ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL Robotic Systems Lab
Prof. Dr. Marco Hutter

Semester Thesis

# Learning Edge Costs and Node Sampling for Navigation Graphs

**Autumn Term 2023**

**Supervised by:**
Marko Bjelonic
Alexander Reske

**Author:**
Lorenzo Piglia

# Declaration of Originality

I hereby declare that the written work I have submitted entitled

**Global Path Planning with Learned Edge Costs and Node Sampling**

is original work which I alone have authored and which is written in my own words.[1]

**Author(s)**

Lorenzo Piglia

**Student supervisor(s)**

Marko Bjelonic
Alexander Reske

**Committee members(s)**

| First name | Last name |
|---|---|

**Supervising lecturer**

| Marco | Hutter |
|---|---|

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

| Place and date | Signature |
|---|---|

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

# Intellectual Property Agreement

The student acted under the supervision of Prof. Hutter and contributed to research of his group. Research results of students outside the scope of an employment contract with ETH Zurich belong to the students themselves. The results of the student within the present thesis shall be exploited by ETH Zurich, possibly together with results of other contributors in the same field. To facilitate and to enable a common exploitation of all combined research results, the student hereby assigns his rights to the research results to ETH Zurich. In exchange, the student shall be treated like an employee of ETH Zurich with respect to any income generated due to the research results.
This agreement regulates the rights to the created research results.

### 1. Intellectual Property Rights

1. The student assigns his/her rights to the research results, including inventions and works protected by copyright, but not including his moral rights ("Urheberpersönlichkeitsrechte"), to ETH Zurich. Herewith, he cedes, in particular, all rights for commercial exploitations of research results to ETH Zurich. He is doing this voluntarily and with full awareness, in order to facilitate the commercial exploitation of the created Research Results. The student's moral rights ("Urheberpersönlichkeitsrechte") shall not be affected by this assignment.
2. In exchange, the student will be compensated by ETH Zurich in the case of income through the commercial exploitation of research results. Compensation will be made as if the student was an employee of ETH Zurich and according to the guidelines "Richtlinien für die wirtschaftliche Verwertung von Forschungsergebnissen der ETH Zürich".
3. The student agrees to keep all research results confidential. This obligation to confidentiality shall persist until he or she is informed by ETH Zurich that the intellectual property rights to the research results have been protected through patent applications or other adequate measures or that no protection is sought, but not longer than 12 months after the collaborator has signed this agreement.
4. If a patent application is filed for an invention based on the research results, the student will duly provide all necessary signatures. He/she also agrees to be available whenever his aid is necessary in the course of the patent application process, e.g. to respond to questions of patent examiners or the like.

### 2. Settlement of Disagreements

Should disagreements arise out between the parties, the parties will make an effort to settle them between them in good faith. In case of failure of these agreements, Swiss Law shall be applied and the Courts of Zurich shall have exclusive jurisdiction.

| | |
|---|---|
| Place and date | Signature |

# Contents

# Preface

I would like to thank Marco Hutter and the RSL lab for the wonderful opportunity and the tools they provided to accomplish this project. A special thanks also to A. Reske and M. Bjelonic for the support and the insights they provided to carry on my work.

# Abstract

Autonomous robots face significant challenges when navigating unknown environments: a successful navigation system necessitates a robust representation of the environment and a suitable plan for the robot to reach its target. To solve this task, the construction of a navigation graph has emerged as a widely used approach in robotics research; however, generating this graph for complex and large 3D environments through simulation-based approaches can be computationally expensive due to the complex mechanics of robot motion and interactions with the environment. This limitation renders these approaches unsuitable for real-time applications, especially in dynamic or unpredictable environments.

We present a novel approach that utilizes deep learning techniques, specifically graph neural networks, to learn edge costs and node sampling for navigation graphs. Our approach leverages Point Cloud Data to directly estimate the edge costs of the navigation graph, which enables us to extract information about path traversability and predict other metrics, such as the Cost of Transport. Our results demonstrate the generalizability of our approach to different environments and its robustness to perturbed data. Furthermore, we show that our approach can also perform non-uniform sampling of the points and bias the sampling toward regions where an optimal solution for the path planning problem is more likely to be found.

In summary, our approach provides a promising solution for efficient and effective navigation in unknown environments by learning the edge costs and node sampling of the navigation graph directly from Point Cloud Data. Our results demonstrate that our approach has the potential to overcome the computational limitations of simulation-based approaches and is well-suited for real-time applications in dynamic or unpredictable environments.

# Symbols

## Symbols

| | |
|---|---|
| $\omega$ | angular velocity $[rad/s]$ |
| $\tau$ | torque $[N \cdot m]$ |
| $g$ | gravitational constant $[m/s^2]$ |
| $v$ | linear velocity $[m/s]$ |

## Indices

| | |
|---|---|
| $x$ | x axis |
| $y$ | y axis |

## Acronyms and Abbreviations

| | |
|---|---|
| ETH | Eidgenössische Technische Hochschule |
| HPH | Hörsaalgebäude Physik |
| PCD | Point Cloud Data |
| GPP | Global Path Planning |
| CVAE | Conditional Variational AutoEncoder |
| CNN | Convolutional Neural Network |
| SLAM | Simultaneous Localization and mapping |
| BCE | Binary-Cross Entropy (loss) |
| CoT | Cost of Transport |

# Chapter 1

# Introduction

Global Path Planning (GPP) is an essential component of autonomous navigation in complex environments. The problem of Global Path Planning involves finding a safe and efficient path for a robot to travel from its start location to a goal location in an environment that can be either known in advance (for example in the case of a controlled facility) or unknown and potentially dynamic, with obstacles like doors that can open or block entire paths (Figure 1.1).

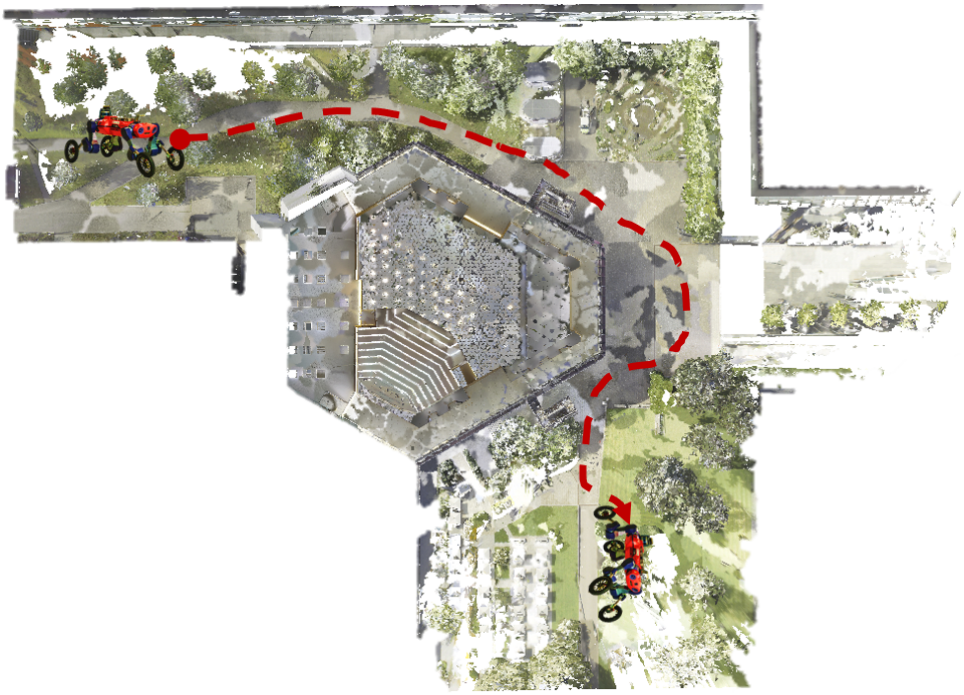


Figure 1.1: Example of a solution to the Global Path Planning problem: The robot generates a path through the environment avoiding detours and obstacles

The Global Path Planning problem is an important aspect of various autonomous applications, such as autonomous vehicles, search and rescue robots, delivery robots, warehouse management, exploration and many others. These applications require reliable path planning algorithms that can handle complex environments and sometimes also need to respond to dynamic changes in real time. The Path Planning algorithm must be able to efficiently navigate through different terrains and obstacles, such as stairs or irregular surfaces, to fully exploit the mobility capabilities of

the robot.
The challenges are mainly due to the complex and dynamic nature of the environment, as well as the non-linear and non-convex nature of the robot's motion constraints. In fact, finding an optimal solution to the path planning problem is an NP-hard problem.
Finding an effective heuristic for the controller's performance is also difficult, as it requires balancing multiple competing objectives, such as minimizing travel time, energy consumption, and collision risk.
Finally, Global Path Planning also needs to consider uncertainties, such as sensor noise, and localization errors. These uncertainties can lead to sub-optimal or unsafe paths, making it necessary to design path planners that can handle such uncertainties robustly.

In this work, we present a novel approach to Global Path Planning that leverages the 3D model of the environment (usually a reality capture model) and PRM [1] to construct navigation graphs that can then be used to solve the GPP task.
Starting from the approach introduced in [2], this work aims to demonstrate that by utilizing the point cloud representation of the digital twin (see 2), we can effectively predict edge costs in an environment-agnostic manner by using machine learning. More specifically a Dynamic Graph Convolutional Neural Network (DGCNN) [3] can learn to output a latent representation from a sparse point cloud that can be used to infer the traversability of the environment. This approach is significantly faster than traditional simulation-based methods and can be implemented in real-time on the robot platform. As a result, the navigation system becomes more resilient to changes in the environment and can be easily adapted to different types of environments.
To this end, we conduct a thorough analysis of the robustness of our approach to missing and/or noisy data, to prove its adaptability to real-world data. Additionally, we show that our approach is flexible enough to be applied to various regression targets, such as the Cost of Transport [4] metric, thereby increasing its versatility.
Finally, we adopt a similar approach as outlined in prior research, such as [5] and [6], which use a CVAE (Conditional Variational AutoEncoder) to generate new poses in the configuration space, to demonstrate the effectiveness of our method in applying non-uniform sampling strategies that prioritize sampling in regions where an optimal solution is more likely to be found by leveraging the point cloud description of the environment. Thus we show that this approach can also be used to build the PRM graph of an environment proving that our navigation system can potentially be run completely online on the robot platform with the only input being the PCD of the environment.

