Imperial College London

BEng Individual Project

Imperial College London

Department of Computing

Unsupervised Path Regression Networks

Author:
Michal Pándy

Supervisor:
Dr. Ronald Clark

Second Marker:
Dr. Amir Alansary

June 15, 2020

Abstract

Path planning problems are conventionally solved in an iterative manner. Recent iterative learning methods have greatly improved inference time, but require ground truth paths for training, which are costly to compute. In this work, we demonstrate that complex shortest path problems can be solved via direct regression learned in an unsupervised manner, making the training pipeline simple and scalable. Key to our approach is a novel cost function which only requires a scene and a planning problem specification, and whose minima guarantee collision-free solutions. We show that this cost function is capable of attaining high-quality paths, and experimental results further demonstrate that our method outperforms the objectives of optimisation-based planners, beats state-of-the-art supervised learning baselines for shortest path planning, and is capable of planning in domains with partial observability.

Acknowledgements

I would like to thank **Dr. Ronald Clark** and **Daniel Lenton** for our fruitful meetings and discussions, which resulted in countless ideas for tackling the problems presented in this work.

I would also like to thank my parents **Ladislav** and **Alicja** for their restless support throughout all of my eduction.

Lastly, I want to thank **Maximilian**, **Alex**, and **Emmie** for their unconditional love.

Contents

List of Figures

Chapter 1

Introduction

Spatial navigation is a longstanding and broad problem in embodied AI and robotics, with a diverse set of employable methods. Self-driving cars[13], underwater vehicles[14], mobile robots[15], drones[16], and manipulators[17] all share the task of having to optimally navigate in uncertain environments. The general importance and applicability of path planning elevated it to a well-studied area with thorough reviews and surveys[18, 19, 20].

While there has been extensive research to address aspects of the path planning problem, classical methods often suffer performance penalties due to its computational complexity and multifaceted nature. Apart from having to ensure that agents navigate safely in their environments, planning algorithms also aim to minimise the lengths of the travelled paths, satisfy various kinodynamic constraints, generalise to dynamic and partially observable environments, and operate in higher dimensional task spaces. The fulfillment of all of these requirements leads to complex algorithms with multiple computationally expensive stages. Recent research in deep learning, however, suggests that solving certain areas of path planning can be achieved by deploying a deep neural network, and this way bringing significant speed-ups in inference time. Among others, neural networks have already been shown to be exceptional at reconstructing scene geometries[21], performing two-dimensional path planning[10], and producing high-quality paths in combination with classical planners[22]. These recent breakthroughs point at the usefulness of deep learning in a planning context, which motivates the goal of this dissertation.

In this work, we propose to train a neural network that *directly regresses an entire path* from start to goal in a single network forward pass, by minimising a *novel cost function* dependent only on the scene itself. This cost function is similar in form to those in the gradient-based planning literature[23], but unlike these methods, we derive a formulation whose minima guarantee collision-free solutions with minimal path length. This formulation enables our network to plan in diverse scenes, without the need of calibrating weighting parameters, as illustrated in Figure 1.1.

To demonstrate the general applicability of our method, we further train a variety of networks using our cost function, both on full-state scene descriptions and on depth images. We achieve

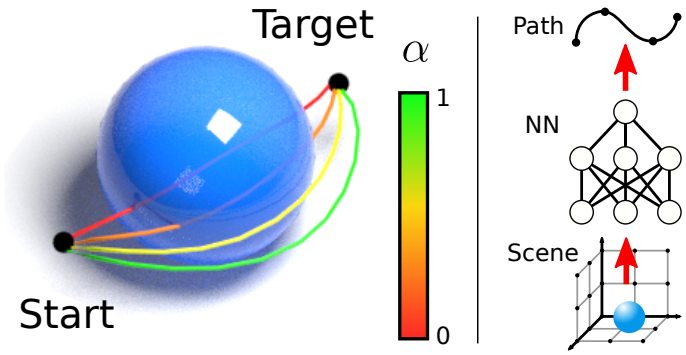


Figure 1.1: (left) Gradient-based planning methods use cost functions consisting of two heuristic terms, a path-length l term and a collision term c_{coll} combined using an arbitrary weighting, $c = l + \alpha c_{coll}$. In this work we design l and c_{coll} to guarantee collision-free paths at the optimum without requiring a calibrated weighting factor. We further propose to solve this optimisation problem using an efficient regression network (right).

state-of-the-art performance for learning-only approaches on benchmark tasks with a significant increase in inference speed. In addition, by using our proposed method, we remove the need to pre-compute an offline dataset, and so we are able to reduce *total* training time by almost two orders of magnitude compared to supervised approaches. To the best of our knowledge, this is the first work that demonstrates the use of an unsupervised cost function to learn complex path planning as a direct regression problem.

1.1 Objectives

In short, there are two main goals of this project. Firstly, we are aiming to train deep neural networks for path planning in an unsupervised manner, that is, without the requirement of ground truth optimal paths. Secondly, we want to formulate a parameterisation that will allow us to generate paths in a single network forward pass. Our objectives can thus be divided into the following areas:

Cost function formulation

The initial step in our work is to formulate a cost function that would have theoretical guarantees which would enable us to train path planning networks without ground truth path supervision. At this stage, we also want to investigate a neural network's capacity necessary to avoid static obstacles, while respecting reasonable path length constraints. The goal of this stage is to:

1. Encode soft and hard path prediction objectives such as length minimisation or collision avoidance in a cost function with theoretical guarantees
2. Devise sampling-based differentiable cost computation
3. Investigate path parameterisations that enable single forward pass regression planning
4. Formulate a general training procedure that can be used to train deep neural networks using our method in several planning contexts

Diverse planning evaluation

Building on the previous objective, in this section we aim to train a diverse set of planning networks for various contexts. Among others, this section aims to:

1. Compare our cost function to objectives present in the stochastic optimisation-based planning literature
2. Investigate and compare the ability of our method to plan in settings where full world state knowledge is present
3. Use our method to train a network capable of planning in partially observable contexts. That is, contexts where the network is conditioned on imperfect scene knowledge
4. Investigate the capability of our method to plan in domains where discrete planners perform well

1.2 Challenges

On a high level, this project was challenging because it proposes a novel approach to learning-based path planning. Hence, there were several unknowns, explored concepts, and negative results before we arrived at a formulation that performs well. There were also other specific hurdles we had to overcome to complete this project, the most significant ones being:

1. **No standard deep learning benchmark for path planning** - Numerous tasks in the AI community have robust associated benchmarks that can be used to compare methods. Unfortunately, since learning-based path planning is a relatively recent research direction, such benchmarks have not yet been established. In practice, this meant that we had to implement unique data loading pipelines and network conditioning for every method we wanted to compare against. Likewise, when we wanted to explore a new avenue, we had to generate our own dataset.

2. **Training related issues** - Some of our networks took a significant amount of time to train (up to 12 hours). Since we wanted to explore the application of our method in various contexts, this constraint reduced our iteration speed. Further, it was not uncommon for experiments to fail due to disc space issues or occasional power outages.

3. **Access to models** - Standard deep learning-based planning approaches often either do not provide their datasets or their trained models. During this project, we had to follow instructions in research papers to train path planning networks, or directly communicate with research groups to obtain their trained models. This hindered our progress, as we had to ensure that we create high integrity comparisons.

4. **Access to hardware** - Some of our networks were trained on high-resolution depth images, which generally require high-memory GPUs to train. Since our cost computation is sampling-based, we needed larger batch sizes to reduce noise in our gradient approximations. We had to implement workarounds such as storing and averaging gradients during training time to ensure our networks converge.

1.3 Contributions

1. **Cost formulation** - We devise a novel cost formulation that guarantees collision-free and shortest solutions in its minima, without the need for domain-specific calibration. We demonstrate the effectiveness of our cost function on benchmarks against standard objectives used in stochastic optimisation-based planning literature. Our formulation allows us to train neural networks for path planning in an unsupervised manner.

2. **Path parameterisation** - We propose a path parameterisation with sufficient expressive capacity to allow direct path regression planning.

3. **Continuous full-state planning** - Our cost formulation together with our parameterisation allows us to train a diverse set of networks for various planning tasks. We demonstrate the planning capabilities of our method by beating benchmarks of state-of-the-art learning-only continuous planners on complex 3D datasets.

4. **Planning from images** - Motivated by the fact that robots are, in practice, never equipped with oracle knowledge about the surrounding world, we show the versatility of our method by training an end-to-end depth to path regression network.

5. **Limitations demonstration** - We perform rigorous ablations to demonstrate the limitations of our method on maze-like environments.

1.4 Publication

The work presented in this dissertation was also summarised as a paper and submitted to the Thirty-Fourth Conference on Neural Information Processing Systems (NeurIPS). The purpose of the NeurIPS annual meeting is to foster the exchange of research on neural information processing systems in their biological, technological, mathematical, and theoretical aspects.

1.5 Report Layout

Chapter 2 and Chapter 3 explain some of the basic concepts necessary for understanding this work, and place it in the context of most recent learning-based path planning research. Chapter 2 sheds light on more advanced concepts in deep learning, B-spline interpolation, and signed distance functions, while Chapter 3 summarises important takeaways and innovation opportunities in the current landscape of learning-based planning.

Chapter 4 is concerned with thoroughly explaining our cost formulation and our parameterisation, which represent the slightly more theoretical aspects underpinning this work. Chapter 5 then extends this theory with empirical evidence in diverse planning domains. Finally, we conclude our work in Chapter 6 with promising directions for future research.

Chapter 2

Preliminaries

In this chapter, we review some of the key concepts necessary for understanding this work. In particular, we look at groundwork in deep learning, splines, and signed distance functions.

2.1 Deep learning

2.1.1 Neural networks

In simple terms, when people write computer programs, they define arbitrary functions $f(x) = y$. This function takes some data x, performs operations on the data via the function f, and returns a result y. Elementary examples of such functions could be sorting programs, where the input x is an unsorted array, f performs a sorting algorithm, and y is the final sorted array. A more complex example of f could be a classification algorithm, which takes an animal image x, applies various feature detection algorithms in f, and returns an animal label y. In this way, software engineers try to approximate a function f^*, which can be seen as the true, bug-free, perfect function achieving a specific task. In the case of array sorting, humans have done well to approximate f^*, however, problems such as image classification require different techniques to approximate f^* well. Indeed, for certain classes of problems, it has been shown that estimating the function f^* by analysing large datasets of possible input-output pairs (x, y) can yield better results than actually aiming to hand-craft f^*[24]. For this purpose, feed-forward artificial neural networks (ANNs) define the mapping $h_\theta(x) = \hat{y}$, where θ is a set parameters which can be iteratively altered to better approximate f^*, based on datasets of expected input-output pairs (x, y).

