noise means can be attributed to the difference in sampling positions since a new set of random position is generated for each one.

The following confusion matrices (Figure 5.2) provide additional insight on the classifier performance. For lower noise values, the results are consistent, but higher noise can affect some structures more than the others. Some confusion can be seen around the different twisted variants (TJ0-TJ90), which makes sense given their similarity.

The Monopile is the most affected by the noise. Since its dimensions are significantly smaller, so is its bounding box, as such, moving a point by 1 meter will lead to a more noticeable transformation when converting to a voxel grid, than it does for a larger structure like WindFloat.

5.1.2 Distance Effect

Figure 5.3 presents a relation between distance, acquired point cloud size, and the softmax value of the correct label. Since larger structures occupy a larger portion of the sensor's field of view, it yields more points, so the data is restricted to a group with similar dimensions: three-legged and twisted jackets.

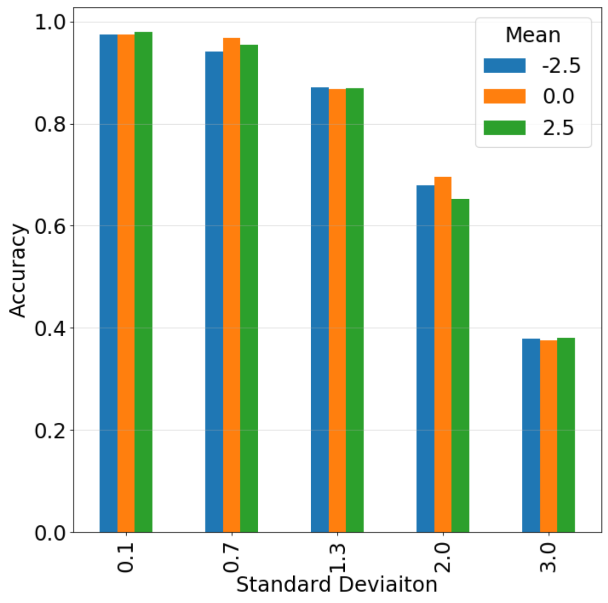


Figure 5.1: Accuracy for different values of Gaussian noise standard deviation and mean.

While the softmax value is unsuitable as a measurement for "prediction certainty", it does confirm the idea that the probability of picking the correct structure diminishes with distance. This can be attributed to lower detail caused by the sparseness of the available 3D data, becoming less distinctive from longer ranges..

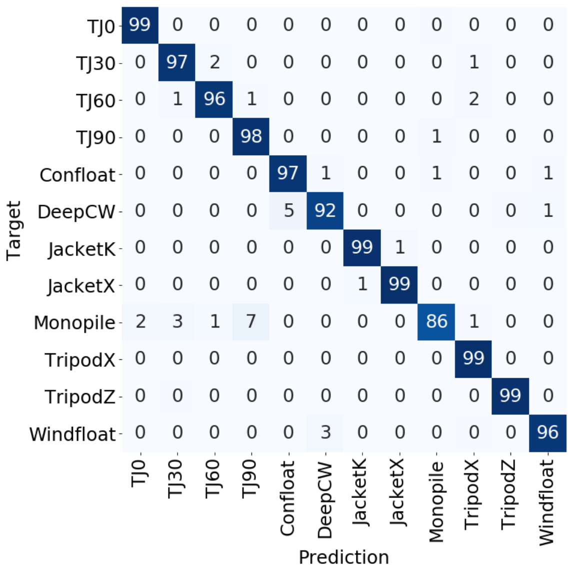
As was mentioned in Section 5.1.1, the monopile is the most sensible type to noise effect, due to its dimensions. However, it is a case where distance can improve the classification results, as can be seen in Figure 5.4. This happens because the higher distance will make the visible surface taller, and since the voxels in a voxel grid represent a cubical region, the filled region will be compressed into a smaller area, thus making the noise effect less noticeable. This effect can be seen in Figure 5.5.

5.2 Registration

5.2.1 Particle Evolution

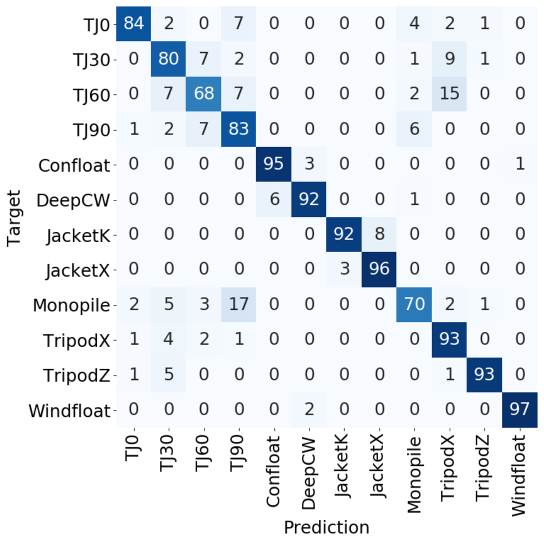
Tracking the PSO particles positions can give some insight into the algorithm's performance. In Figure 5.6 there is an example of this evolution on a case with 10 particles. To facilitate visualization some parameters were adjusted (e.g. the max velocity was lowered to create less erratic movements) and only the X and Y values are shown.