In conclusion, our approach offers several advantages over traditional simulation-based methods, including improved efficiency and real-time implementation.
Throughout this document, we will provide an in-depth analysis of the results, including robustness to missing and noisy data, adaptability to different regression targets, and the use of non-uniform sampling strategies.

# Chapter 2

# Related Work

In literature, there exist several approaches that can be used to solve the GPP problem, including Grid-Based methods [7] [8] [9] [10], Potential Field methods[11] [12] [13] [14], and Sampling-Based planners [15] [16] [17]. Each of these methods has its own advantages and disadvantages, and the choice of method depends on the specific requirements of the application. Sampling-based planners, such as Probabilistic Roadmaps (PRMs) [1], are considered to be the most appropriate for large scale 3D environments, spanning multi-floor areas and indoor and outdoor areas. PRMs approximate the topology of the configuration space, which is the set of all possible robot poses, by constructing a graph or tree using sampled points in the collision-free subset of the configuration space and connecting these points if there is a collision-free local path between them. This approach allows for efficient planning while taking into account the complexity of the environment.
To perform this we need a representation of the operational world in which the robot will be deployed. One common approach is to use topological graphs [18] [19], where the environment is abstracted as nodes and edges representing possible locations and transitions between them. Another approach is to use obstacle maps [20] [21], where the environment is represented as a grid map with occupied and free cells. Height maps [22] [23] [24] are similar to obstacle maps but also include information about the elevation of the terrain. However, these approaches have limitations when applied to large 3D environments that have multi-floor structures and span both indoor and outdoor areas because they perform some form of abstraction of the real world.
Among many approaches, digital twins representations proved to be very effective to represent the operational world of the robot; they are created using data from sensors, simulations, and other sources to accurately represent the characteristics and behavior of the real-world object or system, effectively serving as the indistinguishable digital counterpart of the real-world. Unlike other forms of representation, the digital twin representation is able to handle large 3D environments and does not require any conversion of the real-world environment, thus avoiding information loss. Moreover, it allows for the modeling of dynamic elements within the environment, such as doors that can open and close, by continuously updating the virtual model in real-time, for example by refining it with sensor data.

We will provide now an overview of the various approaches that have been proposed for constructing navigation graphs, with a focus on the methods used for estimating the edge costs. This information is crucial in order to understand the contributions and limitations of our proposed approach.
First, we discuss the simulation-based methods, which use a trained agent with a local navigation policy and use it to simulate the robot along all the edges of the

navigation graph in the digital twin environment. These methods typically rely on complex physics simulations and can be computationally expensive, which makes them unsuitable for real-time applications.
Then, we will discuss the research that has been conducted in the field of learning navigation cost by using machine learning techniques on different map representation.
Next, we examine the recent advancements in the field of pointcloud learning, specifically deep learning architectures designed to operate on sparse 3D pointclouds. These approaches have shown promising results in a number of related tasks, such as object classification and semantic segmentation and in our work we show that they can be adapted to the task of edge cost estimation.
Finally, we explore the research in the field of learned sampling distributions, that is, where a sampling distribution is learned from demonstrations, and then used to bias sampling toward regions of the space that the robot should explore. These approaches aim to optimize the sampling process by selecting points from the environment that are most relevant for solving the navigation task.
In the following sections, we will elaborate further on these related works, and show how our proposed approach leverages these ideas to provide a new and more effective solution for global path planning.

## 2.1 Simulation-Based Methods

Simulation-based methods for constructing navigation graphs typically involve simulating the motion of a robot in a digital replica of the physical environment. These methods rely on a simulation framework, such as Isaac Gym, MuJoCo, or Gazebo, as well as a digital twin representation of the environment. The results of the simulation are then used to construct a graph of possible paths.

One notable example of this approach is PRM-RL [25]. PRM-RL is a method for constructing navigation graphs that leverages the strengths of both Probabilistic Roadmaps (PRMs) and Reinforcement Learning (RL). In this method, a digital twin of the environment is generated and a PRM is constructed by sampling the state space of the environment. The PRM is then used to guide an RL agent to learn how to navigate the environment. The resulting navigation graph represents the set of valid paths that the agent has learned to follow.
Another related work is "Gaitmesh"[26]. Where the authors use a physics simulation engine to simulate the robot's motion in the environment, taking into account the robot's gait and the terrain's topology. The resulting navigation graph is a controller-aware representation of the environment that is well suited for long-range legged locomotion planning.
Finally, the work of A. Unagar [2] uses a pre-defined control policy and relies on the PRM graph to avoid the obstacles, thus without the need to train a local obstacle avoidance policy. Furthermore it also assign a traversability score to the edges of the graph instead of just avoiding obstacles. Finally it also sets the way for using a Neural Network to regress the edges score from a pointcloud, however this approach is only used to perform global path re-planning (in case of environment changes).

These works demonstrate the potential of digital twins and simulation-based approaches for constructing navigation graphs. However, they also have some limitations, such as the fact that the virtual model can be quite large in size and can be challenging to store on the robot platform; moreover, since the simulation process can be computationally intensive it must be completed offline, making it difficult to adapt to dynamic or unknown environments where the digital twin may need

to be updated. This can result in a need for repeated simulations every time the environment changes, making the approach less resilient in these situations.
Nevertheless, they serve as a foundation for further research in this field and highlight the importance of exploring new methods for constructing navigation graphs that overcome these limitations.

Our work starts from [2] and shows how the navigation graph can be built directly from the PCD for the whole environment and not just used for path replanning. Furthermore, we experiment with different architectures, regression targets, datasets, and parameters to show that our approach is resilient to noisy or missing data. Additionally, we demonstrate how the Probabilistic Road Map (PRM) component of our approach can also be substituted by a neural network to bias the point sampling process.

## 2.2 Learning Navigation costs

The use of neural networks for estimating navigation costs in the environment has been an active area of research in recent years. In [22], the authors proposed a method where a neural network is trained to map the high-dimensional and complex state space of the environment to a lower-dimensional and semantically enriched representation. This representation is then used as a heuristic to speed up the motion planning process, leading to improved efficiency and robustness compared to traditional methods.
Another approach, as described in [24], presents a planner that combines reachability and body volumes, with a convolutional neural network used to predict foothold scores for stepping. This approach combines PRM with efficient graph expansion and adaptive sampling for real-time performance in cluttered and narrow environments. In [27], the authors focus on end-to-end point-to-point and path-following navigation for a robot to avoid moving obstacles in small, static environments. The results demonstrate that the policies trained using AutoRL are successful, robust to noise, and do not suffer from the issue of catastrophic forgetfulness commonly seen in other deep reinforcement learning algorithms.
Despite demonstrating promising outcomes, these methodologies are restricted to small-scale and simple environments and utilize a form of approximation for mapping. As a result, they are unsuitable for complex 3D environments at a large scale.

## 2.3 PointCloud Learning

This section will examine the various techniques for point cloud learning found in the literature. One of the most widely used methods for point cloud learning is PointNet [28]. This deep learning architecture was introduced for the purpose of 3D shape classification and segmentation, and operates directly on point clouds without using intermediate representations.
PointNet++ [29] is an improvement on the PointNet architecture, designed to capture local structures in the input point clouds. This is achieved through the recursive application of PointNet on a nested partitioning of the input point set and by utilizing metric space distances to learn local features at increasing contextual scales. Additionally, PointNet++ introduces set learning layers to handle varying point densities, resulting in a more flexible and robust architecture for deep learning on point clouds, which outperforms PointNet on challenging benchmarks.

Dynamic Graph Convolutional Neural Network (DGCNN) [3] is another neural network architecture specifically designed to work with point clouds. It leverages graph convolutional layers to process the point cloud data, which can be represented as a graph with each point as a node and defins the edges based on certain criteria, such as distance between points. The graph convolutional layers in DGCNN update the features of each node based on the features of its neighboring nodes, allowing the network to learn patterns in the point cloud data that are not visible from individual points. DGCNNs are particularly suitable for point clouds due to their ability to handle the unordered and irregular nature of such data, whereas traditional convolutional neural networks are not well suited for this task. Furthermore, DGCNNs can handle large point clouds with a high number of points by using a k-nearest neighbor graph, reducing computational complexity. Compared to PointNet++, DGCNN has several advantages, such as utilizing graph convolutions and EdgeConv layers to effectively capture local structures in the input point clouds, and utilizing a dynamic k-NN graph construction that allows for robustness to variations in point density.
In [30], the authors build upon the DGCNN approach by removing the transformation network, linking hierarchical features from dynamic graphs, freezing the feature extractor, and retraining the classifier, claiming an improved performance in the results.
In [31] the authors present a new approach to point convolution, called Kernel Point Convolution (KPConv), which operates directly on point clouds without intermediate representations. The convolution weights are located in Euclidean space by kernel points and applied to input points close to them, resulting in more flexibility than fixed grid convolutions. The locations of the kernel points can be learned by the network, allowing for the extension to deformable convolutions that adapt to local geometry. While all of these approaches have their strengths and weaknesses, due to the novelty of our work, we cannot rely on standard evaluation metrics and benchmark results, hence the present work explores and evaluates different architectures, and selects the one that produces the best results on the navigation graph problem.