Figure 2.1 illustrates an example of the basic building block of artificial neural networks, or the function h_θ, called the artificial neuron. Extending the previous paragraph, the input values x_i for $i = 1, ..., 3$ represent the input data x. $w_i \in \theta$ for $i = 1, ..., 3$ and $b \in \theta$ is a set of adjustable parameters. $\hat{y}$ is the output, or more commonly a single element of an output vector. Finally, the non-linear activation function s ensures that the network is able to approximate non-linear functions f^*. Hence, the output $\hat{y}$ of the neuron can be simply obtained as seen in equation 2.1.

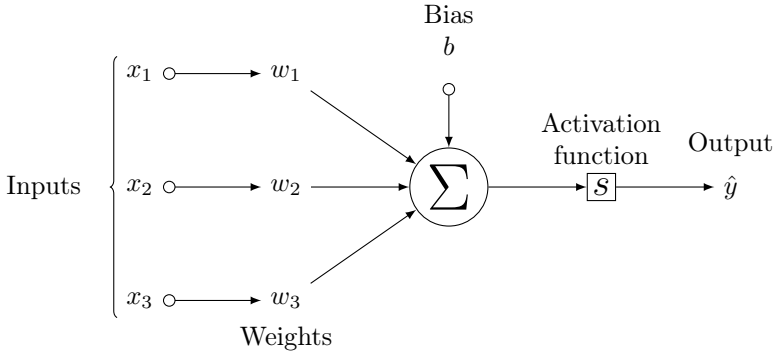


Figure 2.1: Example of a single artificial neuron.

$$s(\sum_{i=1}^{3}(w_i * x_i) + b) = \hat{y} \tag{2.1}$$

In practice, there are usually multiple such neurons stacked in a single layer, often feeding their outputs to another layer of neurons as seen in Figure 2.2. The intuition being that in order to approximate more complex functions f^*, we might require a larger set of parameters in θ. Thus, after successfully training the neural network, we may write $h_\theta = h^{(3)}_{\theta^{(3)}} \circ h^{(2)}_{\theta^{(2)}} \circ h^{(1)}_{\theta^{(1)}} \approx f^*$ for a network with three layers, each with their own set of adjustable parameters.

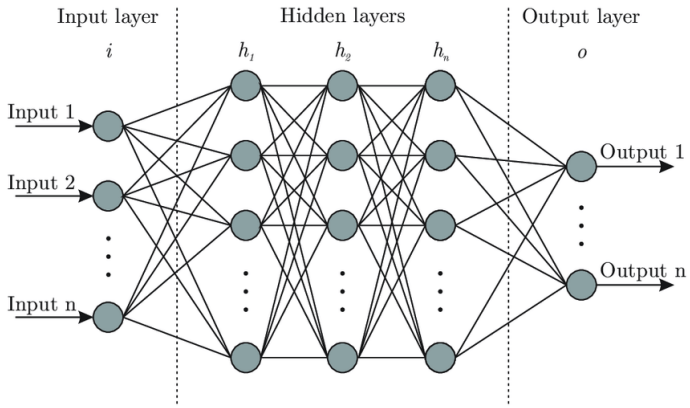


Figure 2.2: Example of an artificial neural network with one input layer, one output layer, and three hidden layers.[1] Each node in the network represents a single artificial neuron, as seen in Figure 2.1.

Updating the parameters in θ, so that our approximation h_θ converges to f^* is widely done using a technique called stochastic gradient descent. The idea is that once we obtain the value of $h_\theta(x) = \hat{y}$ for a single sample x, we can use a sample $f^*(x) = y$ to compute a differentiable cost function $c(y, \hat{y})$ on the expected output y, and the network output $\hat{y}$. Ideally, the cost function must ensure that its minima are in configurations where $y \approx \hat{y}$, this way minimising c via the parameters θ yields $h_\theta \approx f^*$. Using an algorithm called backpropagation[25], we are able to compute the gradient $\frac{\partial c(y,\hat{y})}{\partial w}$, with respect to any $w \in \theta$. Backpropagation thus allows us to adjust the parameter $w \in \theta$ toward a minimum of the function c. Performing these adjustments in $w \in \theta$ gradually allows the network to converge to minima in the function c, in which we expect to obtain a good approximation $h_\theta \approx f^*$.

In practice, multiple samples of x are batched to approximate the gradients in a technique called mini-batch stochastic gradient descent, as a gradient approximation from a single sample can be highly noisy. This means that rather than using a single sample x for computing the gradients, we instead average the gradients over a limited number of samples. The size of these batches needs to be tailored toward the optimisation problem at hand, together with the desired gradient step sizes in the optimisation process, in order to achieve good convergence.

It is widely accepted that training very deep ANNs is fairly difficult. However, empirical evidence also suggests that increasing ANN depth can increase model performance. To tackle issues with increasing ANN depth, there have been several initiatives that propose innovative layers that would enable seamless ANN depth scaling such as [26]. In our work, we leverage highway layers[26] to train deep ANNs for path planning. The key concept that allows for increasing the depth in these networks are gating units that learn to regulate the flow of information across the network.

The necessity to possess ground truth labels y is a key factor of **supervised** learning described above. In **unsupervised** learning, however, we do not have access to ground-truth values y in our efforts to approximate f^* . Some of the most popular techniques in unsupervised learning include cluster analysis, which aims to segment datasets based on various metrics.

2.1.2 Convolutional neural networks

Another concept important for our purposes, especially considering that we perform learning from images, are convolutional neural networks[24] (CNNs). Convolutions have been used in image

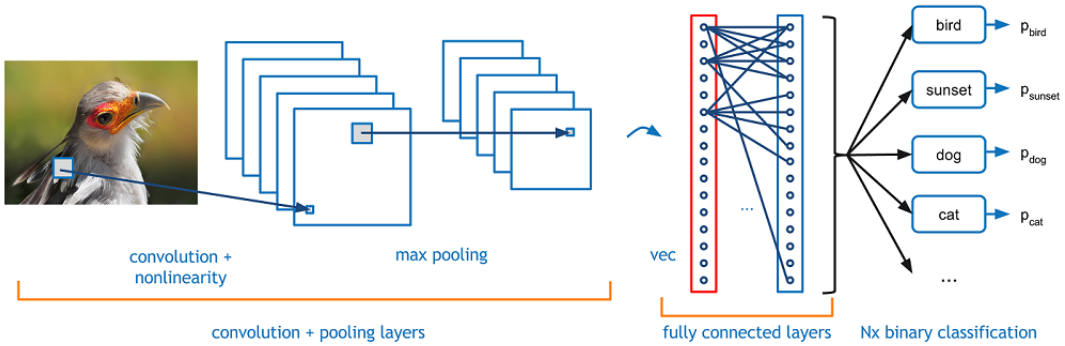


Figure 2.3: A standard CNN architecture for image classification. [2]

processing before they found their applications in deep learning, especially to extract interesting features from images such as edges or corners. By altering the values of the convolution kernels, we are able to specify the desired properties of output images. For example, if we want less noise in our images, we might use Gaussian kernels, or if we want to detect edges, Sobel kernels are a possible option. Unlike in fully-connected layers described above, in a convolutional layer, we try to learn the weights that should be present in convolution kernels. This way, the network learns to detect image features that best aid the minimisation of $c(y, \hat{y})$. As seen in Figure 2.3, apart from convolutions, network architectures might also contain pooling layers. Pooling layers are most commonly used in order to reduce the spatial dimensions of the input feature maps.

As in Section 2.1.1, CNN performance also benefits from increasing the CNN depth. To tackle challenges associated with increasing CNN depth, architectures such as AlexNet[24], VGG-16[27], or ResNet-50[28] have been proposed throughout the years. In our work, we leverage a ResNet-50[28] backbone to train a path planning network from depth images.

2.1.3 Recurrent neural networks

The final concept we examine in this section are Recurrent Neural Networks (RNNs) [25]. Although we do not make use of these architectures in our work, they are popular in the learning-based planning community. RNNs are popular primarily because of their ability to process temporally sequenced information, such as human language, video samples, or music samples. The high-level difference between RNNs and simple feed-forward neural networks is that they introduce a loop in the network architecture. That is, information processed later in the sequence affects earlier model decisions. One drawback of RNNs is that they suffer from the vanishing gradient problem, obstructing RNNs from learning long-range sequential dependencies. Among other approaches, this has been tackled by LSTMs[29] (Long Short-Term Memory) and GRUs[30] (Gated Recurrent Units), using different methods for preserving long-term dependencies in the network.

2.2 Non-uniform rational basis splines (NURBS)

In this section, we introduce the concept of non-uniform rational basis splines (NURBS), which we extensively utilise to generate paths.

2.2.1 B-splines

Suppose we are given a finite set of anchor points using which we would like to define a traversable path:

$$A \coloneqq \{\, p \mid p \in \mathbb{R}^d, d \in \mathbb{N}_{>0} \,\} \tag{2.2}$$

As an example, for $d = 2$, we could fit a polynomial that passes through all $p \in A$ to represent our paths. We could choose a degree $m = |A| - 1$ polynomial $P(x) = \sum_{i=0}^{m} a_i * x^i$ and solve a system of $m + 1$ linear equations to find the coefficients a_i for $i = 0, \ldots, m$. This approach would certainly work, but as we increase $|A|$ in an attempt to capture more complex paths, higher degree polynomials become sensitive to the positions of our anchor points and do not always yield smooth shapes. However, the smoothness of these polynomials along positions in A is a desired property,

as we would like them to model smooth paths that are safely traversable. To solve this problem, instead of fitting a single high degree polynomial, we choose to piece together several low degree polynomials by enforcing continuities along the joining anchor points. This way, we ensure some degree of smoothness across the spline crossing our anchor points.

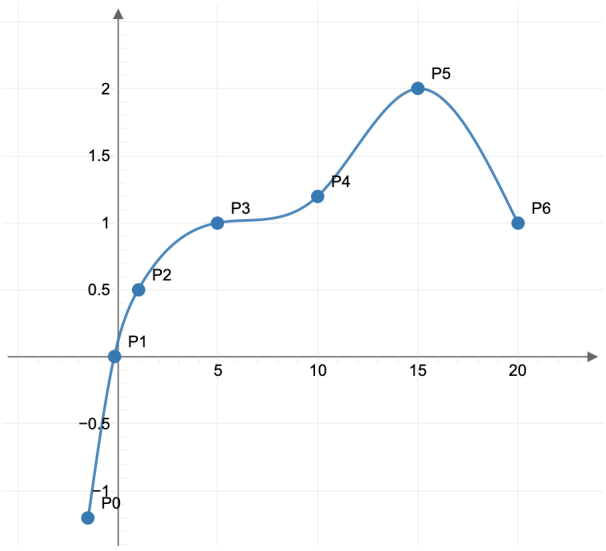


Figure 2.4: The spline above illustrates an example of fitting piecewise cubic polynomials between pairs of anchor points.

As we can see in Figure 2.4, we can create reasonably smooth splines by joining low degree polynomials. In our work, we aim to experiment with more general splines than piecewise cubic polynomials. For this reason, we consider basis splines (B-splines), which are defined via B-spline basis functions. Informally, the B-spline basis functions create a basis for the vector space of piecewise polynomials of the same degree. As a result, this formulation allows us to easily experiment with paths of arbitrary degree and control point count, by utilising common algorithms such as De Boor's algorithm[31] for performing the spline interpolation.