The intended PSO behaviour can be observed as the particles start from disperse positions and converge to the same region, even though they sometimes select sub-optimal solutions, thus exploring its surroundings to find a more optimal solution, while avoiding getting stuck in local



Target \ Prediction	TJ0	TJ30	TJ60	TJ90	Confloat	DeepCW	JacketK	JacketX	Monopile	TripodX	TripodZ	Windfloat
TJ0	99	0	0	0	0	0	0	0	0	0	0	0
TJ30	0	97	2	0	0	0	0	0	0	1	0	0
TJ60	0	1	96	1	0	0	0	0	0	2	0	0
TJ90	0	0	0	98	0	0	0	0	1	0	0	0
Confloat	0	0	0	0	97	1	0	0	1	0	0	1
DeepCW	0	0	0	0	5	92	0	0	0	0	0	1
JacketK	0	0	0	0	0	0	99	1	0	0	0	0
JacketX	0	0	0	0	0	0	1	99	0	0	0	0
Monopile	2	3	1	7	0	0	0	0	86	1	0	0
TripodX	0	0	0	0	0	0	0	0	0	99	0	0
TripodZ	0	0	0	0	0	0	0	0	0	0	99	0
Windfloat	0	0	0	0	0	3	0	0	0	0	0	96

(a) Standard Deviation 0.1 and 0.7



Target \ Prediction	TJ0	TJ30	TJ60	TJ90	Confloat	DeepCW	JacketK	JacketX	Monopile	TripodX	TripodZ	Windfloat
TJ0	84	2	0	7	0	0	0	0	4	2	1	0
TJ30	0	80	7	2	0	0	0	0	1	9	1	0
TJ60	0	7	68	7	0	0	0	0	2	15	0	0
TJ90	1	2	7	83	0	0	0	0	6	0	0	0
Confloat	0	0	0	0	95	3	0	0	0	0	0	1
DeepCW	0	0	0	0	6	92	0	0	1	0	0	0
JacketK	0	0	0	0	0	0	92	8	0	0	0	0
JacketX	0	0	0	0	0	0	3	96	0	0	0	0
Monopile	2	5	3	17	0	0	0	0	70	2	1	0
TripodX	1	4	2	1	0	0	0	0	0	93	0	0
TripodZ	1	5	0	0	0	0	0	0	0	1	93	0
Windfloat	0	0	0	0	0	2	0	0	0	0	0	97

(b) Standard Deviation 1.3

Figure 5.2: Confusion matrices with different noise standard deviations. The numbers for each cell are normalized such that each row adds up to 100(may have rounding errors).

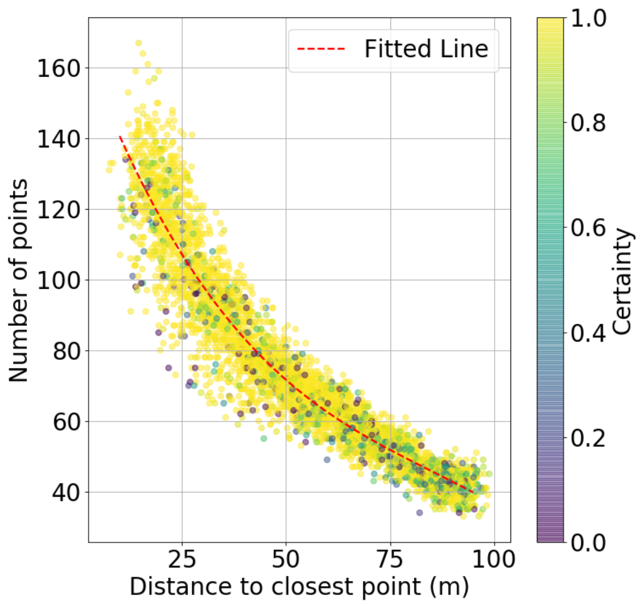
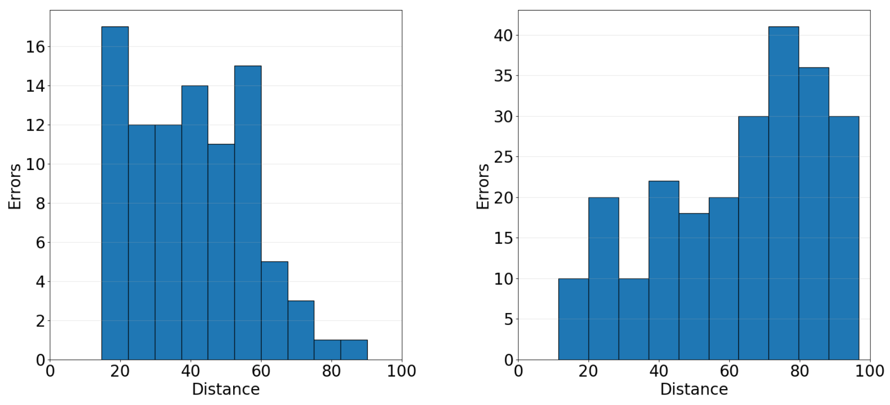


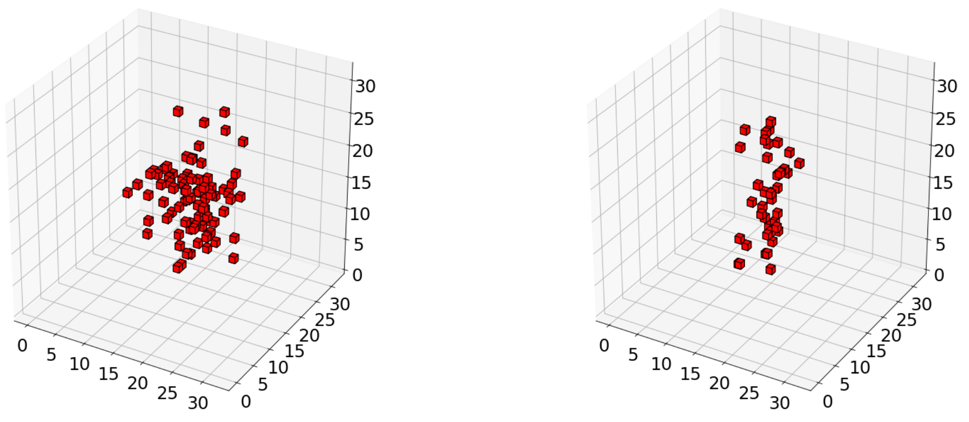
Figure 5.3: Number of points and distance to each acquired points cloud on the twisted and tripod structures, where there was a correct prediction. The colour represents the output of the correct after running a softmax function.