## 2.4 Sampling Strategies

In the field of robotics, using a learned distribution for sampling points from the environment is a promising strategy for guiding exploration or decision-making. The conventional approaches to sampling involve either random sampling or uniform sampling. However, these methods can be inadequate for robotic systems that have limited access to the state space and can become computationally expensive for large environments.
There have been several papers that have explored the use of learned distributions for various tasks in robotics: In [5] the authors propose a method for learning a sampling distribution from demonstrations, which is then used to bias sampling towards regions of the state space where the optimal solution is likely to be found. This is achieved through a conditional variational autoencoder, which computes the sampling distribution and generates samples from the latent space, conditioned on the specific planning problem. The proposed method is general and can be used with any sampling-based planner, while maintaining the theoretical guarantees of sampling-based approaches. The results of several planning problems show that the proposed methodology effectively learns representations of the relevant regions of the state space, leading to a significant improvement in terms of success rate and convergence to the optimal cost. In [6], the authors present a method to directly learn an informed distribution of views based on the spatial context in the robot's

map. Additionally, they investigate methods for learning the information gain of each sample. They demonstrate that their approach has the potential to enhance both efficiency and robustness in exploration tasks in robotics.

# Chapter 3

# Learning Edge Costs

In this chapter we will describe how our method works for learning navigation costs. In the following sections, we will describe each step of our architecture in detail and provide empirical results to demonstrate its performance on various datasets.

## 3.1 Architecture

Our approach consists of two distinct steps. First, we utilize a Probabilistic Roadmap (PRM) of the environment and its corresponding Point Cloud Digital (PCD) twin to sample points along the edges of the PRM graph. Second, we employ a neural network that takes these sampled points as inputs and produces a regression score for each edge. These scores are then aggregated to construct the final navigation graph. A comprehensive overview of our method is provided in Figure 3.1.

### 3.1.1 Sample Generation

In this section of the thesis, we delve into the crucial step of generating data samples from the point cloud representation of the environment, which will serve as the input to our proposed architecture. As we will demonstrate later on, the architecture can also learn how to perform this task without any prior input.

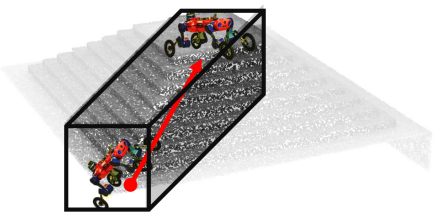


Figure 3.2: Sample generation

To achieve this, we make use of the digital twin of the environment, represented in the form of a Point Cloud Data (PCD). From this PCD, we extract a number of points around each edge in the Probabilistic Roadmap (PRM) graph, serving as our input. The selected points are located within a maximum distance from both the start pose and the goal pose in the PCD. This maximum distance is an important hyperparameter of our approach, as we will show in the Results section that the optimal value is approximately 2 meters (see A.2).

As illustrated in Figure 3.2, we only sample points that are close to the edge. To ensure a constant number of sampled points, we either randomly drop out or duplicate points as necessary.

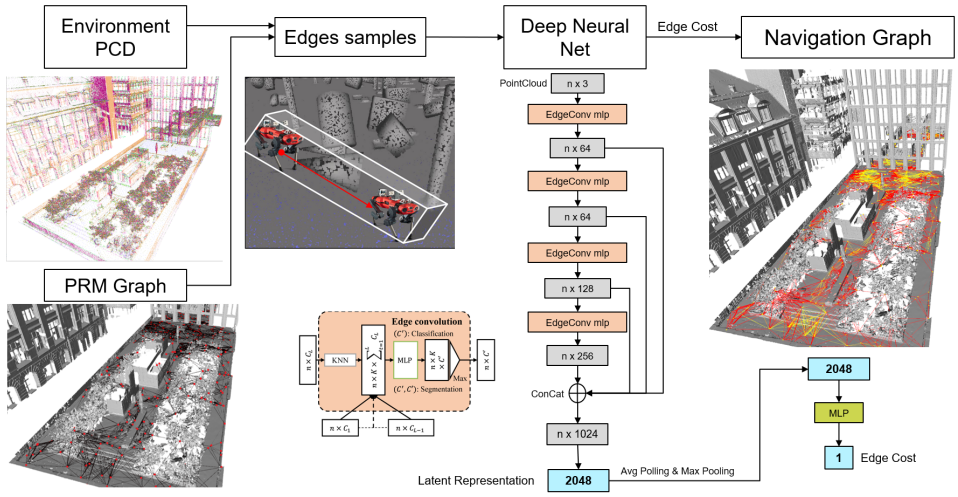
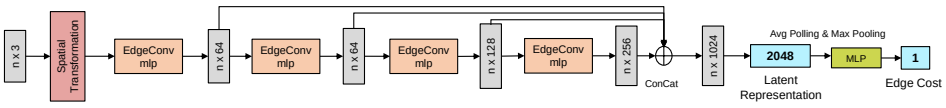



Figure 3.1: Flowchart of the Complete Architecture

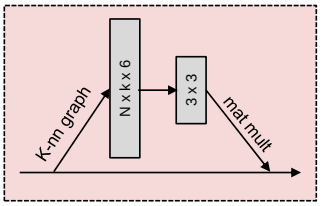
### 3.1.2 Neural Network Architecture

In this section, we examine how we can use the information encoded in the PointCloud to infer the traversability score of each edge of the navigation graph. To apply a neural network to 3D points, we require an architecture that can handle the unordered and irregular nature of the Point Cloud Data (PCD), and is thus permutation and translation invariant. Several possible architectures exist to achieve this, and we draw on the work of [2] to examine the use of Dynamic Graph Convolutional Neural Networks (DGCNNs) [3]. DGCNNs are designed to work with graph-structured data, such as point cloud data, and can handle graphs with variable numbers of nodes and edges.

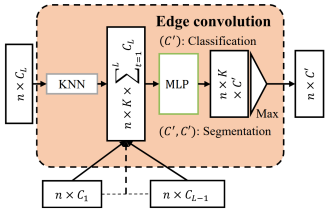
The DGCNN works by applying convolutional operations to the graph-structured data, allowing it to learn a latent representation of the edges that encodes the information of the PCD. The encoder part of the network is shown in Figure 3.3, the edge convolution operation (figure 3.3c) is channel-wise and aggregates edge features of all the connected edges; the spatial transformation (figure 3.3b) aligns the the input point set to a canonical space by applying an estimated 3x3 matrix. The latent representation, together with the start and goal poses, is fed through




(a) Network structure




(b) Spatial transformation




(c) Edge convolution operation

Figure 3.3: Full encoder architercture

several fully connected layers to predict the cost of traversing each edge in the navigation graph. For traversability estimation, we use a binary cross entropy loss (BCE) as the regression target is a probability.
Another possible architecture that we explored is [30] where the authors suggest that by removing the transformation network and linking the hierarchical features from different layers a better performance can be achieved, however we have concluded that in our case the differences between the two architectures are negligible.
Finally we also experimented with the usage of a Kernel Point Convolution [31], both with the rigid and deformable flavors, which operates directly on points in Euclidean space rather than converting them into graph structures. The weights of KPConv are located in space by kernel points and applied to the input points near them, in theory, this allows KPConv to have more flexibility and to be extended to deformable convolutions that can adapt to local geometry.
We argued that KPConv might be superior to Deep Graph Convolutional Neural Networks (DGCNN) because it operates directly on the points in space, instead of converting them into graph structures, hence allowing it to be more flexible and adaptable to local geometry, such as complicated obstacles in the environment. However, our results showed that the performance of KPConv was inferior to that of DGCNN. In Appendix A.2.2 we show the comparison between the different architectures; and in Appendix A.1 we show the parameters used for the final architecture.
Finally, we have concluded that DGCNN is the best suited architecture for our work and the next results will refer to this architecture.

## 3.2 Experiments

### 3.2.1 Datasets

In our work, we utilized both real-world and synthetic datasets to train and validate our system. The real datasets were mainly used for validation, providing us with a way to measure the performance of our system in real-world scenarios. In Figure 3.4 and 3.5 we show the mesh of the 2 real datasets we used in our work

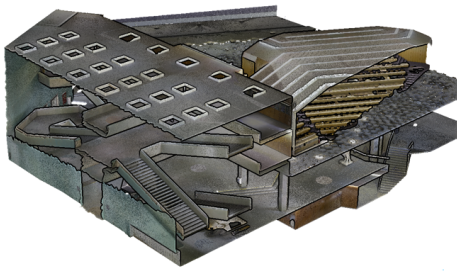


Figure 3.4: HPH Dataset

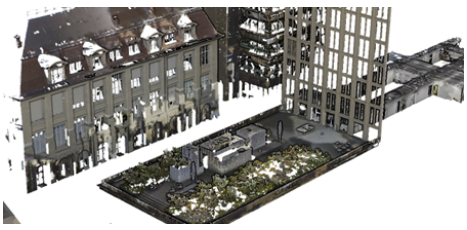
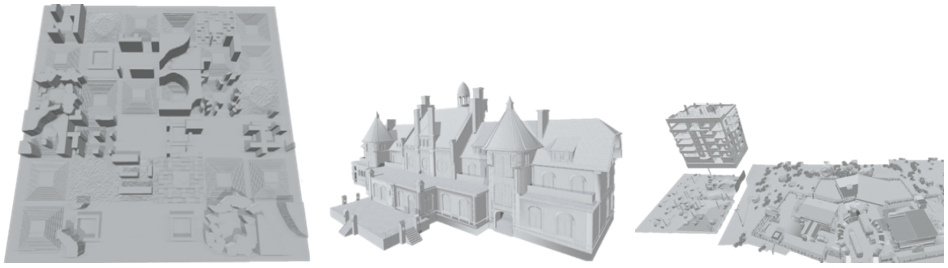


Figure 3.5: LEE Dataset

On the other hand, the synthetic datasets were used for training, allowing us to generate large amounts of data and to train our system in a controlled environment, with watertight meshes. Among the synthetic datasets, we used a variety of sources, including Unity models of virtual environments, randomly generated worlds, and mixtures of Unity models and randomly generated worlds. The use of Unity models allowed us to leverage the realistic and diverse environments available in video games, while the randomly generated worlds provided a more controlled environment that allowed us to examine the behavior of our system in more detail. Moreover the randomly generated world are also used a way to quickly prototype

and train a certain configuration as they provide a way to incorporate different kind of terrains in a small environment. The mixture of Unity models and randomly generated worlds allowed us to incorporate the benefits of both types of datasets, resulting in a system that is well-suited to real-world scenarios, hence our best results were obtained by training the network on both these kind of datasets. In Figure 3.6 we show an example of a randomly generated dataset (3.6a), a Unity model (3.6b) and a mixed dataset (3.6c).