More formally, suppose we are given control points $A = \{ p_1, p_2, ..., p_m \}$, where $A \in \mathbb{P}(\mathbb{R}^d)$, and $d \in \mathbb{N}_{>0}$. Further, suppose we also have a total of m, order $1 \leq k \leq m$ corresponding basis functions N. Finally, we have access to an ordered knot vector $(t_1, t_2, ..., t_{m+k})$, with knot positions defining where splines pieces are joined together. Then, for arbitrary $t_1 \leq t < t_{m+k}$, we may write:

$$S(t) = \sum_{i=1}^{m} N_{i,k}(t) p_i \tag{2.3}$$

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \tag{2.4}$$

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where $S : \mathbb{R} \longrightarrow \mathbb{R}^d$ is the B-spline interpolation function.

As you may notice in equation 2.4, the value of S is defined recursively, which makes efficient batch computation challenging. By fixing k to reasonably small values, it is possible to unroll parts of the recursion tree and compute the values of the interpolation in-place. The Tensorflow Graphics package provides batch B-spline interpolations up to order $k = 5$, with an assumed underlying uniform knot vector (the knot values are equidistant) to produce so-called cardinal B-splines. Using a similar approach, we were able to implement interpolations with an option to provide arbitrary knot vectors for order $k = 4$ B-splines. In practice, paths always require open-uniform knot vectors, i.e., knot vectors with the values of the first and the last knot repeated k times, in order to anchor the spline in the start and goal configuration. While this could be achieved with our implementation, we can also repeat the first and last anchor points k times with a uniform knot vector to achieve the open-uniform effect. For this reason, we opted to rely on

the robust implementation present in the Tensorflow Graphics library while padding our control points k times in the start and goal configurations.

2.2.2 NURBS interpolation

NURBS extend B-splines by extending A with control point weights as:

$$A_{NURBS} \coloneqq \{ (p, w) \mid p \in \mathbb{R}^d, w \in \mathbb{R}, d \in \mathbb{N}_{>0} \} \tag{2.6}$$

and the interpolation is performed using rational basis functions as:

$$S_{NURBS}(t) = \sum_{i=1}^{m} R_{i,k}(t) p_i \tag{2.7}$$

$$R_{i,k}(t) = \frac{N_{i,k}(t) w_i}{\sum_{j=1}^{m} N_{j,k}(t) w_j} \tag{2.8}$$

where $S_{NURBS} : \mathbb{R} \longrightarrow \mathbb{R}^d$ is the NURBS interpolation function. Unfortunately, this interpolation is not directly implemented in standard deep learning libraries, however, we can overcome this by utilising a common approach for obtaining NURBS splines using B-spline interpolation[32]. We can simply convert A_{NURBS} to A' such that:

$$A' \coloneqq \{ [p, w]^T \in \mathbb{R}^{d+1} \mid (p, w) \in A_{NURBS} \} \tag{2.9}$$

and perform the B-spline interpolation using A'. Then, to convert back to $\mathbb{R}^d$ and obtain the results of the NURBS interpolation, we simply normalise the result using its last coordinate. Figure 2.5 demonstrates an example of B-spline interpolation in two dimensions.

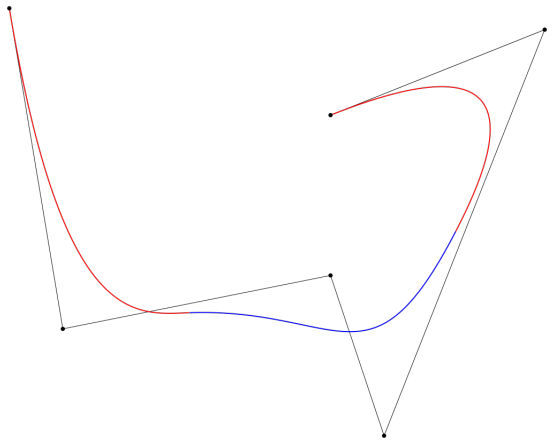


Figure 2.5: B-spline with an open-uniform knot vector, 6 control points, and highlighted components.

2.3 Signed Distance Function

Let Ω with boundary $\partial\Omega$ be a subset of the metric space M, with a metric d. Then, the signed distance function SDF is defined as:

$$SDF(x) = \begin{cases} -d(x, \partial\Omega) & \text{if } x \in \Omega \\ d(x, \partial\Omega) & \text{otherwise} \end{cases} \tag{2.10}$$

The signed distance function has been already leveraged in the field of gradient-based planning[23] and in this project, we likewise make extensive use of this function in the context of d being a euclidean metric and M being the euclidean space. Since the SDF is differentiable for most reasonable real world examples of Ω, and its gradients give useful information about how to escape Ω, we naturally found it useful in this project. In our work, we predominantly simplify planning problems by approximating the scene, or in this case Ω, using cuboids, spheres, and cylinders, for which it is possible to derive analytical SDF functions, which have corresponding analytical gradients. However, SDF gradients can be easily computed by finite differencing over the contents of Ω, thus we do not directly limit ourselves to primitive shapes by using this function. Further, there are several Ω independent algorithms such as the fast sweeping method[33] for calculating SDF values, or in the real world context we could also leverage DeepSDF[34]. Figure 2.6 presents an example shape with its corresponding SDF values.

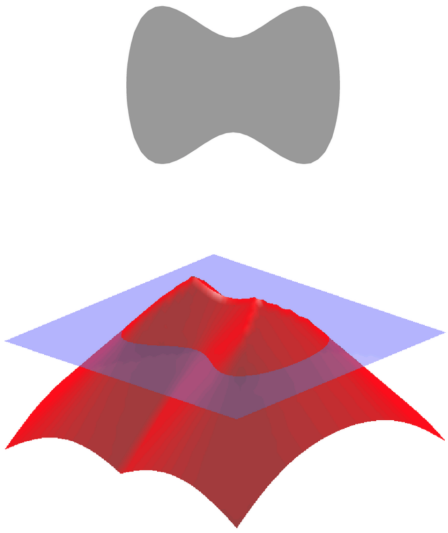


Figure 2.6: At the top, we have an example two dimensional shape, and the corresponding negative signed distance function values illustrated at the bottom. Notice that the gradients of the signed distance function generally point in directions away from the shape.

Chapter 3

Background

In this chapter, we aim to place our work on path planning in the context of past and current research initiatives. First, we draw from Yang at al. (2016)[35] who introduce a taxonomy of path planning algorithms. Their survey mainly focuses on approaches, which do not directly leverage stochastic optimisation, and so we extend their taxonomy with the most recent works in the domain of optimisation. Our main focus in this literature review is to evaluate how our proposed method compares to other research initiatives in path planning.

3.1 Classical planning

Based on [35], we can separate classical path planning algorithms into the following categories:

- Node-based algorithms
- Sampling-based algorithms
- Mathematical model-based algorithms
- Bio-inspired algorithms

3.1.1 Node-based algorithms

All path planning algorithms must assume some form of scene representation to perform planning in. Node-based path planning algorithms, in particular, rely on the fact that we can decompose a scene into fixed-sized cells. Representing the scene as such cells then naturally translates to graph scene representations, with each cell being a graph vertex, and cell adjacency captured as graph edges. When the cells are dense enough, these graphs need not be weighted, however, a possible edge weighting can be the Euclidean distance between adjacent cells.

Perhaps the most well-known node-based algorithm is **Dijkstra's shortest path algorithm**[36], which, for a given source graph vertex, can be used to find the shortest path to all other graph vertices. The **A* algorithm**[37] extends the Dijkstra's shortest path algorithm by adding a heuristic to better guide the algorithm's exploration toward the target locations of the planning problem. Figure 3.1 presents the Dijkstra's shortest path algorithm pseudocode, together with a simple two-dimensional planning problem solved by the algorithm. Figure 3.2 then extends the algorithm by introducing the A* heuristic and solves the same planning problem as in Figure 3.1. Notice that the exploration of the A* algorithm was reduced compared to Dijkstra's shortest path algorithm due to the introduction of the aforementioned heuristic. Note that Dijkstra's shortest path algorithm is a specific case of A* with the heuristic values being always 0.

In many cases, A* is one of the best ways to solve planning problems in terms of its performance, path length, and completeness. In simple terms, A* always finds the shortest path if such a path exists in a reasonable run-time. We might then pose the question, why do other planning methods even exist? The first reason as to why there is a need for alternative algorithms is the fact that A* relies on the ability to discretise the scene it operates in. In continuous configuration spaces, as we increase the dimensionality of the problems, the storage and querying of these search graphs become highly inefficient. For this reason, probabilistically complete algorithms such as RRT[38]

have been developed, which we discuss in Section 3.1.2. Further, A* implicitly assumes that all cells in the scene are observable or that no nodes in the graph are missing. Unfortunately, autonomous agents are rarely equipped with oracle knowledge about the world, and rather rely on noisy sensor readings to make decisions. Lastly, A* is also not equipped with the tools to deal with scenes that change during planning. All these shortcomings motivate the algorithms described in the following sections, and our proposed method is equipped to deal with all of these limitations.

```
DIJKSTRA(G : graph, s : vertex)
 1  for each vertex v ∈ V_G
 2      dist[v] = ∞
 3      parent[v] = NIL
 4  dist[s] = 0, Q = V_G
 5  while Q ≠ ∅
 6      u = EXTRACT-MIN(Q)
 7      for each edge e = (u, v)
 8          if dist[v] > dist[u] + weight[e]
 9              dist[v] = dist[u] + weight[e]
10              parent[v] = u
11  H = (V_G, ∅)
12  for each vertex v ∈ V_G, v ≠ s
13      E_H = E_H ∪ {(parent[v], v)}
14  return H, dist
```

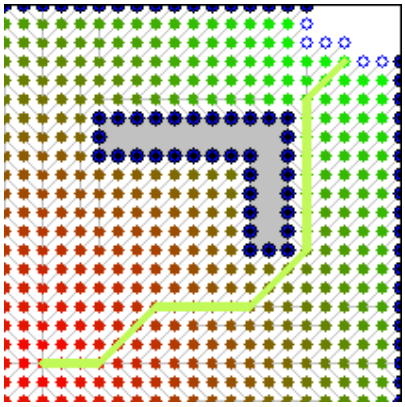


Figure 3.1: On the left, we have a pseudocode for Dijkstra's shortest path algorithm. Note that the EXTRACT-MIN function simply returns the lowest weight vertex from Q. Q in practice is often implemented as a priority queue. On the right, we have an example of running Dijkstra's shortest path algorithm on a simple two-dimensional planning problem[3]. The algorithm is terminated when it reached the target location. The grid cells represent the vertices explored by the algorithm before finding the target location.