(a) Monopile (b) Others

Figure 5.4: Number of classification errors to distance relation between monopile and others at $\sigma = 1.3m$.

minima. The evolution of the error function can be seen in Figure 5.7. Where the smaller the deviation the better performance. As it can be seen the particle fitness score may become worse from one iteration to another, selecting less accurate solutions to generate variability in the search pool. This is caused by the stochastic behaviour given by the random weights applied to the local and global best during the particle position and velocity update step.



(a) 30 meters (b) 90 meters

Figure 5.5: Voxelized samples captured from a monopile turbine at different distances with $\sigma = 1.5$ with a sideways point of view.

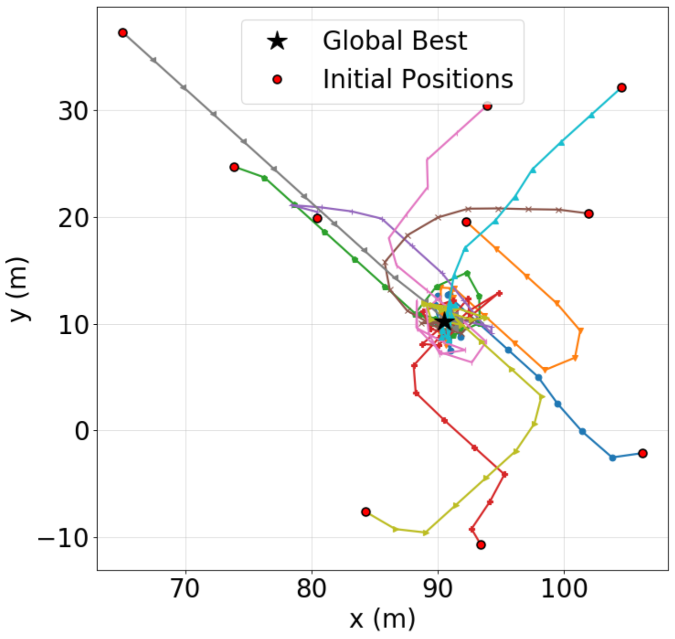


Figure 5.6: Particle evolution for the x and y values. Each colour represents a particle.

In Figure 5.8, a similar test case is displayed with additional information on the Z value and using the marker colour to display the fitness error in that position. To make the results consistent, the roll, pitch and yaw were locked to their solution values before executing the procedure.

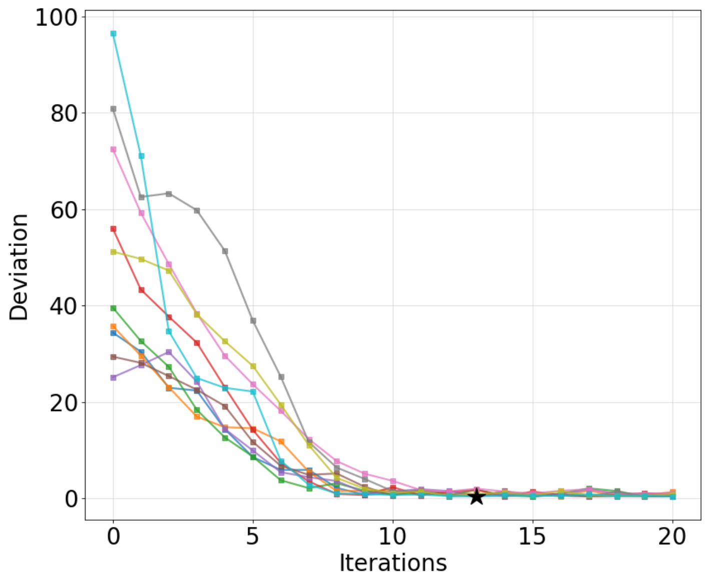


Figure 5.7: Deviation over time for each particle. The black star marks the global best.

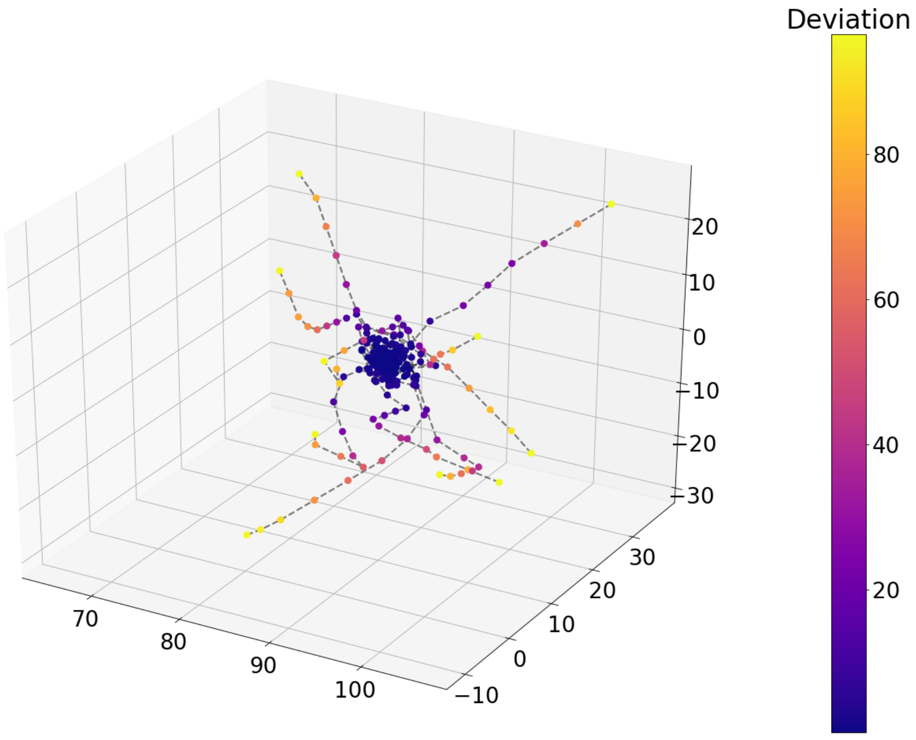


Figure 5.8: Particle evolution for the X,Y and Z values. The colour represents the fitting error after performing ICP.