(a) Randomly generated dataset (b) Unity model (c) Mix of random datasets and video game maps

Figure 3.6: Synthetic Datasets

### 3.2.2 Results

In this section, we present the results of our experiments and evaluate the performance of our proposed system. To measure the performance of our system, we use two metrics, the $L1$ loss and the $R^2$ metric. The $L1$ loss measures the mean absolute error between the ground truth and the predicted traversability scores, and it provides a straightforward interpretation of the model's performance, as it is a measure of the difference between the true and predicted values. The $R^2$ metric, on the other hand, provides a measure of how well the model fits the data, by measuring the proportion of the variance in the dependent variable that is predictable from the independent variable. A value of $R^2$ close to 1 indicates that the model fits the data well, while a value close to 0 indicates that the model does not explain the variability in the data.
In our experiment, we also visually demonstrate the performance of the network by displaying the difference in the edge score between the network's prediction and the ground truth. To do this, we compute the "navigation graph delta" which represents the difference in traversability score between the predicted edge score by the network and the actual ground truth edge score. By visualizing the navigation graph delta, we can easily identify the areas where the network is making correct predictions and where it is struggling.
In Figure 3.7 we show the results on the LEE Dataset[1] and is clear how the architecture is able to predict correctly almost all the edges.
In Table 3.1, we present the performance of the network on various combinations of training and evaluation environments. As demonstrated, the closer the training environment resembles a realistic scenario, the better the network performs (hence the improvement from the randomly generated world to the mixed one, where we also used Unity model in the training environment). Moreover, training the network with a more accurate Ground Truth (GT), obtained by running the simulation for a longer duration, results in even better performance. This is a crucial finding, as

[1] In the Appendix A.1 we will show the result from a poorer version of our network and highlight the limitations of the approach when there are discrepancies between the training and inference domains.

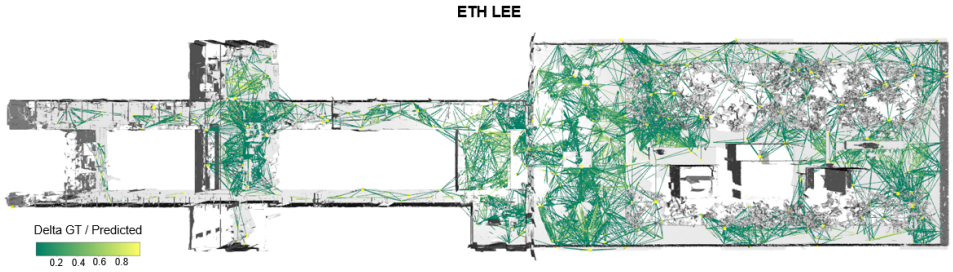



Figure 3.7: LEE Navigation Graph Delta

| Inference Environment | Metric | Training Environment | | |
|---|---|---|---|---|
| | | Rand Gen | Mixed 3 Iter | Mixed 10 Iter |
| ETH HPH | $L1$ | 0.17 | 0.15 | **0.14** |
| | $R^2$ | 0.63 | 0.70 | **0.72** |
| ETH LEE | $L1$ | 0.18 | 0.13 | **0.11** |
| | $R^2$ | 0.58 | 0.75 | **0.79** |

Table 3.1: L1 loss and $R^2$ for different combinations of training environments

highlighted in Figure 3.8, the network requires the same amount of computation time for training and evaluation, whereas the simulation approach requires a linear increase in computational time as the precision of the simulation increases. This highlights the efficiency of our approach in terms of computational cost.

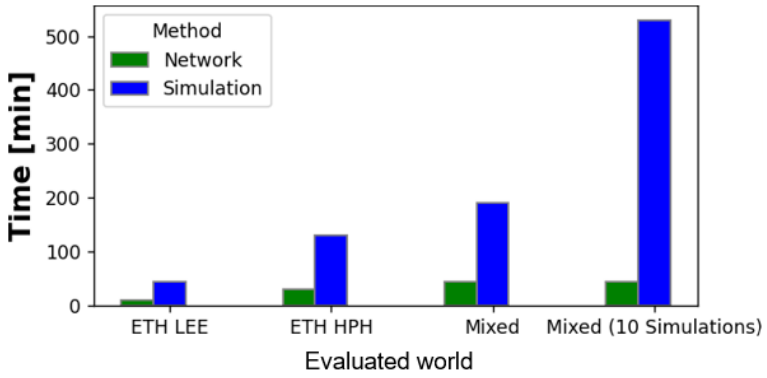



Figure 3.8: Evaluation Times

### 3.2.3 Input Randomization

Now we will delve a bit deeper on the robustness of our algorithm by evaluating the performance on different kind of perturbations to the PCD. These experiments are designed to test the ability of our model to handle noise and other variations that are commonly found in real world PCD and demonstrate the adaptability to real world scenarios.

#### Noisy Data

In Table 3.9 we assess the performance of our network when trained with a perfect PCD and evaluated on gradually noisier datasets, the noise we are injecting in the

poincloud is a GRV distributed as $X \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = 0$ and $\sigma$ is measured in meters. It is evident that once the noise surpasses a certain threshold ($\sigma \simeq 0.3m$), the network performance starts to drop rapidly. This is obvious if we consider that the network has only been trained on "perfect" PCD's and has never seen a noisy sample. However, we also demonstrate that by injecting random noise into the training data, the network can adapt to this variability and generalize much better to unseen noisy samples during inference. The results presented in Table 3.10 show that the network can handle a wide range of noise levels and perform consistently well even when evaluated on noisy PCD. This highlights the adaptability of our network to real-world scenarios, where the point cloud data may be subject to various sources of noise and variability.

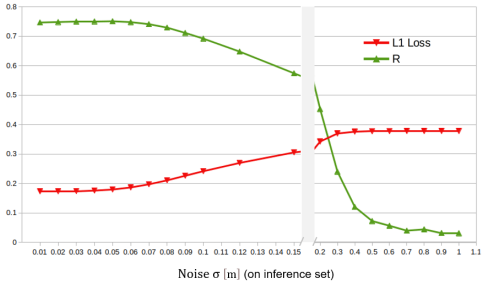



Figure 3.9: No Noise in Training

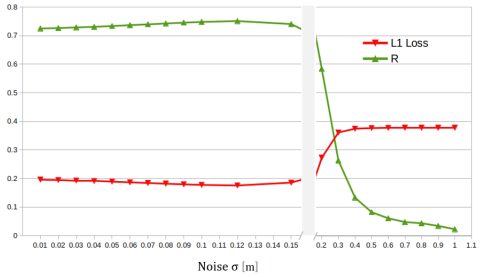



Figure 3.10: $\mathcal{N}(0,\ 0.1^2)$ Noise in Training

### PCD Resolution

In addition to testing the robustness of our algorithm to different types of perturbations, we also conducted experiments to evaluate the resilience of our model to different amounts of point dropout. Point dropout refers to the removal of a certain percentage of points from the point cloud data, simulating scenarios where some data may be missing or unavailable. Our results show that our algorithm is able to maintain a high level of performance even when faced with significant amounts of point dropout, further demonstrating its adaptability and resilience to real-world data, this is mainly due to the convolutional nature of the DGCNN which is also able to operate on very sparse data. In Figure 3.11 we show the effect of varying the resolution of the pointcloud (I.E. the spacing between the points); in Figure 3.12 we show the effects of randomly removing a percentage of points. Both this result show that the network can adapt to a wide range to different resolutions, also when very different from the one it has been trained with.