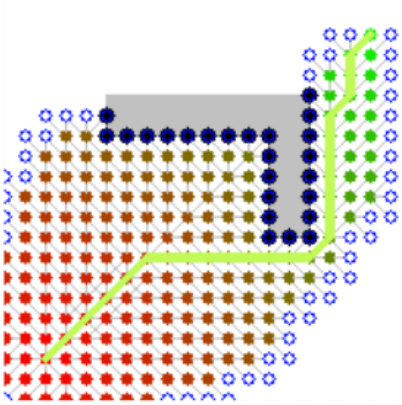


Figure 3.2: A* due to its heuristic allows us to solve the planning problem presented in Figure 3.1 with less unnecessary exploration.[4]

3.1.2 Sampling-based algorithms

Rapidly-exploring random trees

Sampling-based algorithms are capable of effectively searching for optimal paths in high-dimensional continuous spaces. An example of a sampling-based algorithm is the **rapidly-exploring random tree (RRT)** algorithm, introduced by LaValle et al. (1998)[38]. To quickly find a route between a source and a target location, the RRT algorithm attempts to efficiently fill the space it is expanding in. Rooted in the starting configuration, the RRT algorithm uniformly samples the scene, and if possible, attempts to connect the random sample to the nearest tree vertex that has been already constructed. The connection would fail if connecting the RRT to the random sample would cause a collision with objects present in the scene. In Figure 3.3, we can observe how the RRT expands during incremental planning iteration steps.

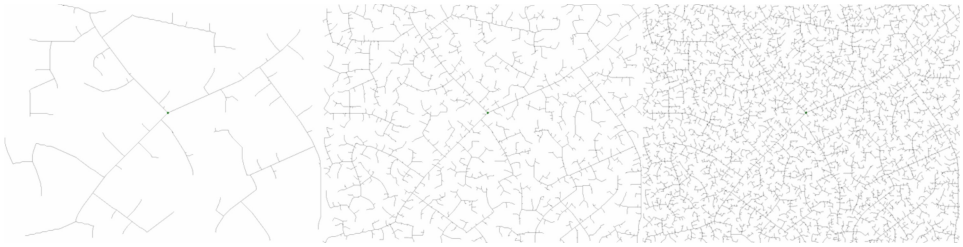


Figure 3.3: RRT algorithm at different iteration steps. From left to right, we can observe how the RRT fills the space from the centre starting configuration as iterations increase.[5]

Various other methods such as RRT-Connect[39], RRT*[40], RRT*-Smart[41], RRT*FN[42], and Informed RRT*[43] have since built on the idea of RRTs to optimise some aspect of the tree construction. All of these approaches extend the basic RRT approach with different heuristics. RRT-Connect, for example, attempts to grow two versions of the RRT in parallel, both in the source and the target configuration until the two trees are merged. This way, RRT-Connect achieves significant performance boosts as compared with classic RRT. RRT* extends the basic RRT concept by recording the cost of each node in the tree, and rewiring, should a feasible lower cost path be found. This way, RRT* ensures that the path is optimal as the number of the sampled nodes approaches infinity. In general, all of the RRT flavours mentioned above have their appropriate applications, most importantly, all of them are probabilistically complete, that is, the algorithms can find feasible paths as sampling approaches infinity.

The probabilistic completeness of RRT in high-dimensional continuous spaces suggests that RRT addresses all the problems of A* that we analyse in Section 3.1.2. Once again, it seems there is nothing more to be said in terms of path planning. However, the computational complexity distribution of RRT-based planning methods is said to be heavy tailed[44], as it is hard to estimate their performance due to its world-state dependence and random sampling. Hybrid methods, combining deep learning with RRT, have thus shown increased inference speed capabilities by largely guiding the sampling process via a neural network[22]. RRT-based algorithms further tend to require full scene state knowledge in order to perform planning. However, as we mention in Section 3.1.1, it is uncommon for robots to be equipped with oracle knowledge about their surroundings. Lastly, RRT-based methods often display metric sensitivity[45]. Metric sensitivity means that the performance of these methods is sensitive to the size of metric heuristics selected to perform more greedy sampling in the scene. As we propose a metric-independent regression network, and further demonstrate planning capabilities from partial observations, we address the aforementioned issues RRT-based methods face. On the contrary, our proposed method is not probabilistically complete, which suggests that future extensions might involve attempts to achieve this.

Artificial potential fields

Another important class of sampling-based algorithms are **artificial potential fields**. This method is attractive due to its responsiveness in real-time planning applications. The basic idea of the artificial potential fields approach is that an agent should be attracted to the goal configuration, while being simultaneously repelled by the obstacles. The potential of the robot in a given configuration is calculated as the sum of all the incident attractive and repelling forces. The negative gradient of this sum is the most promising direction of motion from a specific robot configuration. The choice of the object repulsion function plays a key role in this method, as it determines the gradients present in the algorithm. Most commonly, a combination of paraboloidal and conical functions are used for the attractive potential of the objective, and repulsive forces are loosely modelled by the inverse distance of the robot from the objective. Guldner et al. (1995)[46] were able to successfully put the principle of artificial potential fields into practice. Figure 3.4 illustrates the potential gradients of the artificial potential fields method in a simple two-dimensional planning problem.

While computationally feasible, the artificial potential fields method often fails to produce globally optimal paths, as it tends to be sensitive to local potential gradient field minima. There have been proposed methods to counter this phenomenon[47], and it remains and active area of research. Artificial potential fields are further known to show poor performance in narrow passages

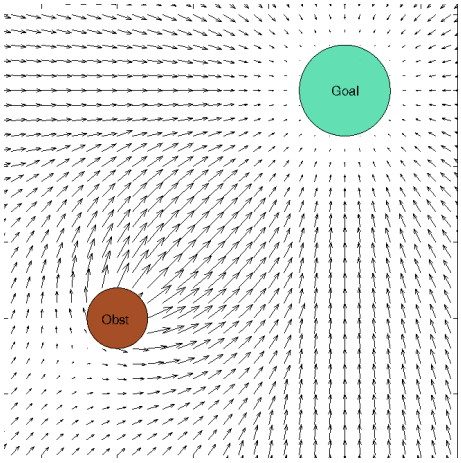


Figure 3.4: Demonstration of the potential field gradients created by a sphere obstacle in a scene.[6]

and dynamic environments, and run into problems with symmetrical obstacles. These shortcomings mean that these methods are rarely deployed in practical settings, but their ability to utilise scene specific gradients for path planning informed our approach.

3.1.3 Mathematical model-based algorithms

The most widely employed mathematical model-based path planning algorithms include linear programming and optimal control-based approaches.

Linear programming-based approaches are able to formulate pathfinding as a convex optimisation problem with numerous constraints. Linear programming is expressive enough to encode various physical constraints, which can include the size of the moving robot[7] or various dynamics constraints[48]. The cost function the linear programs aim to optimise usually contains terms such as path length, distance from objects, or expanded energy. This way, linear programming can find optimal paths with various desired domain-specific properties. Figure 3.5 demonstrates the output of a linear program's path prediction[7], constrained by the car dimensions. **Optimal control**-based approaches are able to formulate the pathfinding problem as a set of differential equations.[49]. Similarly to linear programming-based methods, these approaches encode extensive sets of constraints.

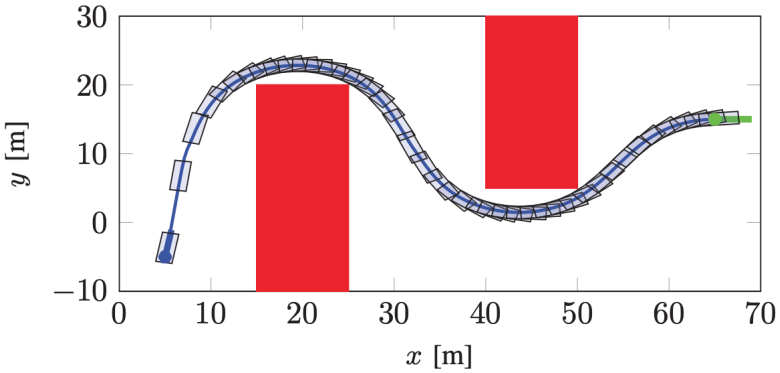


Figure 3.5: Example of an output from a linear programming-based path planning algorithm of Plessen et al. (2017)[7]. The figure demonstrates the optimal path's awareness of car dimensions.

While mathematical model-based algorithms directly provide an increased degree of expressiveness to solve constrained path planning problems, they usually suffer from an extremely slow inference speed, due to their requirement of per-scene optimisation. They have been shown to be usable in real-time applications[50], however, such algorithms still fall behind both sampling-based methods and our method in terms of responsiveness. Further, each method is largely tailored

toward a domain-specific application, which makes them hard to apply in numerous areas without drastic changes to their formulations.

3.1.4 Bio-inspired algorithms

Genetic algorithms have been previously applied to the path planning problem. Genetic algorithms generally have five basic steps: initialisation, selection, genetic crossovers, mutations, and termination. The idea behind genetic algorithms is that by selectively combining the best samples of a population, a genetic algorithm can terminate with a highly sophisticated solution for a specific task. Erinc et al. (2007)[51] successfully applied this principle to the path planning problem. Erinc et al. (2007) initialise their population with samples of RRT solutions. They do this without checking for RRT collisions, to speed up the population generation process. The genetic crossover operation that Erinc et al. (2007) propose is suited to optimise the paths for the requirements such as path smoothness, but primarily focusing on length minimisation and collision avoidance. Usually, multiple mutations need to be performed in order to ensure that the genetic algorithm converges to optimal paths. This means that these approaches are more suitable for offline planning settings, where responsiveness is not a key requirement.

3.1.5 Summary

In this section, we unfolded the path planning taxonomy proposed by Yang at al. (2016)[35] in the context of our method. Classical approaches, especially node-based planning 3.1.1 and sampling-based planning 3.1.2, have been the go-to planning methods in the past decades in various contexts due to their reliability and performance. However, as recent research demonstrates [22, 52], deep learning is a promising avenue for enhancing or replacing these methods. In Section 3.2 we dive deeper into the latest research in path planning using stochastic optimisation and consider how our methods fit into the current landscape of the state-of-the-art methods for path planning.

3.2 Stochastic optimisation-based planning

The advent of deep learning inspired a wave of learning-based path planning approaches. A wide range of reinforcement learning[53, 54, 55, 56], supervised learning[57, 58], and logic-based learning[59] methods have been proposed to tackle aspects of path planning. In parallel, other optimisation-based methods, which do not rely on stochastic gradient descent have been developed such as CHOMP[23], STOMP[60], ITOMP[61], or GPMP2[12]. From a high level, our proposed method can be viewed as a combination of the two paradigms that were created in parallel. In particular, we address certain shortcomings related to objective formulations of methods such as [23, 60], which allows us to apply our method in a wide variety of planning settings deep learning has already been deployed in. In this section, we examine these methods in detail and consider the innovations our method introduces.

3.2.1 Two-dimensional path planning

RNN-based methods

RNNs (refer to 2.1) have been used extensively to tackle the two-dimensional path planning problem[62, 8, 63, 64]. The motivating reason for using RNNs for path planning is that in simple setups, paths show sequential dependence. That is, if a part of a path is forced to avoid an obstacle, it is likely that the adjacent sequence needs to avoid an obstacle as well. Likewise, in simple scenarios, if a part of a path is straight, it is likely that the next part will remain straight.