5.2.2 Timings

The PSO-ICP step is the longest one of the whole procedure, taking up more than half of the total execution time, despite the optimizations done.

With the PSO-ICP method, three components can affect the execution time: number of ICP iterations, PSO parameters (maximum iterations, number of particles), and point cloud sizes. The first two are predefined and constant, and so is the number of points from the template point cloud. This leaves out the size of the sensor point cloud, which can vary with many factors (as discussed in Section 5.1). This is exalted by the fact that, during the evaluation, every transform operation is performed on the sensor point cloud. This is still desirable since the incomplete point cloud is always significantly smaller than the template's, but it does lead to varying execution times.

An experiment was run to observe this timing variation. A point cloud of 800 points was randomly sampled to different values in steps of 50 and its execution time was evaluated. The results are shown in Figure 5.9, which shows that relation is roughly linear($O(n)$). This begs the question about whether this could be used as an optimization step to sample the sensor output to a lower number of points before performing the PSO-ICP evaluation and how that could affect the quality of the results.

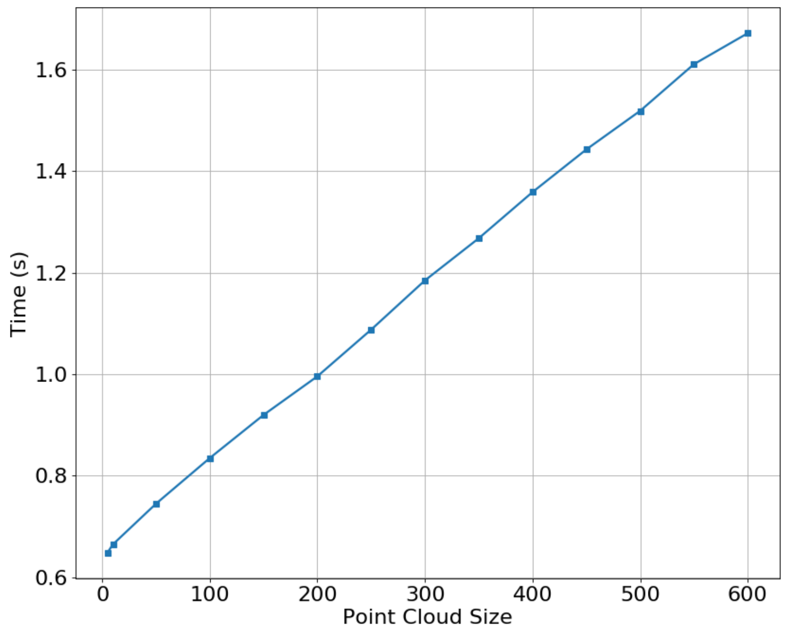
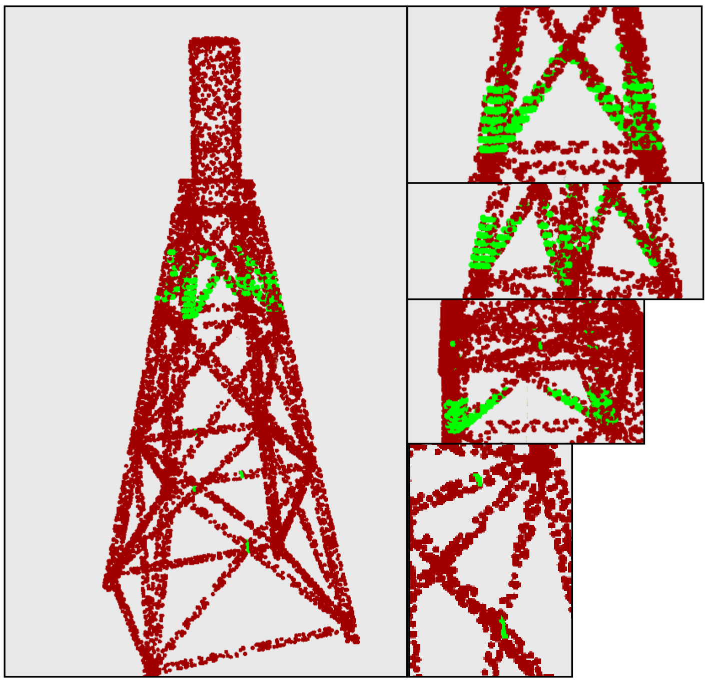


Figure 5.9: PSO processing time for different point cloud size.

5.2.3 Final Results

The following results were taken after with the whole integrated system, i.e. filtering, classification, and registration. Sensor noise of standard deviation 0.5 and mean 0 is used. Two examples can be seen in Figures 5.10 and 5.11 for fixed foundations and floating structures, respectively. Notice that the green dots represent the perceived 3D point clouds and the red dots are the completed data provided by the known models.


(a) jacketX

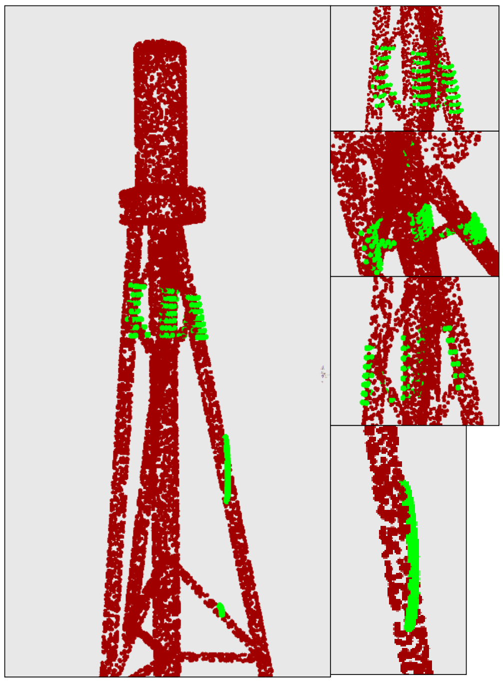

(b) TJ60

Figure 5.10: Registration result for fixed turbines. Green dots are acquired, red dots were added.