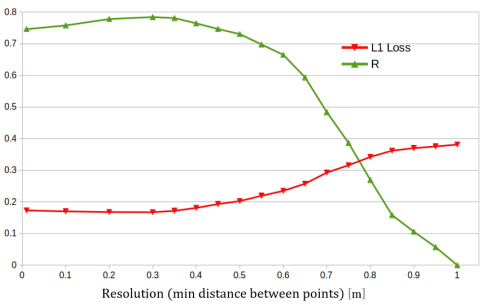



Figure 3.11: Resolution

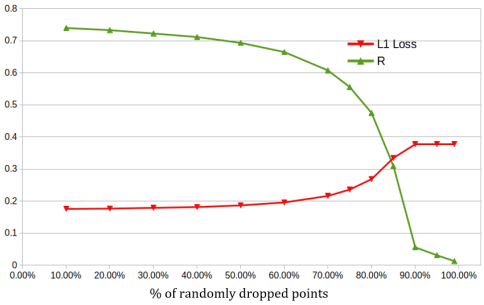



Figure 3.12: Random Point Dropout

### 3.2.4 CoT Metric

In this section, we will delve into the evaluation of the Cost of Transport (CoT) metric. CoT [4] is defined as the ratio of the energy expenditure to the transported weight and it provides a measure of the efficiency of the locomotion:

$$CoT_{mech} = \sum_{j=1}^{n} \sum_{i=1}^{4} max(\tau_{ij} \cdot \omega_{ij}, 0)/(n \cdot m \cdot g \cdot v_{avg}) \tag{3.1}$$

Where $n$ is the number of joints per leg, $\tau_{ij}$ and $\omega_{ij}$ are the torque and the angular velocity of a single joint, m is the mass of the robot and g the gravitational acceleration. In our work, we demonstrate that our algorithm is able to accurately learn the CoT metric and predict it in a variety of environments. In Figure 3.13 we compare our results with ground truth values obtained from simulating the robot's motion and report the error in the CoT prediction, showing how our model is able to predict very well the real energy used by the robot. For this task we just had to change the loss the network was using from BCE to a Huber Loss since the target now is not anymore a probability; the rest of the architecture remained exactly the same as for the traversability estimation. We are able to effectively train the network on this different task. Our results show that the network is able to generalize well and produce accurate CoT predictions in different environments; proving that the approach we used is generalizable to diverse regression targets and can potentially tackle diverse tasks. In A.2.4 we also show numerical results for this task.

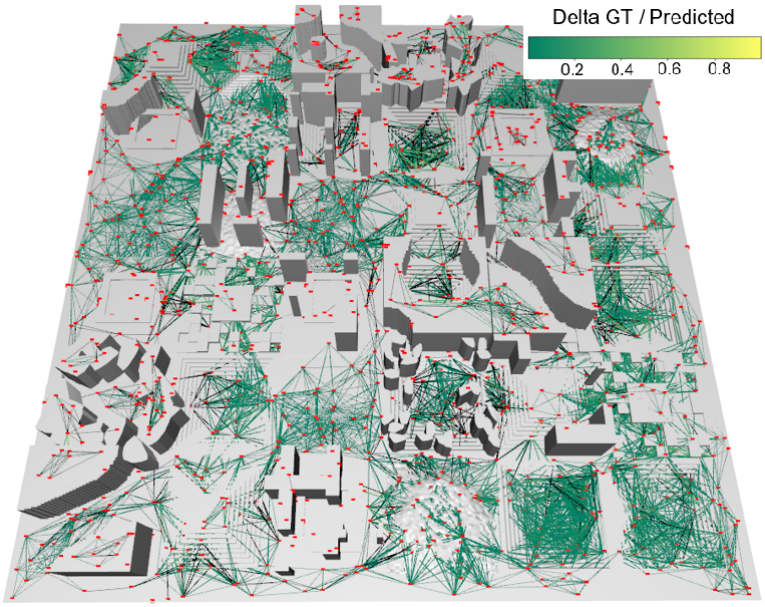



Figure 3.13: CoT Comparison: The error between the GT and the real data is expressed in percentage, a 0.5 means the predicted/real energy expenditure differ by 50%.

# Chapter 4

# Learning Point Sampling

In the previous works we have shown such as [2] and [25], the process of constructing a Probabilistic RoadMap (PRM) graph for navigation purposes has proven to be computationally expensive and inefficient, relying on rejection sampling methods. The method is based on generating points randomly in the environment and then checking if they are feasible for the robot to reach. This process is repeated until a sufficient number of feasible points has been generated. The main problem with this approach is that it requires extensive computation time, making it infeasible for real-time applications, such as online navigation for robots, because it creates a bottleneck in terms of speed and adaptability. Given the challenges posed by this process, it is important to explore alternative methods for PRM-graph generation. In this section, we aim to demonstrate the potential of our proposed architecture to tackle this issue. Specifically, we show that our network can also learn to generate new points for the navigation graph in an efficient and accurate manner.

## 4.1 Architecture

Starting from the work of [5] and [6], we use the encoder of our network (Figure 3.3), which is trained to generate a latent representation of the pointcloud, to generate a map representation of the pointcloud. This map representation is then used as input to a conditional variational autoencoder (CVAE) which can generate new samples for the PRM graph building. A draft of the general idea behind the architecture is shown in figure 4.1.

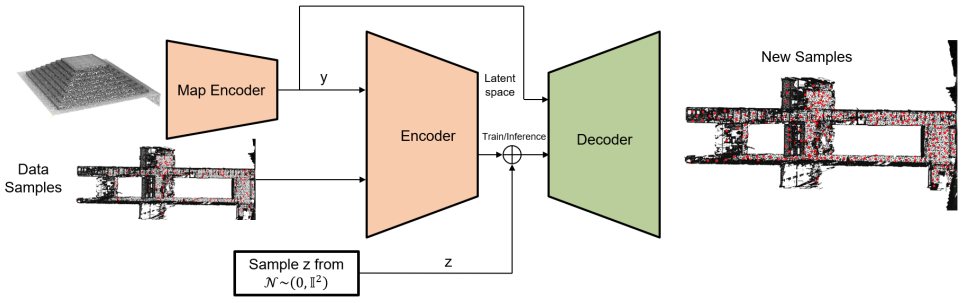



Figure 4.1: Sampling Network Idea

The key idea behind the CVAE is to combine the strengths of VAE and cGAN by using the encoder of a VAE to learn a probabilistic mapping between the data space and a latent space, and the input information to condition the generation

process. This allows the model to generate new samples that are similar to the input information and also have a compact representation in the latent space.
More specifically, the model, when used to generate graphs on unseen maps, will learn how to generate samples from a GRV conditioned on the specific portion of the PCD map. In figure 4.2 we show how the complete pipeline works, it only requires a PCD input (the digital twin of the environment) and firstly it outputs a PRM graph, afterwards, it passes the PRM graph trough the DGCNN network (see 3.1.2) to re-sample the points centered around each edge and estimates the edge costs.

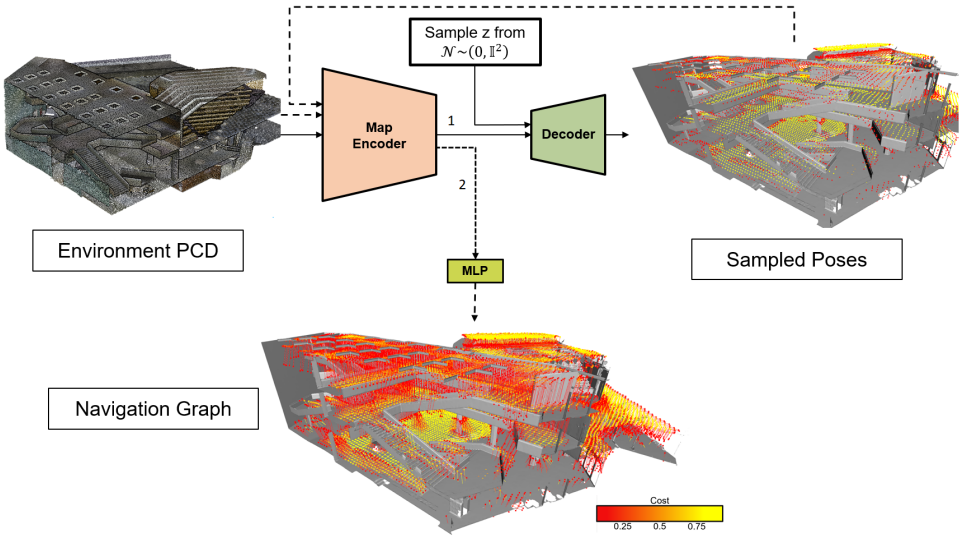



Figure 4.2: Final Pipeline

### 4.1.1 Sample Generation

To obtain the Ground Truth data for training, we sampled multiple points on the digital twin mesh and simulated the robot in each pose. By running the simulation multiple times, we obtained a probability that reflects the feasibility of the robot to stand in that pose. However, this data cannot yet be used to train the CVAE as all the samples it was trained on for edge cost regression had a fixed small size (as described in Section 3.1.1). To address this, we applied a sliding window on the PCD and sampled the points and poses inside each portion of the space. This generated new training data for each pose, which could then be used to train our CVAE. Moreover, to add diversity in the coordinates of the sampled points, each one of the samples is randomly shifted and rotated so that the network has a more diverse dataset to train with. Figure 4.3 illustrates this process.

### 4.1.2 Neural Network Architecture

Our approach, as we anticipated, employs a conditional variational autoencoder (CVAE) to predict the 6-degree-of-freedom (6DoF) of a 3D pose. Our design is largely inspired by the approach in [5], with modifications to accommodate the 6D pose space. To address the inherent rotational symmetry of this pose space, we employ a Geodesic Loss as the evaluation metric for our model. The Geodesic Loss measures the distance between the predicted and ground truth poses by computing the length of the shortest path on a Riemannian manifold, providing a more accurate and resilient comparison than a simple mean squared error (MSE) Loss.