Out of all the examined approaches, the one of Nicol et al. (2018)[62] most closely resembles local path planning, that is, the proposed algorithm produces a full path, but the path is iteratively generated by inputting the agent state to the RNN. This way, the method obtains the following action an agent should take, leading the robot toward the objective location, iteratively generating the full path. The RNN is trained in a supervised setting on randomly generated ASCII mazes, using A* ground truth paths. Although the success rate of this approach remains low (26% in a challenging maze setup), the authors argue that this counter-balanced by the online nature of their RNN setup, which means the full world-state is not available to the agent.

An LSTM-based approach[8] by Inoue et al. (2019) makes use of an auto-encoder network to process obstacle maps, before they are used to train an LSTM. This enables the authors to reduce the size of the LSTM, which increases inference speed. The LSTM network is trained on encoded obstacle maps, and pairs of source-target locations. The supervision is performed using ground truth RRT paths. As opposed to the work of Nicol et al. (2018), the path inference is performed with full world-state knowledge.

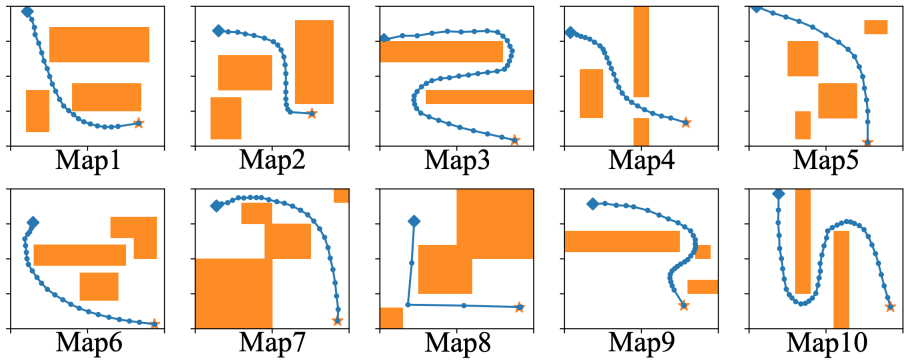


Figure 3.6: Examples of successfully planned LSTM-based paths from the work of Inoue et al.(2019)[8].

CNN-based methods

Purely CNN (refer to 2.1) based two-dimensional path prediction techniques have been less popular in the learning-based path planning community, however, they have been applied in setups where constraints beyond path properties had to be taken into account. Several works[9, 65, 66] show that CNNs have the capacity to learn planning task-driven geometry representations. In most cases, this means processing images to predict future possible vehicle locations. Banzhaf et al. (2019)[9] was able to train a CNN to predict future possible vehicle paths based on the scene geometry, vehicle dimensions, and current vehicle configuration. They successfully integrate their CNN-based method to guide configurations of BiRRT* to improve path planning responsiveness. Figure 3.7 illustrates the network architecture utilised in Banzhaf et al. (2019)[9], together with the input and output representations used in the planning task.

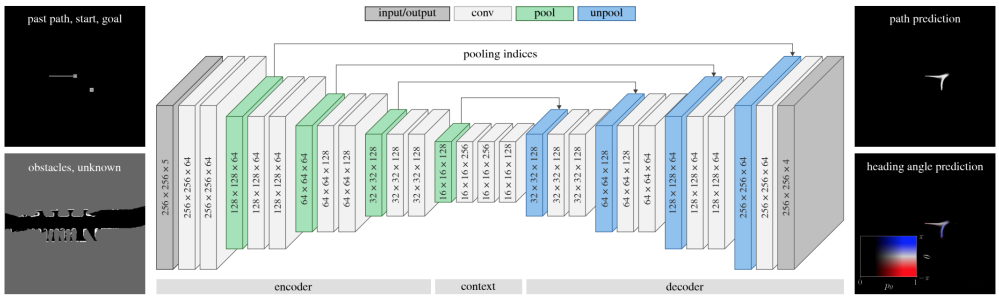


Figure 3.7: Banzhaf et al. (2019) [9] proposed CNN architecture for vehicle motion prediction. The network receives past path, and obstacles as an input, while outputting a feasible path projection together with a heading angle prediction.

Value iteration networks

Value iteration networks (VIN) [10] train an end-to-end network, with an architectural design which encourages implicit path planning via value iteration during the network forward pass. The key to their approach is a differentiable approximation of the value-iteration algorithm, which

can be represented as a CNN. In [10], these networks are trained on two-dimensional discrete maze-like environments supervised by Dijkstra shortest paths. While this novel paradigm achieves impressive performance and generalisation in the presented grid worlds, it is yet to be seen how this approach can effectively generalise in continuous settings or high-dimensional action spaces. Figure 3.8 presents a sample planning problem solved by VIN.

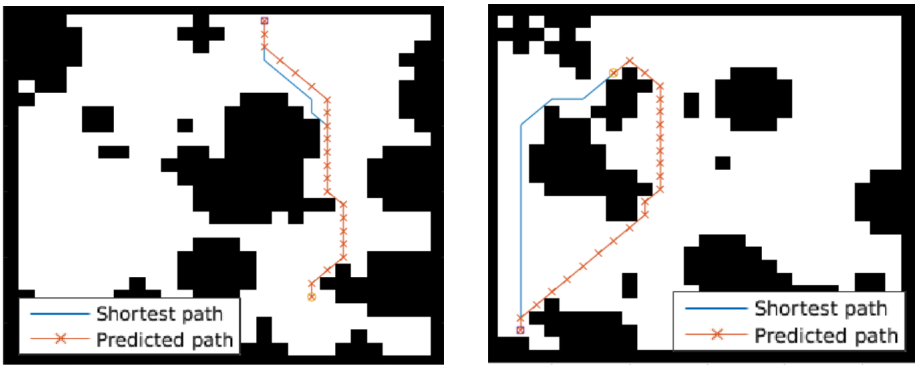


Figure 3.8: Sample path planning problems solved by VIN[10] in maze-like grid environments.

Summary

With the introduction of value iteration networks[10], deep learning methods became near-perfect at solving path planning problems in maze-like environments as presented in Figure 3.8. In our work, we aim to define a dimensionality invariant approach, which means that we expect to deploy our method in higher than two-dimensional spaces as well. The results of VIN on the illustrated grid worlds, however, form a solid baseline for us to perform comparisons against. Experiments on the demonstrated two-dimensional challenging planning settings will allow us to gauge how generally applicable our approach is.

3.2.2 Higher-dimensional path planning

Higher-dimensional planning approaches are more closely related to our work. In this section, we review and compare several state-of-the-art methods.

Motion Planning Networks

Motion Planning Networks(MPNet) [11] by Qureshi et al. (2019) leverage a multi-stage planning approach to finding optimal paths. The first element of the deep learning part of their pipeline consists of a point cloud (3D) or image (2D) encoder network, similar to the one seen in the Nicol et al. (2018) [62]. The encoder can be either trained using a reconstruction loss, or end-to-end, using PNet. The PNet represents a feed-forward neural network, which given the encoded representation, and a current robot state, is able to predict the following robot state, which would bring the robot closer to the target location. Hence, the encoder can be trained by backpropagating the loss of PNet, in an end-to-end fashion. The loss function of PNet is obtained by calculating the MSE loss between ground truth expert demonstrations or samples of RRT* planner solutions and MPNet's predictions. Figure 3.9 demonstrates some of the results Motion Planning Networks were able to achieve both in 2D and 3D settings.

The main part of the non-learning-based module, called Neural-Planner, uses a bidirectional procedure to generate collision-free paths using MPNet. By extending an MPNet path iteratively both from the start and the target locations, in an RRT-Connect-like fashion, this routine tries to directly connect the expanding paths if there exists a direct, collision-free segment between them. In case such a path is not found, the authors implement a replanning functionality that leverages the stochasticity of the network to generate a different set of near-optimal bidirectional paths, or otherwise calls RRT* on a short segment of the workspace. In a practical online application, the Neural-Planner routine would be queried for each replanning request. An overview of the aforementioned components can be found in Figure 3.10.

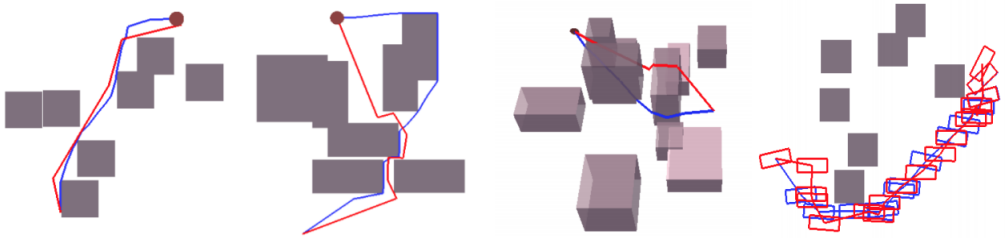
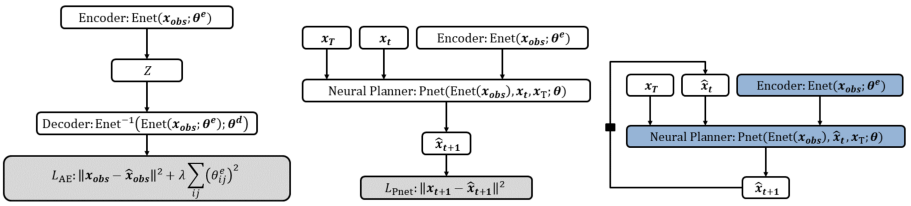


Figure 3.9: Red paths are generated by Motion Planning Networks based on the work of Qureshi et al. (2019)[11], while the blue paths represent ground truth RRT* paths.



(a) Offline: Encoder Network (b) Offline: Planning Network (c) Online: Neural Planner

Figure 3.10: This figure illustrates the different planning components of Motion Panning Networks[11] described above. On the left, we have the training of the encoder network, together with the demonstration of PNet. We can notice that the output of a single PNet forward pass is the next desired robot state $\hat{x}_{t+1}$. On the right, Neural Planner is capable of unrolling the full path online.

In terms of computational efficiency, the MPNet approach is shown to consistently outperform state-of-the-art sampling-based planners such as RRT* or BIT*. While the approach of Qureshi et al. (2019) shows very promising performance for 3D planning tasks, Jurgenson et al. (2019) [67] demonstrate that these approaches are failure-prone in challenging planning settings.

As hinted above, the MPNet method employs multiple path optimisation strategies on top of the network output, that are not learning-based. To summarise, the two main additional non-learning-based methods MPNet uses are lazy state contraction and replanning. Briefly, in the replanning step, the MPNet method attempts to regenerate colliding path segments either using an oracle planner, or MPNet itself. In lazy state contraction, the algorithm greedily tries to connect non-consecutive states, if these connections are collision-free, this way pruning extraneous states which could be colliding.