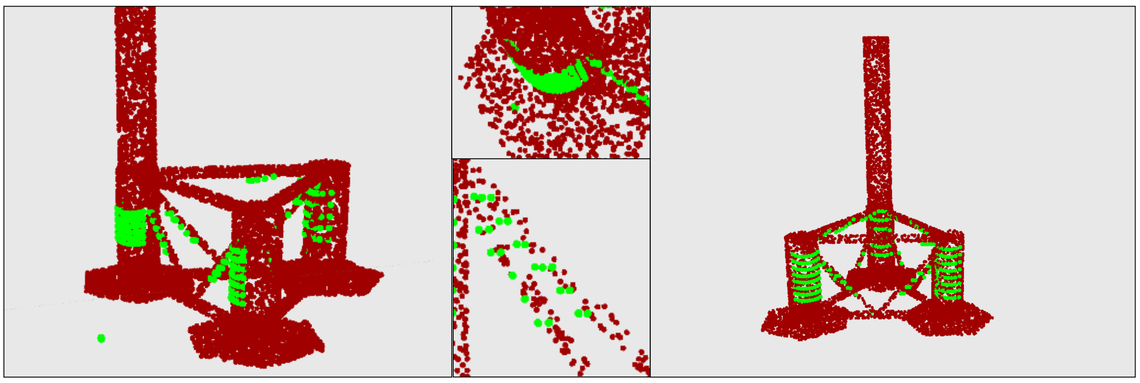


Figure 5.11: Windfloat completion. Green dots are acquired, red dots were added.

Throughout testing, the registration procedure worked reliably with different noise configurations and viewpoints, excluding special cases like when the classifier outputted the wrong turbine type. The biggest noted issues were:

- Jitter when running recursively.
- Almost symmetrical structures.

When running the procedure recursively, i.e. repeatedly take in new data to complete, a certain degree of jitter could be observed, where the structure would have slightly different tilts each iteration. This effect was more noticeable on the inferred underwater information, especially when the multibeam sonar did not capture any information from the immerse layer.

Figure 5.12 shows a case where the registration algorithm converged to an incorrect alignment where, despite a portion of the turbine's pole being observed, the template model does not reflect this. This is a risk for almost symmetrical structures, where an incorrect alignment can create a very low error due to the small proportion of misaligned points, when compared to the fitted ones. This can be attributed to the particle initialization, where one of them found a very low alignment error on the early iterations and pulled the others in through the "global best" metric.

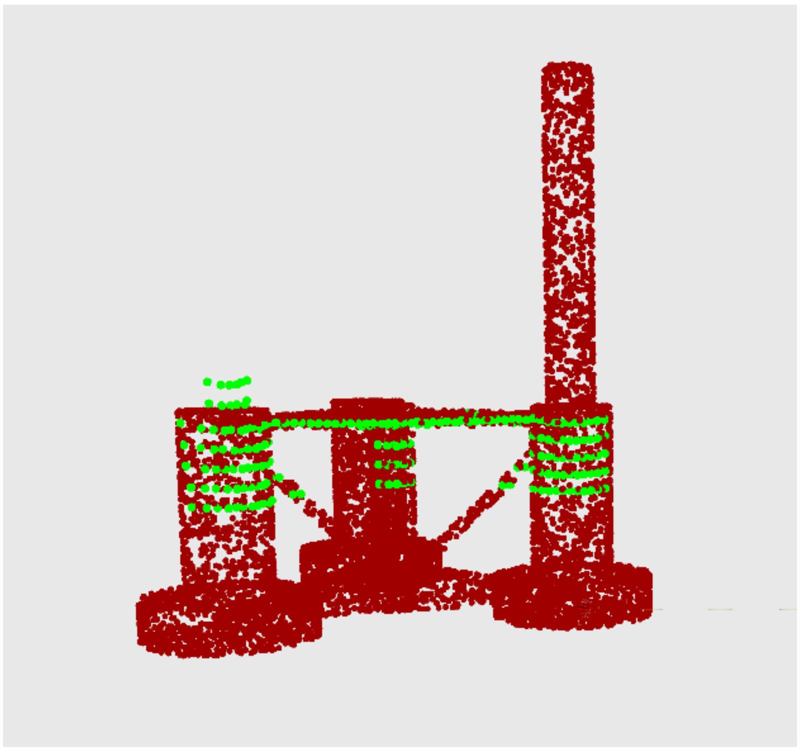


Figure 5.12: Alignment error.

On a good occasion (such as this one), this type of error can be harmless as the pole position is not relevant for the end application. However, there is also the danger that it could lead to an odd transformation that does not resemble the real structure. For example, if the braces were occluded, shifting the template upwards would yield a lower error. With that being said, these occurrences were rare during testing and got fixed on the next execution.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, a data imperfection problem was tackled in the context of inspection systems for offshore wind farms. Using database-prior shape completion technique, it was possible to reconstruct the shape of previously known man-made structures sensor data that yields incomplete information of the surrounding environment.

The work was implemented and tested in a realistic 3D simulator where a scenario with different wind turbines was created. 3D CAD models of real wind turbines were developed to generate a synthetic dataset for train and testing the proposed approach. To address the simulated to real data differences, distinct configurations of Gaussian noise were used.

The procedure contains four steps. First, the sensor data is filtered to remove unnecessary information using a pass-through filter and euclidean clustering. Then, the above water data is run through a 3D CNN, trained on a dataset developed for this problem, to classify the structure. With this information, a template of the complete model is retrieved and aligned on the sensor data with a PSO-ICP algorithm. Finally, there is a subtraction step to remove overlapping information from the template. This method was tested on a simulation environment, which provides limited insight and it could benefit from testing on a real-time scenario.

For the classification task, an automatic procedure was developed to generate a turbine dataset from the simulation environment. Different neural network models and parameters were tested and compared on different noise configurations to find the best performing combination. Results demonstrated that the model was able to correctly classify structures with more than 95% accuracy for the expected amount of noise with the used sensor. Other tests show the decrease in accuracy for a heavy amount of noise.