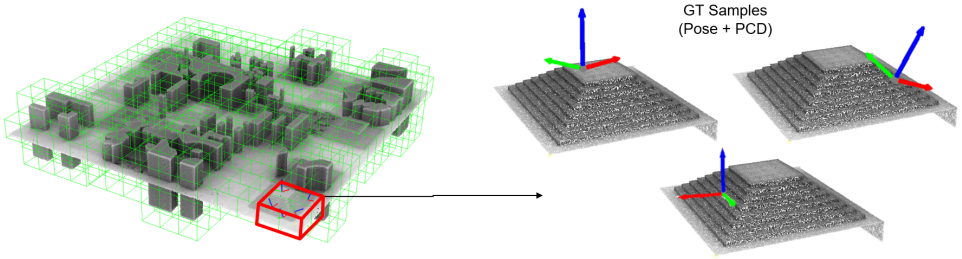



Figure 4.3: Data Generation

A significant challenge in designing our neural network architecture was how to handle samples with no feasible poses for the robot. To address this issue, two alternative network architectures were proposed and depicted in figures 4.4 and 4.5. In the first architecture, a multi-layer perceptron (MLP) was added after the DGCNN net to classify the presence of feasible poses in the pointcloud. However, this approach proved ineffective as the task of determining the presence of feasible poses in a pointcloud sample is inherently ambiguous, leading to poor network training results.

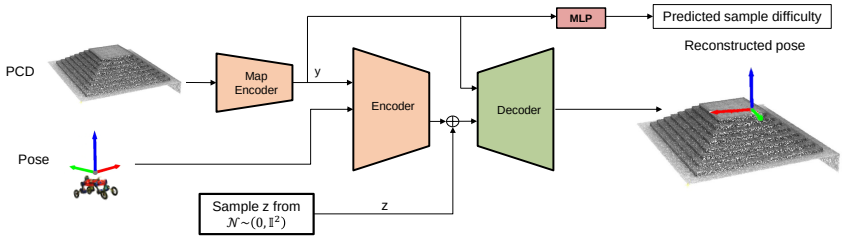



Figure 4.4: Sampling Architecture 1.

In the second architecture, an MLP was attached after the decoder network. The MLP was tasked with determining the confidence of the network in each predicted or reconstructed pose (in contrast to what we did in the previous approach, where we classified the entire sampled PCD, here the score is given to a single sampled pose), allowing us to directly prune all the sampled edges with confidence scores below a predetermined threshold, leading to improved performance. To evaluate the confidence of each sampled pose we treated it as a classification problem and used a BCE loss, the ground truth was obtained by simulating the robot standing in each pose multiple times, much similar to how we treated the traversability score in section 3. During training, this approach demonstrated promising results and was ultimately chosen as the final architecture.

## 4.2 Results

The performance of our proposed neural network was evaluated on various datasets. In figure 4.6, we present the results on a randomly generated dataset. The edges are color-coded based on their traversability, while the points are color-coded based on the confidence score assigned to them by the network. The results demonstrate that the network effectively learns to prioritize points that are farther away from obstacles and assigns lower scores to points that are more likely to be problematic to reach (such as stairs or tight corridors). Since the results are generated from

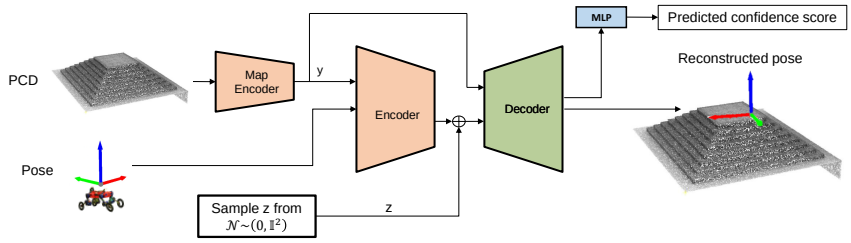



Figure 4.5: Sampling Architecture 2.

the network from scratch they inherently cannot be compared with a ground truth navigation graph (as we did for the DGCNN evaluation in section 3.2.2) and we had to heuristically verify the performance of the generated navigation graphs by visually inspecting the results.

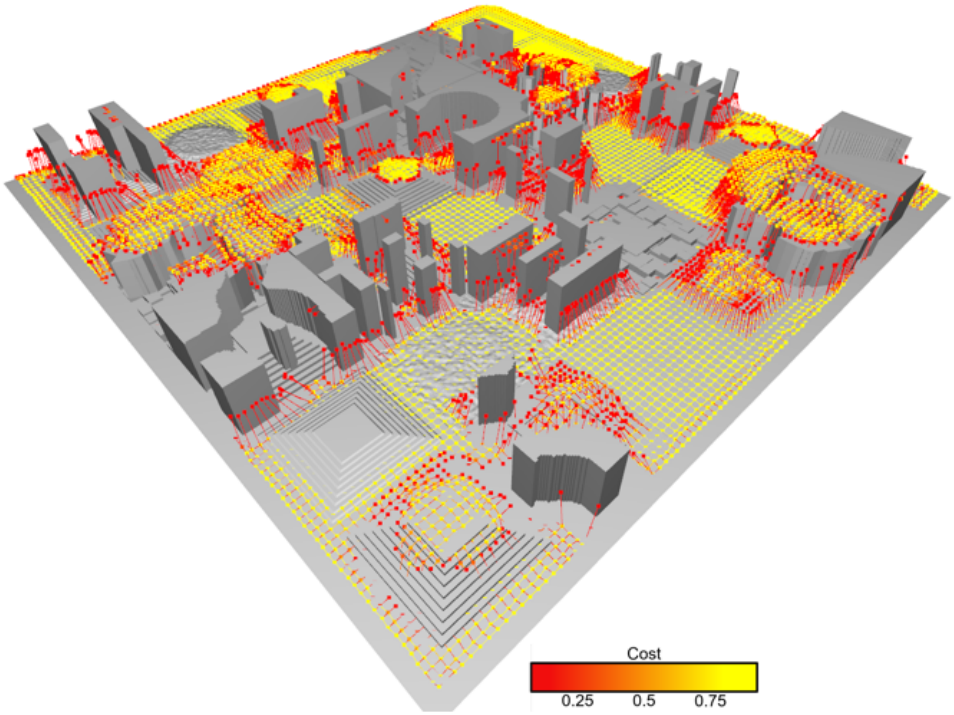



Figure 4.6: Generated Navigation Graph on Randomly Generated Environment

The method is still not mature and often performs poorly, especially in novel environments (with different topologies and structures compared to the training set). In Section 5.1 we highlight the limitations and present some ideas on how the work can be improved in future iterations.
In conclusion, our proposed conditional variational autoencoder (CVAE) leverages the map representation as a condition, enabling the generation of diverse and representative samples of the environment. This capability helps to overcome the computational constrains inherent in rejection sampling and is able to generate new poses orders of magnitude faster. Moreover, the CVAE can also be applied to generate samples for unseen environments, thereby rendering our approach more generalizable and applicable to real-time tasks.

# Chapter 5

# Conclusion

This work presents a novel approach for robot motion planning that combines an edge cost regression network and a conditional variational autoencoder for efficient and effective sample generation. The edge cost network utilizes a DGCNN to extract features from pointclouds, and trains to predict the feasibility of robot motion along specific edges of a navigation graph. The CVAE, on the other hand, biases the sampling towards regions of the space where an optimal solution is more likely to be found, overcoming the limitations of rejection sampling methods.
The edge cost regression network demonstrates a marked advantage in terms of efficiency over simulation-based approaches, as well as robustness to noisy and perturbed data. Furthermore, its ability to adapt to different regression targets highlights its versatility and generizability, making it well-suited for a wide range of applications in various domains.
The proposed sampling network architecture addresses the challenge of generating a PRM graph directly from a PCD and is also able to handle samples with no feasible poses, by predicting the confidence of the generated pose and allows for direct pruning of low confidence poses, leading to improved performance compared to rejection sampling methods.

## 5.1 Limitations and Future Work

Regarding the edge regression network, the failure cases are mainly caused by discrepancies between the training and the evaluation datasets (see Section A.1). It is important to train on a dataset that resembles the most a real environment (such as the mixed dataset we showed in Figure 3.6); we showed that the method is resilient to changes and disturbances on the Point Cloud, and hence it can potentially work with real time data captured by the robot: A future step would surely be to evaluate the performance in such a situation. For the sampling network, we showed how the network is able to correctly learn how to sample points, however the generated PRM graphs look still a bit rigid and don't seem to adapt well to the features of the environment. A thorough investigation has to be conducted to show the applicability of this approach, for instance using different parameters for the CVAE or tuning better the hyper-parametres. It is also interesting to see the performance from using different generating networks, such as generative adversarial networks [32]. Finally, an important step would be to streamline the whole architecture and have it running on-line on the real robot, especially to prove the advantages in terms of computational time and to showcase the potentiality to use it in unknown environments and in real time (for instance on a SLAM map acquired by the robot's sensors).

In conclusion, our approach provides a promising solution to the problem of robot motion planning in dynamic and uncertain environments. The combination of an edge cost regression network and a CVAE-based sample generator shows significant improvements in terms of efficiency, effectiveness and generalization compared to traditional methods. We believe that this work opens up new avenues for research in robot motion planning, and lays the foundation for further advances in this field.

# Bibliography


[1] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] A. Unagar, "Global path planning in a digital-twin," 2021.

[3] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," 2018.

[4] M. Bjelonic, C. Dario Bellicoso, M. Efe Tiryaki, and M. Hutter, "Skating with a force controlled quadrupedal robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7555–7561.

[5] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," 2017.

[6] L. Schmid, C. Ni, Y. Zhong, R. Siegwart, and O. Andersson, "Fast and compute-efficient sampling-based local exploration planning via distribution learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7810–7817, 2022.

[7] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, p. 560–570, oct 1979.

[8] J. Kim, M. Kim, and D. Kim, "Variants of the quantized visibility graph for efficient path planning," *Advanced Robotics*, vol. 25, no. 18, pp. 2341–2360, 2011.