OracleNet

OracleNet[52] extends the two-dimensional use case of LSTM-based path predictions to a higher-dimensional setting. As the name of the network suggests, this approach uses an A* planning algorithm as an oracle for training the path prediction process. Similarly, as in the case of PNet, OracleNet creates paths sequentially, predicting the next optimal position that the robot should move to. Accordingly, the training signal of A* paths is deconstructed into discrete components to train the LSTM sequentially. Further, OracleNet uses a virtually equivalent method as PNet for constructing paths in an iterative, bidirectional manner. Figure 3.11 presents the ability if OracleNet to bring a robotic arm to a target configuration.

As we saw earlier, there are great similarities between the OracleNet and the MPNet approach in terms of their implementation. Both approaches unroll paths iteratively based on greedy next-best robot configurations. One disadvantage of OracleNet is that it is not trained on any scene signal, hence it is not capable of generalising to previously unseen environments and needs to be re-trained on every setup. This means that for comparisons, the MPNet approach is more directly comparable to our method.

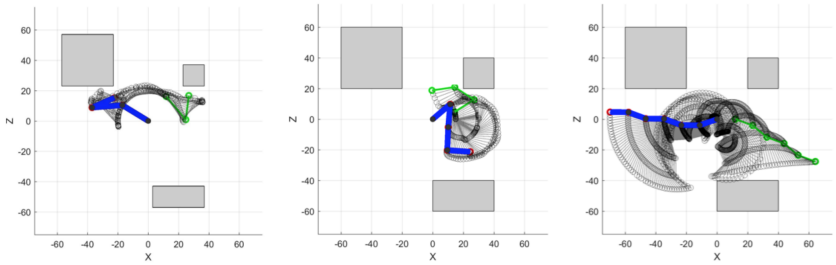


Figure 3.11: From left to right, we can see 3, 4, and 6 link robotic arms moving to a (green) target configuration based on the OracleNet approach.

Reinforcement learning-based planning

Jurgenson et al. (2019) [67] argue that path planners such as OracleNet or MPNet are severely limited in cases where agents have to pass through complex, tight spaces. The authors attribute this phenomenon to the training data distribution, as there is no information about the oracles (A* or RRT*) hitting, or coming very close to an edge. Figure 3.12 illustrates the phenomenon of underrepresented edge cases. Our approach does not use oracles for training, which means that this shortcoming does not apply in our case.

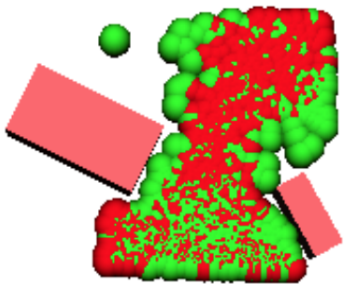


Figure 3.12: In this image, Jurgenson et al. (2019) illustrate the distribution of end-effector poses when trained in a supervised setting via RRT* or A* supervision. The edge distributions are much less dense than the ones passing in the middle.

Jurgenson et al. (2019) design an RL actor-critic-based method. The authors introduce two key ideas in their work: model-based actor updates and targeted expert demonstration-based exploration. As for their actor update, they rely on knowing the dynamics and the reward function to reduce the policy gradient approximation errors of the actor. They can approximate the Q^{π} function to arbitrary precision using [68], to create a novel smoothed domain-specific actor update. The authors further note that uniform explorations yield a high sample inefficiency, and hence provide the agent with motion planner demonstrations for failed workspaces. This way, the actor can learn to make a very specific series of steps in an adversarial environment setup. This is an important innovation, as usually, in a uniform exploration setting, the actor is unlikely to make the specific steps to solve a hard planning problem as there is no signal motivating seemingly risky actions.

Additional optimisation-based planning approaches

In this section, we aim to address optimisation-based planners that do not rely on stochastic gradient descent. These methods aim to formulate a path cost function, which is only dependent on the scene and the dynamics of the planning agents. For these reasons, these approaches are closely related to our work, as planning without ground truth path supervision is one of our key goals.

One approach these methods take is to model paths as mass-spring systems: with paths having length-dependent internal energy, with additional external energy generated by obstacles [69, 70].

Alternatively, collisions can be handled as a constraint, using generalised Voronoi diagrams [71]. These approaches typically require an initial path estimate from sample-based planners. CHOMP demonstrated that paths can be optimised directly from naive initial guesses, with no sample-based planner predictions beforehand [23], using the gradients of their cost function. STOMP then demonstrated that adding stochasticity into the optimisation enables the avoidance of local minima [60], and ITOMP extended these methods to handle object dynamics [61]. GPMP2 [12] further demonstrate how a Gaussian process path parameterisation can improve the overall optimisation-based planning methodology. Figure 3.13 demonstrates the gradients present in a single step of the GPMP2 [12] optimisation process.

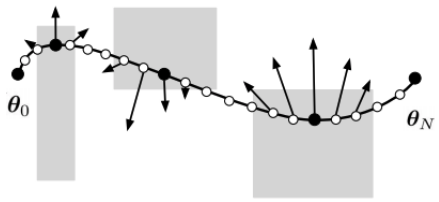


Figure 3.13: Illustration of the gradients in a single optimisation step of the GPMP2[12] method.

A key difficulty with the planners mentioned above lies in formulating a cost function that includes both *collision* and smooth *shortest path* terms. This is challenging as collisions are hard-constraints, making it difficult to equate them to path length without hand-engineering weighting parameters. Figure 1.1 demonstrates the key innovation of our approach, which builds on top of this part of the optimisation-based planning literature. Apart from the difficulty with choosing a cost function, the inference time of these methods is far from real-time, as they are required to run separate optimisations for each scene, which can be prohibitive in dynamic contexts requiring fast robotic responses. Our cost function is similar in form to those of the works presented above, but unlike these methods, we derive a formulation whose minima guarantee collision-free solutions with minimal path length.

3.2.3 Summary

To summarise, the deep learning-based approaches we have examined allow for two possible innovations:

1. **Regression planning**: All the works we have examined perform iterative planning. The current agent state is used to infer the next state, in order to bring the agent closer to the target configuration, and this way plan paths. We propose regression networks, which plan a full start-to-target path in a single inference step, bringing inference speed improvements close to orders of magnitude.

2. **Unsupervised learning**: The deep learning approaches we have examined do not utilise the information of the scene as extensively as other optimisation-based planning techniques, but rather rely on some form of supervision from an oracle path planner. We argue, that by addressing issues of other optimisation-based approaches (3.2.2) related to path and scene scale invariance, we can solely rely on scene supervision to learn path planning which generalises across diverse planning settings. This can be seen as a form of unsupervised learning, as our networks never witness examples of optimal paths during training.

In the following chapters, we describe how we achieve the two points mentioned above, and how this novel approach compares to existing techniques.

Chapter 4

Approach

In this chapter, we outline the full derivation of our cost function. A key challenge in formulating a smooth cost function for shortest path planning lies in the fact that collision avoidance is a hard constraint, which is often replaced by soft penalty terms. Finding a scale invariant weighting between collisions and path lengths is challenging, as they are not directly comparable. For this reason, common optimisation-based methods [72, 23, 61, 60] require per-task calibration, which is not possible in our setup, as we aim to optimise our planners so that they generalise across diverse sets of scenes, rather than a single planning problem. Hence, we derive a novel formulation which guarantees collision-free paths, invariant to path or scene scaling. Further, we introduce a parameterisation such that our cost function is differentiable, and our path representation has sufficient expressive capacity. Lastly, we discuss a general optimisation process for our path planning networks, together with several training considerations.

4.1 Cost function derivation

Given a set of arbitrary obstacles O, the corresponding obstacle observations Π, a source configuration $s \in \mathbb{R}^d$, and a target configuration $t \in \mathbb{R}^d$, we aim to optimise the path planning function:

$$f_{plan} : \mathbb{P}(\Pi) \times \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \Theta \tag{4.1}$$

where Θ represents a set of all possible path parameterisations in the considered task space. To successfully optimise f_{plan}, we define a cost function:

$$c : \Theta \times \mathbb{P}(O) \times \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R} \tag{4.2}$$

In our work, we aim to optimise f_{plan} to converge to paths which are shortest and collision-free. This assumption naturally leads to c having components penalising collisions and path lengths, respectively. Hence, we may write:

$$c(\theta, o, s, t) = l(\theta, s, t) \; + \; c_{coll}(\theta, o, s, t) \tag{4.3}$$

where $\theta \in \Theta$, $o \in \mathbb{P}(O)$. Further,

$$l : \Theta \times \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}_{\geq 0} \tag{4.4}$$

represents a suitable path length measure in the task space, and

$$c_{coll} : \Theta \times \mathbb{P}(O) \times \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}_{\geq 0} \tag{4.5}$$

is a collision cost measure.

While the exact structure of c_{coll} is unknown for now, we can already point out useful properties c_{coll} should have. Suppose for an arbitrary path planning problem with $o \in \mathbb{P}(O)$, $s \in \mathbb{R}^d$, and $t \in \mathbb{R}^d$ our optimisation problem has an optimal path parameterised by $\theta_{opt} \in \Theta$. Then, in order for f_{plan} to converge to non-colliding paths, we require:

1. **Minimum property** :

$$\forall\, \theta \in \Theta :\ c_{coll}(\theta_{opt}, o, s, t) \leq c_{coll}(\theta, o, s, t) \tag{4.6}$$

2. **Non-colliding property** : Assuming that a non-colliding solution always exists, we require for $\theta \in \Theta$ such that the corresponding path does not collide:

$$c_{coll}(\theta_{opt}, o, s, t) = c_{coll}(\theta, o, s, t) = 0 \tag{4.7}$$

3. **Global optima property**:

$$\forall\, \theta \in \Theta :\ c(\theta, o, s, t) \geq c(\theta_{opt}, o, s, t) \tag{4.8}$$
$$\Leftrightarrow \quad c_{coll}(\theta, o, s, t) \ + \ l(\theta, s, t) \geq c_{coll}(\theta_{opt}, o, s, t) \ + \ l(\theta_{opt}, s, t) \tag{4.9}$$
$$\Leftrightarrow \quad c_{coll}(\theta, o, s, t) \ \geq l(\theta_{opt}, s, t) \ - \ l(\theta, s, t) \tag{4.10}$$

Theorem 1.

$$\forall \theta \in \Theta, o \in \mathbb{P}(O), s \in \mathbb{R}^d, t \in \mathbb{R}^d :\ c_{coll}(\theta, o, s, t) = \sum_{o' \in o} \mathbb{1}_{o'}(\theta) * C(o') \tag{4.11}$$

where $C : O \longrightarrow \mathbb{R}_{\geq 0}$ *is the circumference of the bounding sphere around the provided object and*

$$\mathbb{1}_{o'}(\theta) = \begin{cases} 1 & \textit{if path given by } \theta \textit{ collides with } o' \\ 0 & \textit{otherwise} \end{cases} \tag{4.12}$$

satisfies minimum property (1), non-colliding property (2), and global optima property (3).