Two different registration methods were tested. Despite requiring more processing time, the PSO-ICP method proved to be more robust on the different testing scenarios. Optimizations were made to reduce its cost and make it viable to run on an online application. On the tested hardware, the execution time was lowered from 45 seconds to approximately 1 second.

Overall, the testing showed that the procedure could reliably detect, classify, and infer the missing structure on the simulated environment.

6.2 Future Work

To further develop this project, the following extensions are proposed:

- Expand the classification module to use data from other sensors, like the sonar and cameras.
- Extend the model database and classifier to include not only wind turbines, but also other offshore elements, such as electricity substations, vessels, and buoys. While, in this work, turbines were focused on, the approach is generic enough that those models could be implemented.
- Use a more robust approach to filter the sensor data.
- Experiment with denser point clouds, by using multiple views of the same object or point cloud upsampling.
- Experiment on a real life use case.
- Experiment with a more end-to-end approach, i.e. a single neural network model to perform detection, classification and pose estimation to improve performance.

Annex A

In this annex, the developed CAD models for fixed foundation turbines are presented as described in Section 3.4. They are labelled according to their names used for the classification results in Chapter 5.

Fixed Foundations

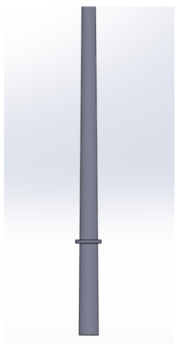
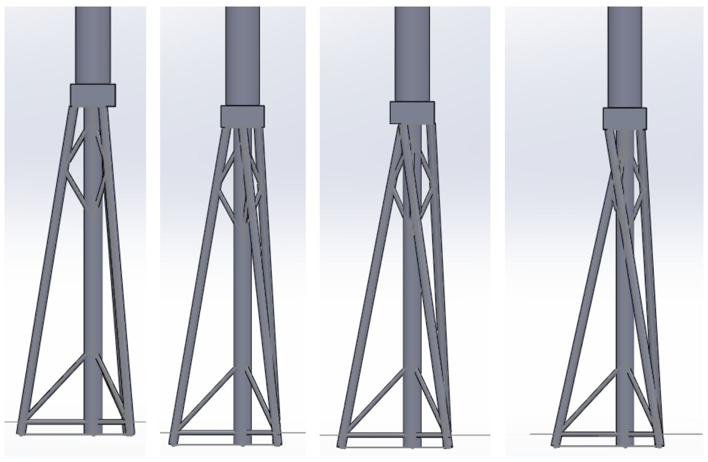
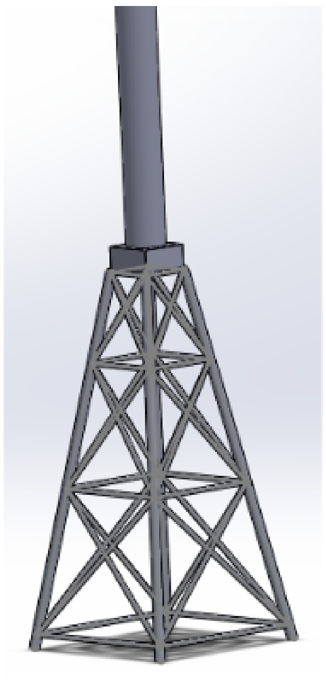


Figure 1: Monopile



(a) 0°(TJ0) (b) 30°(TJ30) (c) 60°(TJ60) (d) 90°(TJ90)

Figure 2: Twisted Jacket Foundations with different twist angles.


(a) JacketX

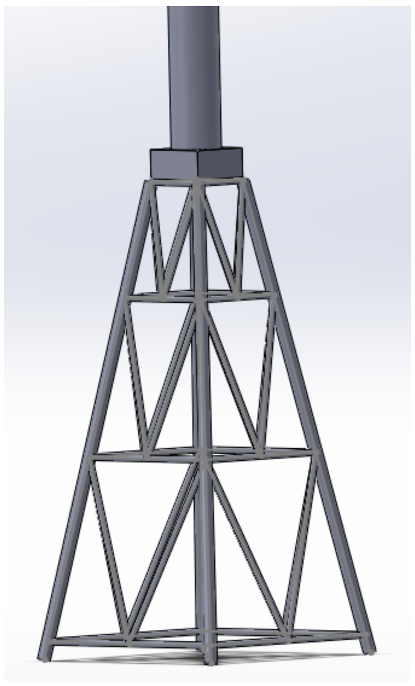
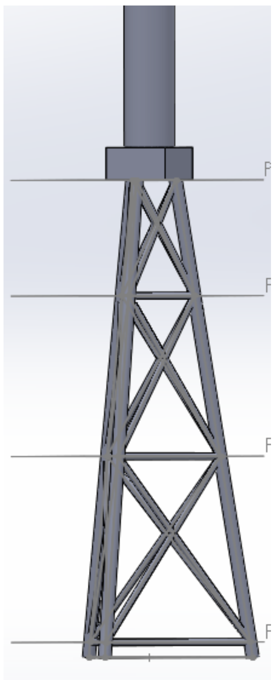