[9] B. Zhu, C. Li, L. Song, Y. Song, and Y. Li, "A algorithm of global path planning based on the grid map and v-graph environmental model for the mobile robot," 10 2017, pp. 4973–4977.

[10] Jang Gyu Lee and H. Chung, "Global path planning for mobile robot with grid-type world model," *Robotics and Computer-Integrated Manufacturing*, vol. 11, no. 1, pp. 13–21, 1994.

[11] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[12] Y. Wang and G. S. Chirikjian, "A new potential field method for robot path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 977–982.

[13] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, pp. 207–222, 2002.

[14] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer, "Modified artificial potential field method for online path planning applications," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 180–185.

[15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[16] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto, "An efficient sampling-based method for online informative path planning in unknown environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1500–1507, 2020.

[17] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[18] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, pp. 273–290, 2012.

[19] L. Guo, Q. Yang, and W. Yan, "Intelligent path planning for automated guided vehicles system based on topological map," in *2012 IEEE conference on control, systems & industrial informatics*. IEEE, 2012, pp. 69–74.

[20] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[21] L. Blackmore, H. Li, and B. Williams, "A probabilistic approach to optimal robust path planning with obstacles," in *2006 American Control Conference*. IEEE, 2006, pp. 7–pp.

[22] T. Klamt and S. Behnke, "Towards learning abstract representations for locomotion planning in high-dimensional state spaces," *CoRR*, vol. abs/1903.02308, 2019.

[23] J. Guzzi, R. O. Chavez-Garcia, M. Nava, L. M. Gambardella, and A. Giusti, "Path planning with local motion estimations," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2586–2593, 2020.

[24] L. Wellhausen and M. Hutter, "Rough terrain navigation for legged robots using reachability planning and template learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6914–6921.

[25] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," 2017.

[26] M. Brandão, O. B. Aladag, and I. Havoutis, "Gaitmesh: Controller-aware navigation meshes for long-range legged locomotion planning in multi-layered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3596–3603, 2020.

[27] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016.

[29] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *CoRR*, vol. abs/1706.02413, 2017.

[30] K. Zhang, M. Hao, J. Wang, C. W. de Silva, and C. Fu, "Linked dynamic graph cnn: Learning on point cloud via linking hierarchical features," 2019.

[31] H. Thomas, C. R. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," *CoRR*, vol. abs/1904.08889, 2019.

[32] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.

# Appendix A

## A.1 Training Parameters

In this section we will illustratrate the parameters of our networks that led to the results showed in 3.2.2 and 4.2

### A.1.1 Edge Cost Network

| | **Parameter** | **Value** |
|---|---|---|
| Hyperparams | Learning rate | 0.001 |
| | Weight decay | 0.0001 |
| | Dropout | 0.5 |
| | Epochs | 12 |
| | Batch size | 8 |
| | Optimizer | Adam |
| DGCNN | Input Points | 1024 |
| | Latent Repres Dims | 2048 |
| | K-nn Neighbours | 60 |
| | PCD dim | 2 metres |

Table A.1: Hyperparameters for the DGCNN network

### A.1.2 Sampling Network

The hyperparemeters are the same as for the DGCNN network, we just resorted to a smaller map representation to have a more efficient training and to avoid overfitting

| | **Parameter** | **Value** |
|---|---|---|
| CVAE | Latent Repres Dim | 512 |
| | Encoder In Dim | 7[1]+ 512 |
| | Encoder Out Dim | 35 |
| | Decoder In Dim | 35 + 512 |
| | Decoder Out Dim | 7[1] + 1 |
| DGCNN | Input Points | 1024 |
| | Latent Repres Dims | 512 |
| | K-nn Neighbours | 60 |
| | PCD dim | 2 metres |

Table A.2: Hyperparameters for the CVAE network

[1] 3 if we only sample the position and not the pose

## A.2 Results - Extra

### A.2.1 Training World

In fig A.1 we show the result with a network trained on a randomly generated dataset (3.6), it is evident how the network fails to predict correctly the traversability scores on the edges in the center of the image; this because if we have a look at real dataset we see that there are ping-pong tables there, a kind of structure never seen before by the network. It is important then to use always a dataset that contains all the kind of structures that might be present in the environment in which we want to deploy the robot, not to have a Domain Gap between the training and inference environment.

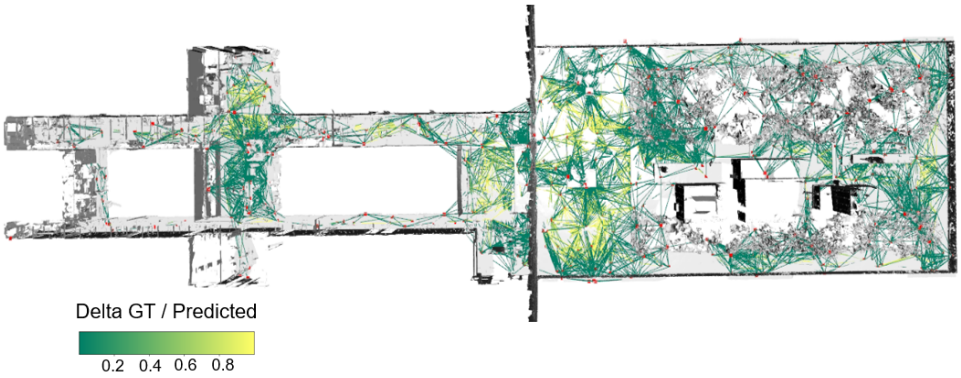



Figure A.1: LEE evaluation on randomly generated dataset

### A.2.2 Edge Cost Network

In the next table we show the results from using different encoder network architecture, it is evident that the DGCNN we chose was clearly superior in all of the metrics.

| **Architecture** | **Metric** | **Inference Environment** | |
|---|---|---|---|
| | | ETH HPH | ETH LEE |
| DGCNN | $L1$ | 0.14 | 0.11 |
| | $R^2$ | 0.72 | 0.79 |
| LDGCNN | $L1$ | 0.23 | 0.20 |
| | $R^2$ | 0.48 | 0.44 |
| KP Conv | $L1$ | -[2] | 0.46 |
| | $R^2$ | -[2] | 0.30 |

Table A.3: Performance of different Map Encoder Architectures

### A.2.3 Voxel Dimensions

Here we show the results from training with different dimentions for the sampled points around the navigation graph edges, it is clear that a distance of $\approx 2.0m$ is the clear optimum which encodes enough information of the surrounding environment without including unnecessary (IE too far away) obstacles.

[2]Environment not evaluated for this architecture

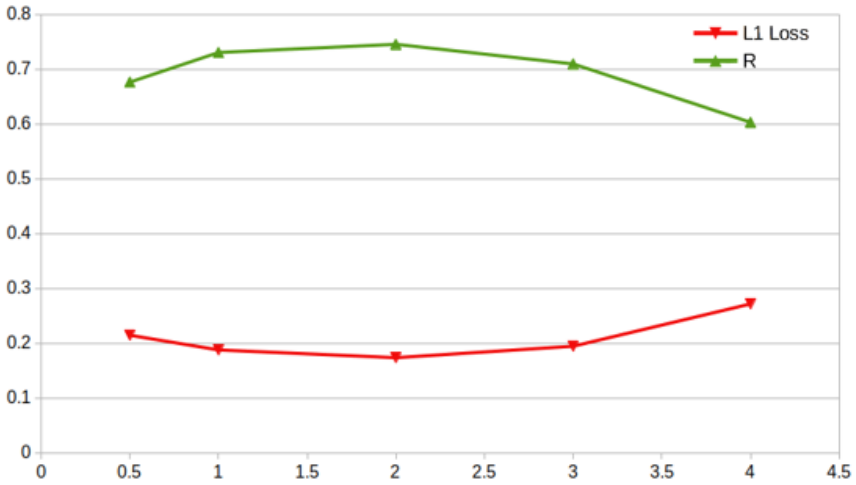



Figure A.2: Effect of different voxel dimensions (on randomly generated dataset)

### A.2.4 CoT

Here we highlight the numerical results for the CoT cost regression, all the results were obtained from a network trained on the mixed dataset.

| | | **Inference Environment** | | |
|---|---|---|---|---|
| | | ETH HPH | ETH LEE | Mixed[3] |
| **Metric** | $L1$ | 0.004 | 0.005 | 0.003 |
| | $R^2$ | 0.19 | 0.23 | 0.53 |