Proof.
Take an arbitrary path planning problem with $o \in \mathbb{P}(O)$, $s \in \mathbb{R}^d$, and $t \in \mathbb{R}^d$ with an optimal non-colliding shortest path parameterised by $\theta_{opt} \in \Theta$. Further, let $S : \Theta \times \mathbb{P}(O) \longrightarrow \mathbb{P}(\Theta)$ be defined to provide a set of path segments that are colliding with objects in the scene.

In case of (1, 2), for any $\theta \in \Theta$ such that $S(\theta, o) = \emptyset$, we will have

$$\forall o' \in o : \mathbb{1}_{o'}(\theta) = 0 \tag{4.13}$$

Hence, by 4.11, 4.13, for an arbitrary non-colliding path $\theta \in \Theta$, we will necessarily have $c_{coll}(\theta, o, s, t) = 0$. Further, by the assumption that θ_{opt} must be non-colliding, we also have $c_{coll}(\theta, o, s, t) = c_{coll}(\theta_{opt}, o, s, t) = 0$. Because c_{coll} is defined to be a measure, $c_{coll}(\theta_{opt}, o, s, t) = 0$ also happens to be a minimum, so further $\forall \theta \in \Theta : c_{coll}(\theta_{opt}, o, s, t) \leq c_{coll}(\theta, o, s, t)$. We have shown both (1) and (2).

For (3), take an arbitrary $\theta \in \Theta$ and consider two cases:

Case 1: $l(\theta_{opt}, s, t) \ < \ l(\theta, s, t)$

In this case, $l(\theta_{opt}, s, t) \ - \ l(\theta, s, t) \ < \ 0$, so (3) holds by the definition of c_{coll} 4.11. Particularly, because c_{coll} is always non-negative.

Case 2: $l(\theta_{opt}, s, t) \ \geq \ l(\theta, s, t)$

First, suppose that the path θ does not collide with any objects in o. By the assumption of this case, we have $l(\theta_{opt}, s, t) \geq l(\theta, s, t)$ and so necessarily also $l(\theta_{opt}, s, t) = l(\theta, s, t)$. Otherwise, θ would have yielded a shorter path than θ_{opt}, which is a contradiction with θ_{opt} being the optimal path. Hence, by (2) and θ being non-colliding, we have:

$$c_{coll}(\theta, o, s, t) = l(\theta_{opt}, s, t) \; - \; l(\theta, s, t) = 0 \tag{4.14}$$

which implies that (3) holds in this case.

Now in turn, suppose that the path θ does collide with objects in o. Since we aim to define a cost function which is not supervised by information about θ_{opt}, we do not have access to θ_{opt} and hence to $l(\theta_{opt}, s, t)$. Because of this limitation, we will now aim to derive an upper bound for $l(\theta_{opt}, s, t) \; - \; l(\theta, s, t)$ by defining an oracle transformation:

$$T : \Theta \times \mathbb{P}(O) \longrightarrow \Theta \tag{4.15}$$

such that

$$\forall \theta \in \Theta, o \in \mathbb{P}(O) : S(T(\theta, o), o) = \emptyset \tag{4.16}$$

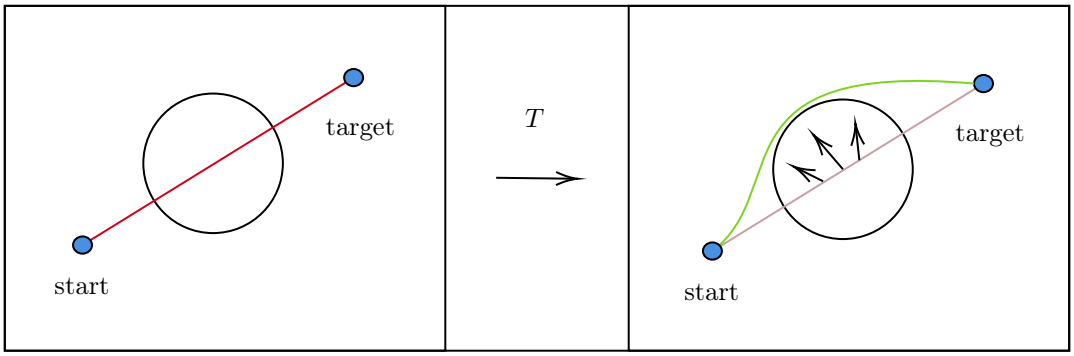


Figure 4.1: The T transformation transforms a colliding path segment into a non-colliding segment.

Figure 4.1 demonstrates an example valid T transformation. Note that we must have $l(T(\theta, o), s, t) \geq l(\theta_{opt}, s, t)$ as θ_{opt} is shortest. In practice, the exact definition of T could be task space specific and we would be able to prove Theorem 1 and consequently optimise f_{plan}. However, defining T to project the colliding segments of the path θ marginally beyond its respective colliding objects, or possibly on the respective object bounding spheres is sufficient. We assume that the scene geometry always allows for a T transformation to be performed.

Assuming the definition of T as above, we may write:

$$T(\theta, o) = (\theta \setminus (\bigcup_{\theta' \in S(\theta, o)} \theta')) \cup (\bigcup_{\theta' \in S(\theta, o)} T(\theta', o)) \tag{4.17}$$

Since l is a measure on θ, by countable additivity, we have:

$$\begin{aligned} l(T(\theta, o), s, t) &= l(\theta, s, t) \; - \sum_{\theta' \in S(\theta, o)} l(\theta', s, t) \; + \sum_{\theta' \in S(\theta, o)} l(T(\theta', o), s, t) \\ &= l(\theta, s, t) \; + \sum_{\theta' \in S(\theta, o)} l(T(\theta', o), s, t) - l(\theta', s, t) \end{aligned} \tag{4.18}$$

Now, we may consider $l(T(\theta', o), s, t) - l(\theta', s, t)$ for arbitrary $\theta' \in S(\theta, o)$. Let $o' \in o$ be the object that θ' collides with. Then, we will have:

$$C(o') \; \geq \; l(T(\theta', o), s, t) \; \geq \; l(T(\theta', o), s, t) - l(\theta', s, t) \tag{4.19}$$

Hence, we have from 4.18, 4.19:

$$l(\theta, s, t) + \sum_{o' \in o} \mathbb{1}_{o'}(\theta) * C(o') \;\geq\; l(T(\theta, o), s, t) \geq l(\theta_{opt}, s, t) \tag{4.20}$$

From 4.20 and 4.11, then directly follows (3). □

Using the definition of c_{coll} from 4.11, we can thus define c to be:

$$c(\theta, o, s, t) = l(\theta, s, t) \;+\; c_{coll}(\theta, o, s, t) = l(\theta, s, t) \;+\; \sum_{o' \in o} \mathbb{1}_{o'}(\theta) * C(o') \tag{4.21}$$

Notice that this cost formulation, unlike [23, 72, 12] does not contain a collision scaling factor, which these methods utilise to tailor their costs toward a particular planning problem. Hence, the formulation 4.21 allows us to optimise independently of scene or path scaling. In particular, the consequence of 4.21 is that we can now condition neural networks on scenes, rather than having to perform optimisation per scene.

4.2 Parameterisation

In the previous section, we describe a cost function c which we can use to optimise a planning function f_{plan}. In this section, we consider a possible parameterisation of Θ so that c is differentiable and we may optimise f_{plan} using stochastic gradient descent.

As noted in Chapter 1, we are aiming to train path regression networks. In contrast to iterative approaches such as [10, 11], which require input from the previous agent state $s_t \in \mathbb{R}^d$ to infer the next state $s_{t+1} \in \mathbb{R}^d$, we instead use f_{plan} to predict paths from the start configuration to the goal configuration in one inference step. This way, we can ensure a good f_{plan} inference speed both at train and at test time. While letting $\theta \in \Theta$ be a fixed-sized unrolling of states in the task space is possible, this would require a large cardinality of θ, for f_{plan} to be able to express complex smooth paths. This approach in practice is hard to optimise and would incur significant inference speed penalties.

As with other works [73, 74, 75, 76, 77, 78], we consider a parameterisation where:

$$\theta_{OPT} \;=\; \{(p_k, w_k) \mid \; p_k \in \mathbb{R}^d, w_k \in [0, 1], k \in \{1, 2, ..., n\}\} \tag{4.22}$$

and $n \in \mathbb{N}_{>0}$ is a problem specific task complexity parameter. With such parameterisation, we can use θ_{OPT} to define a path in the form of a non-uniform rational B-spline (NURBS) (see 2.2) with control points p_k, control point weights w_k, a default open-uniform knot vector to anchor the spline in the start and goal configurations, and a degree parameter $p \in \mathbb{N}_{>0}$. In practice, having $p > 1$ is sufficient for most planning setups, but note that one must ensure that $n > p$ for the NURBS to be defined.

The benefit of using NURBS to parameterise our paths is threefold. Firstly, NURBS provide us with an easy way to interpolate low cardinality θ to yield complex paths. This is essential to easily optimise f_{plan} as it reduces its output dimensionality with respect to fixed-sized unrolling of agent states. Secondly, the weight parameters w_k allow f_{plan} to readily reduce its output dimensionality by setting any of them to 0. This way, should the parameter n be set too high, f_{plan} can adaptively reduce its output dimensionality for simple path planning problems. Finally, NURBS continuities ensure path smoothness which is desirable for most planning applications.

Now, for an arbitrary $\theta_{OPT} \in \Theta$, $s \in \mathbb{R}^d$, $t \in \mathbb{R}^d$, and $o \in \mathbb{P}(O)$, we show how to approximate $c(\theta, o, s, t)$ by evaluating the NURBS interpolation with a high enough sampling rate $1/s$ for each value in $B \coloneqq \{s * k \,|\, 0 \leq s * k < n - p\,,\; k \in \mathbb{N}\}$. Let $N : \Theta \times B \longrightarrow \mathbb{R}^d$ be the NURBS interpolation function.