(b) JacketK

Figure 3: 4-Legged Jacket Foundations


(a) TripodX

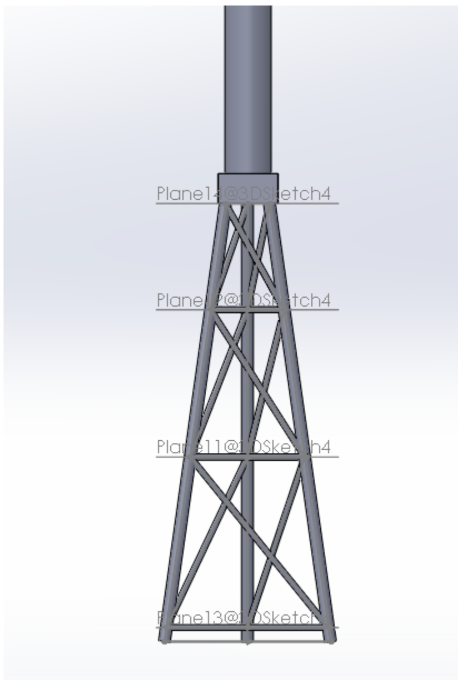
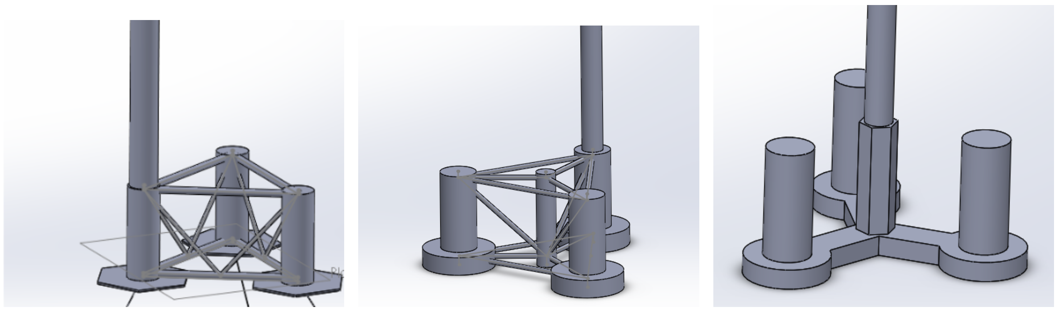

(b) TripodZ

Figure 4: 3-Legged Jacket Foundations

Floating Foundations



(a) WindFloat (b) DeepCW (c) ConFloat

Figure 5: Floating Foundations

Bibliography

[1] C. A. Dill, "Green energy: Insuring a renewables future," 2019.

[2] "Offshore wind power generation in the Sustainable Development Scenario, 2000-2030," 2020.

[3] L. Kishfy, "Modeling Vibrations on an Offshore Wind Turbine."

[4] F. Castro, "Primeiro parque eólico flutuante em Portugal já arrancou. Estará concluído até ao fim do verão," 2019.

[5] S. S. N. Cruz, A. Matos, S. Cunha, "Zarco – an Autonomous Craft for Underwater Surveys.," *In Proceedings of the 7th Geomatic Week.*, 2007.

[6] D. F. Campos, A. Matos, and A. M. Pinto, "Multi-domain mapping for offshore asset inspection using an autonomous surface vehicle," *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*, pp. 221–226, 2020.

[7] D. F. Campos, A. Matos, and A. M. Pinto, "An Adaptive Velocity Obstacle Avoidance Algorithm for Autonomous Surface Vehicles," *IEEE International Conference on Intelligent Robots and Systems*, pp. 8089–8096, 2019.

[8] J. Albiez, D. Cesar, C. Gaudig, S. Arnold, R. Cerqueira, T. Trocoli, G. Mimoso, R. Saback, and G. Neves, "Repeated close-distance visual inspections with an AUV," *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, 2016.

[9] B. Hoefakker, "Gemini Offshore Wind Park. (2019). The Gemini Offshore Wind Park."

[10] A. M. Pinto and A. C. Matos, "MARESye: A hybrid imaging system for underwater robotic applications," *Information Fusion*, vol. 55, no. August 2019, pp. 16–29, 2020.

[11] Y. K. Wu, W. C. Wu, and J. J. Zeng, "Key issues on the design of an offshore wind farm layout and its equivalent model," *Applied Sciences (Switzerland)*, vol. 9, no. 9, 2019.

[12] G+ Global Offshore Wind Health & Safety Organisation, "2018 Incident Data Report," 2019.

[13] K. Yin, H. Huang, H. Zhang, M. Gong, D. Cohen-Or, and B. Chen, "Morfit: Interactive surface reconstruction from incomplete point clouds with curve-driven topology and geometry control," *ACM Transactions on Graphics*, vol. 33, no. 6, 2014.

[14] W. Zhao, S. Gao, and H. Lin, "A robust hole-filling algorithm for triangular mesh," *Proceedings of 2007 10th IEEE International Conference on Computer Aided Design and Computer Graphics, CAD/Graphics 2007*, p. 22, 2007.

[15] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 1–13, 2013.

[16] P. Speciale, M. R. Oswald, A. Cohen, and M. Pollefeys, "A symmetry prior for convex variational 3D reconstruction," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9912 LNCS, pp. 313–328, 2016.

[17] S. Thrun and B. Wegbreit, "Shape from symmetry," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. II, pp. 1824–1833, 2005.

[18] N. J. Mitra, L. J. Guibas, and M. Pauly, "Partial and approximate symmetry detection for 3D geometry," *ACM SIGGRAPH 2006 Papers, SIGGRAPH '06*, pp. 560–568, 2006.

[19] I. Sipiran, R. Gregor, and T. Schreck, "Approximate Symmetry Detection in Partial 3D Meshes," *Computer Graphics Forum*, vol. 33, no. 7, pp. 131–140, 2014.

[20] D. Li, T. Shao, H. Wu, and K. Zhou, "Shape completion from a single RGBD image," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 7, pp. 1809–1822, 2017.

[21] M. Pauly, N. Mitra, and J. Giesen, "Example-Based 3D Scan Completion.," *Symposium on Geometry Processing*, p. 23, 2005.

[22] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, vol. 26, no. 3, 2007.

[23] Y. Li, A. Dai, L. Guibas, and M. Nießner, "Database-Assisted Object Retrieval for Real-Time 3D Reconstruction," vol. 34, no. 2, pp. 435–446, 2015.

[24] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik, "Aligning 3D models to RGB-D images of cluttered scenes," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 4731–4740, 2015.

[25] H. Liu, Y. Cong, C. Yang, and Y. Tang, "Efficient 3D object recognition via geometric information preservation," *Pattern Recognition*, vol. 92, pp. 135–145, 2019.