Table A.4: Performance of the CoT metric on different datasets

[3]Evaluated only on the test split of the dataset

## A.3 Navigation Graphs

### A.3.1 DGCNN Edge Traversability

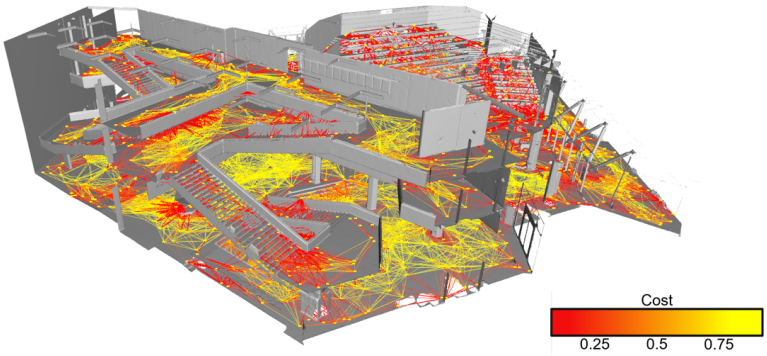



Figure A.3: Complete Navigation Graph HPH

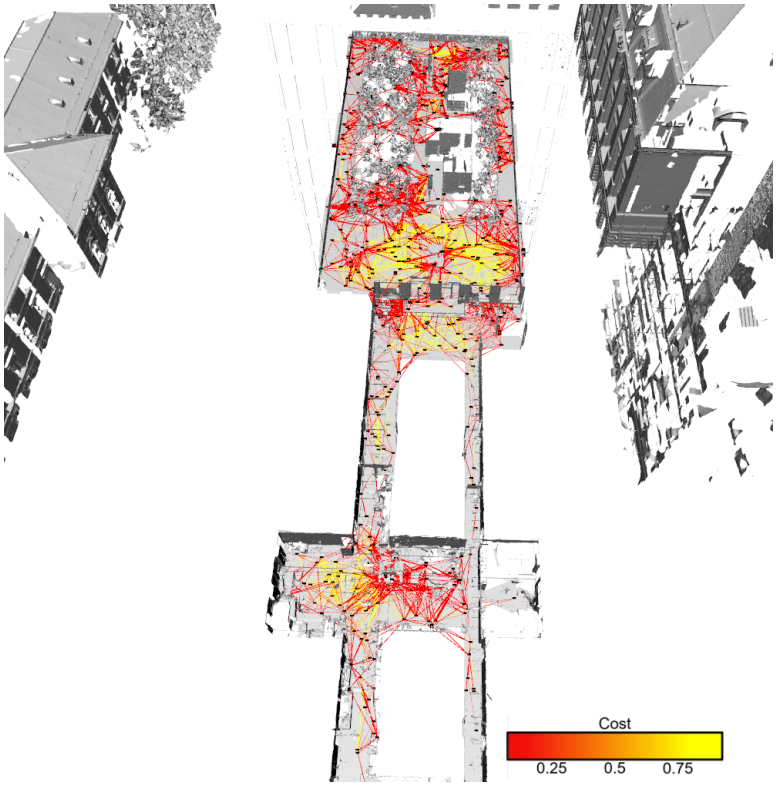



Figure A.4: Complete Navigation Graph LEE

### A.3.2 CoT

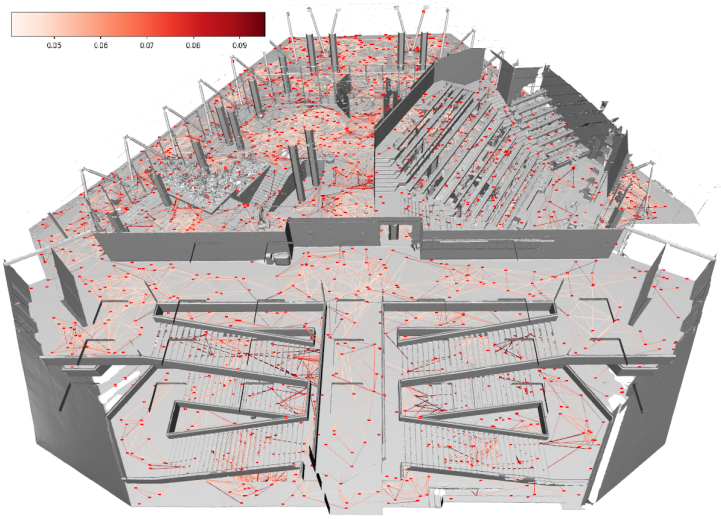


Figure A.5: Complete CoT Graph HPH

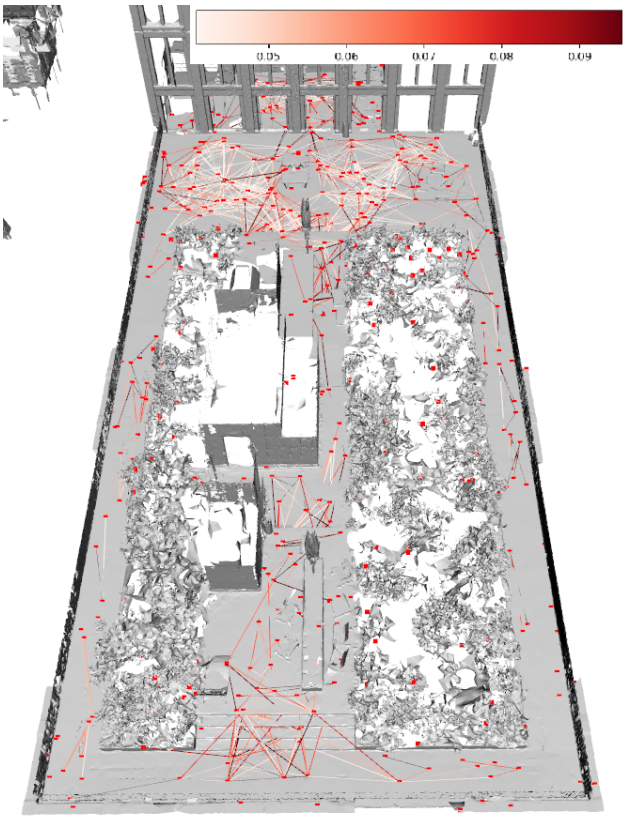



Figure A.6: Complete CoT Graph LEE

### A.3.3 Sampling Architecture

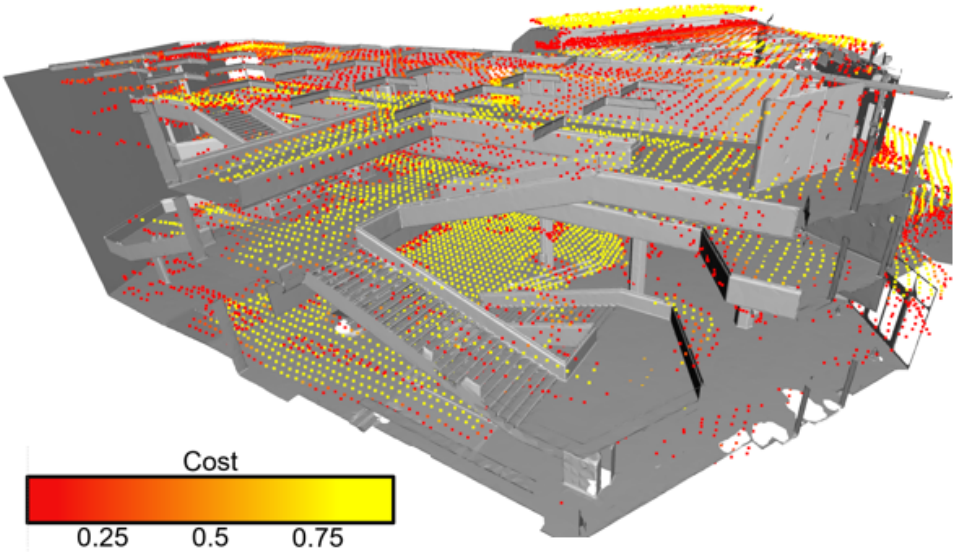



Figure A.7: Example of the sampled poses for the HPH dataset

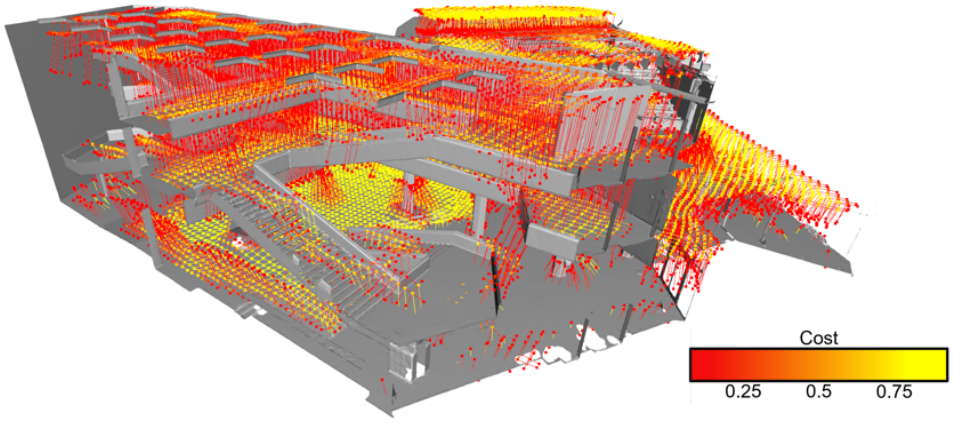



Figure A.8: Sampled Navigation Graph HPH

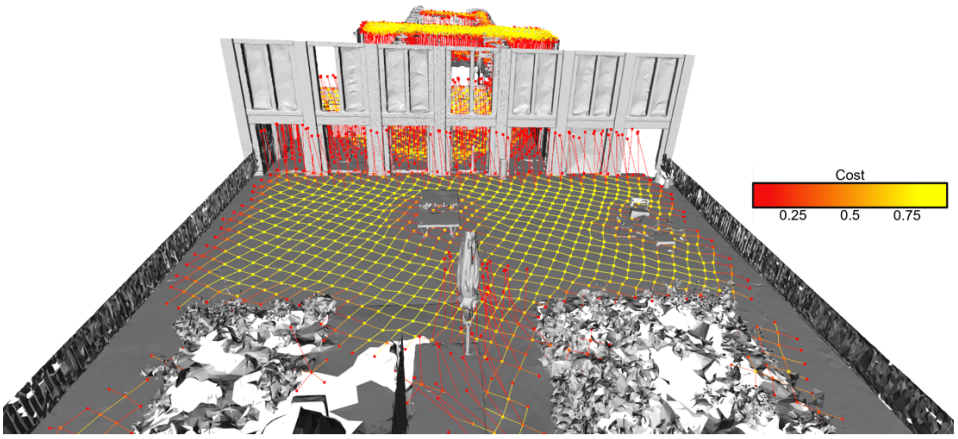



Figure A.9: Sampled Navigation Graph LEE