[26] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu, "High-Resolution Shape Completion Using Deep Neural Networks for Global Structure and Local Geometry Inference," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 85–93, 2017.

[27] A. Sharma, O. Grau, and M. Fritz, "VConv-DAE: Deep volumetric shape learning without object labels," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9915 LNCS, pp. 236–250, 2016.

[28] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, "Semantic Scene Completion from a Single Depth Image," 2016.

[29] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1912–1920, 2015.

[30] A. Dai, C. R. Qi, and M. Nießner, "Shape completion using 3D-encoder-predictor CNNs and shape synthesis," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6545–6554, 2017.

[31] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann, "Shape Inpainting Using 3D Generative Adversarial Network and Recurrent Convolutional Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2317–2325, 2017.

[32] O. Litany, A. Bronstein, M. Bronstein, and A. Makadia, "Deformable Shape Completion with Graph Convolutional Autoencoders," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1886–1895, IEEE, 2018.

[33] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," *Proc of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 16, no. 3, pp. 403–412, 2010.

[34] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, 2015.

[35] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and Discriminative Voxel Modeling with Convolutional Neural Networks," 2016.

[36] Donaldmeagher, "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, vol. 19, pp. 129–147, 1982.

[37] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6620–6629, 2017.

[38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 77–85, 2017.

[39] Z. Cao, Q. Huang, and R. Karthik, "3D object classification via spherical projections," *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, pp. 566–574, 2018.

[40] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman, "Convolutional neural networks on surfaces via seamless toric covers," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[41] P. J. Besl and Neil D. McKay, "A Method for Registering of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–255, 1992.

[42] M. Greenspan and G. Godin, "A nearest neighbor method for efficient ICP," *Proceedings of International Conference on 3-D Digital Imaging and Modeling, 3DIM*, vol. 2001-Janua, pp. 161–168, 2001.

[43] T. Jost and H. Hügli, "A multi-resolution scheme ICP algorithm for fast shape registration," *Proceedings - 1st International Symposium on 3D Data Processing Visualization and Transmission, 3DPVT 2002*, pp. 540–543, 2002.

[44] G. C. Sharp, S. W. Lee, D. K. Wehe, and A. Arbor, "ICP Registration using Invariant Features 1 Introduction," *Technology*, vol. 24, no. 1, pp. 90–102, 2000.

[45] E. Trucco, A. Fusiello, and V. Roberto, "Robust motion and correspondence of noisy 3-D point sets with missing data," *Pattern Recognition Letters*, vol. 20, no. 9, pp. 889–898, 1999.

[46] L. Shen, H. Huang, F. Makedon, and A. J. Saykin, "Efficient registration of 3D SPHARM surfaces," *Proceedings - Fourth Canadian Conference on Computer and Robot Vision, CRV 2007*, pp. 81–88, 2007.

[47] J. Yang, H. Li, D. Campbell, and Y. Jia, "Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2241–2254, 2016.

[48] Q. Yu and K. Wang, "A hybrid point cloud alignment method combining particle swarm optimization and iterative closest point method," *Advances in Manufacturing*, vol. 2, no. 1, pp. 32–38, 2014.

[49] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey, "PointNetLK: Robust & Efficient Point Cloud Registration using PointNet," 2019.

[50] J. Groß, A. Osep, and B. Leibe, "AlignNet-3D: Fast Point Cloud Registration of Partially Observed Objects," 2019.

[51] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[52] B. Bingham, C. Agüero, M. McCarrin, J. Klamo, J. Malia, K. Allen, T. Lum, M. Rawson, and R. Waqar, "Toward Maritime Robotic Simulation in Gazebo," 2020.

[53] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation," *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, 2016.

[54] H. Yoshida, H. Fujimoto, D. Kawano, Y. Goto, M. Tsuchimoto, and K. Sato, "Range extension autonomous driving for electric vehicles based on optimal velocity trajectory and driving braking force distribution considering road gradient information," *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, no. Figure 1, pp. 4754–4759, 2015.

[55] M. Wittingen, *Offshore Wind Turbine Monopile Foun- dation Installation with a Dynamic Posi- tioned Vessel*. PhD thesis, Delft University of Technology, 2018.

[56] I. W. Chen, B. L. Wong, Y. H. Lin, S. W. Chau, and H. H. Huang, "Design and analysis of jacket substructures for offshore wind turbines," *Energies*, vol. 9, no. 4, 2016.

[57] K. R. Hussein, A. W. Hussein, E. H. Hegazy, and A. A. Amin, "Structural design of a floating foundation for offshore wind turbines in red sea," *Analysis and Design of Marine Structures - Proceedings of the 4th International Conference on Marine Structures, MARSTRUCT 2013*, no. March, pp. 575–583, 2013.

[58] G. Johannes, *WindFloat design for different turbine sizes*. PhD thesis, Técnico Lisboa, 2014.

[59] S. Zhou, B. Shan, Y. Xiao, C. Li, G. Hu, X. Song, Y. Liu, and Y. Hu, "Directionality effects of aligned wind and wave loads on a Y-shape semi-submersible floating wind turbine under rated operational conditions," *Energies*, vol. 10, no. 12, 2017.

[60] "WindFloat Pilot Project Phase," 2018.

[61] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," *KI - Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[62] S. Zhi, Y. Liu, X. Li, and Y. Guo, "Toward real-time 3D object recognition: A lightweight volumetric CNN framework using multitask learning," *Computers and Graphics (Pergamon)*, vol. 71, pp. 199–207, 2018.

[63] J. C. Su, M. Gadelha, R. Wang, and S. Maji, "A deeper look at 3D shape classifiers," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11131 LNCS, pp. 645–661, 2019.

[64] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–14, 2016.

[65] M. De Deuge, A. Quadros, C. Hung, and B. Douillard, "Unsupervised feature learning for classification of outdoor 3d scans BT - Australasian Conference on Robitics and Automation," *Australasian Conf. Robotics Automat. (ACRA)*, vol. 2, pp. 2–4, 2013.

[66] R. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," 2011.