In case of the length component $l(\theta, s, t)$, we have:

$$\begin{aligned} l_{OPT}(\theta, s, t) = \int_0^{n-p} \|N(\theta, x)\|\, dx &= \lim_{\delta x \to 0} \sum_{x=0}^{n-p} \|N(\theta, x)\|\, \delta x \\ &\approx \sum_{x \in \{0, s, 2s, ...\}}^{n-p-s} \|N(\theta, x + s) - N(\theta, x)\| \end{aligned} \tag{4.23}$$

In case of the collision component $c_{coll}(\theta, o, s, t)$, we first define an object selector function $\tau : \mathbb{P}(O) \times \mathbb{R}^d \longrightarrow O$ as:

$$\tau(o, p) = \arg\min_{o' \in o} SDF(o', p) \tag{4.24}$$

and a point cost $c_p : \mathbb{R}^d \times \mathbb{P}(O) \times \Theta \longrightarrow \mathbb{R}_{\geq 0}$ function as:

$$c_p(p, o, \theta) = \begin{cases} \frac{C(\tau(o,p))}{\Delta(p,o,\theta)} & \text{SDF}(\tau(o, p), p) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.25}$$

with $\Delta : \mathbb{R}^d \times \mathbb{P}(O) \times \Theta \longrightarrow \mathbb{N}_{>0}$ providing the number of configurations along θ which collide with the same object as a given configuration, simply defined as:

$$\Delta(p, o, \theta) = \sum_{y \in \{0, s, 2s, \ldots\}}^{n-p} \delta_{\tau(o,p)}^{\tau(o, N(\theta, y))} \tag{4.26}$$

Note that Δ is always greater than 0, due to the branching condition in c_p, as every colliding point has at least itself as a corresponding colliding point with the same object. Hence, we can finally approximate c_{coll} as:

$$\begin{aligned} c_{OPT-coll}(\theta, o, s, t) &= \sum_{o' \in o} 1_{o'}(\theta) * C(o') \\ &\approx \sum_{x \in \{0, s, 2s, \ldots\}}^{n-p} c_p(N(\theta, x), o, \theta) \end{aligned} \tag{4.27}$$

In the equations above, $SDF : O \times \mathbb{R}^d \longrightarrow \mathbb{R}$ is a differentiable signed distance function (see 2.3). Note that although the derivation of $c_{OPT-coll}$ seems complex, it can be simply seen as a counting process along the interpolated spline. Figure 4.2 demonstrates the simple idea underpinning the approximation formally described above.

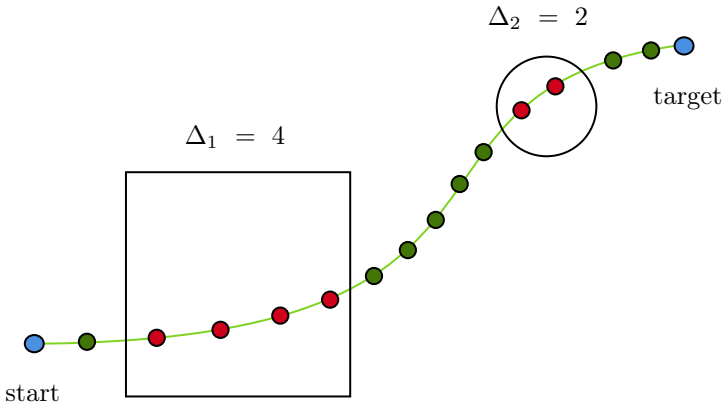


Figure 4.2: This figure illustrates the approximation of c_{coll} via $c_{OPT-coll}$. The green points along the path are non-colliding, hence their $c_p = 0$ and they do not have an effect on the collision cost. On the other hand, the red points lie within objects, hence we need to include the corresponding bounding sphere circumferences in the collision cost. The object specific Δ simply ensures that in the final $c_{OPT-coll}$ sum, an object contributes exactly only by its bounding sphere circumference.

Although we can now easily compute $c_{coll-OPT}$ using θ_{OPT} parameterisation, we can not use gradient descent to optimise f_{plan} using c just yet, as the gradients of $c_{coll-OPT}$ are undefined. To provide gradients for $c_{OPT-coll}$, we further approximate c_{coll} as:

$$\sum_{x \in \{0, s, 2s, \ldots\}}^{n-p} c_p(N(\theta, x), o, \theta) * H(\min_{o' \in o} SDF(o', N(\theta, x))) \tag{4.28}$$

$$H(x) = \frac{2}{1 + e^{x-\delta}} \tag{4.29}$$

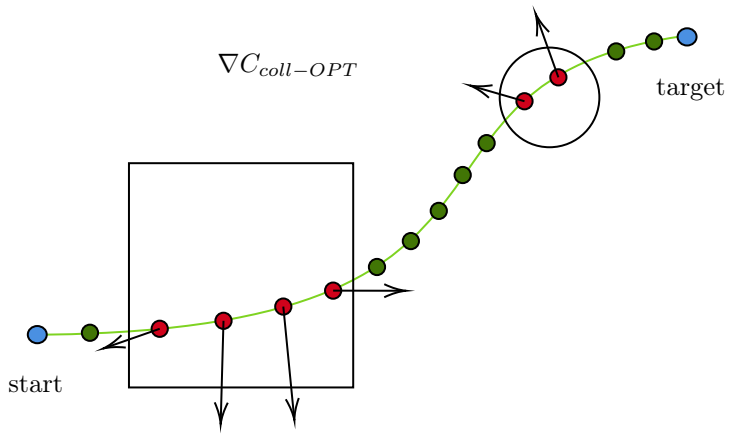


Figure 4.3: Visualisation of the gradient of our cost function's collision component based on Figure 4.2 and definition 4.28.

where $H : \mathbb{R} \longrightarrow \mathbb{R}$ is a smooth approximation of a step function and δ is a safe distance parameter, which controls the extent to which the paths should avoid the obstacles. H could in practice be any function with $H(\delta) = 1$, $\forall x \leq \delta : \ H(x) \geq 1$, and $\lim_{x\to\infty} H(x) = 0$. The intuition behind the given approximation of c_{coll} lies in the fact that C provides the scaling of the gradient that ensures obstacle avoidance, while the gradient of H directs spline points outside of objects, through the gradient of the differentiable SDF function. Please refer to Figure 4.3 for a visualisation of the gradient. The final form of our cost function, parameterised by θ_{OPT} is:

$$c_{OPT}(\theta, o, s, t) = l_{OPT}(\theta, s, t) \ + \ c_{OPT-coll}(\theta, o, s, t) \tag{4.30}$$

Now, for an arbitrary planning task, we can train f_{plan} using the cost function c_{OPT} via gradient descent. Figure 4.4 captures how the path might be transformed after a single gradient step.

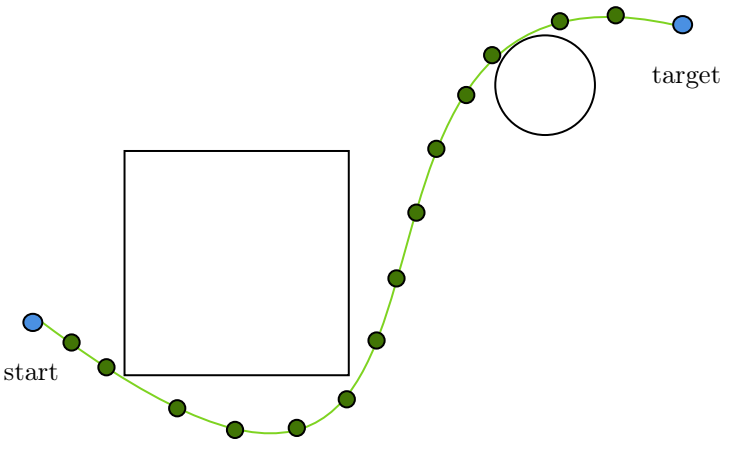


Figure 4.4: Visualisation of a possible path obtained from a gradient step based on Figure 4.3. Notice that the visualised path after taking a gradient step is not globally optimal, as there is a shorter, direct path around the sphere. In general, to obtain globally optimal paths we rely on the stochasticity of SGD.

Note that the use of SDF gradients for planning is not a novel idea in itself, for example, [23] successfully leveraged these gradients to perform optimisation for path planning. This section notably introduces a differentiable parameterisation θ_{OPT}, which can be used to perform regression planning, which innovates the iterative planning approach deep learning methods employ.

4.3 General optimisation process

In this section, we dive into some of the specifics of the deep learning setup we use in Chapter 5. In particular, we consider our general training procedure, and we further consider how to efficiently sample training data.

4.3.1 General network architecture

The general network architecture we use in our experiments is depicted in Figure 4.5. Note that we tailor both the input network, the backbone, and the fully-connected network to the planning dataset at hand. For example, if we plan based on images, we might use a CNN in place of the backbone network, otherwise, a simple fully-connected network might suffice.

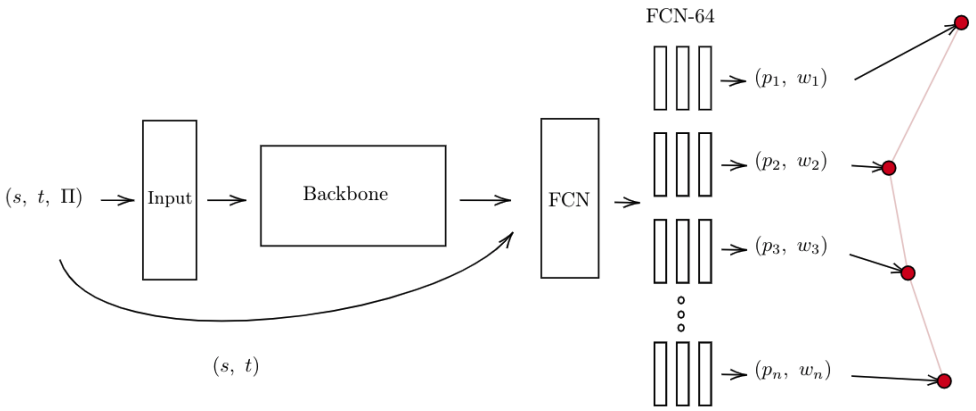


Figure 4.5: General network architecture used in our experiments.

Our general planning optimisation process is thus as follows:

1. Use start configuration s, target configuration t, and scene observation Π to obtain a set of anchors with weights from the network.
2. Use θ_{OPT} to perform NURBS interpolation, and compute the cost c_{OPT} with respect to ground truth scene knowledge O.
3. Use backpropagation to update the weights of the network based on the gradients of c_{OPT}.

4.3.2 Training considerations

Our cost formulation (4.1), allows us to train neural networks for path planning without the presence of ground truth paths. While unsupervised path planning is an attractive concept, we still need to ensure that we can generate useful data distributions for training. In particular, assume that we are given a training dataset $D = (\Pi^{(i)}, O^{(i)})$ of scene observations $\Pi^{(i)}$ and ground truth scene descriptions $O^{(i)}$. In practice, $\Pi^{(i)}$ could, for example, be represented by a scene depth map, a point cloud, or vectorised scene descriptors. For $O^{(i)}$, we could have an analytical object parameterisation, voxel grids, or possibly point clouds. As long as $\Pi^{(i)}$ contains enough information for f_{plan} perform planning, and $O^{(i)}$ allows us to compute path signed distance functions, the representations do not affect our method. However, there are several additional considerations we must take into account:

1. **Consideration 1** - As we generate training locations $(s,\ t)$ for examples of $(\Pi^{(i)}, O^{(i)})$, one consideration is what the breakdown between examples where a straight line, start-to-target path collides with objects in $O^{(i)}$ (colliding path example), and where a straight-line path is the optimal collision-free path (non-colliding path example) should be. Having a distribution where one strongly dominates the other can introduce biases in our training process.
2. **Consideration 2** - Another consideration we must make is how to ensure that the examples of $(s,\ t)$ locations for $(\Pi^{(i)}, O^{(i)})$ can be generated in a way which enables f_{plan} to learn

useful things about path planning. As an example, consider a parameterisation where $\Pi^{(i)}$ is represented by a scene depth image. In such cases, generating examples of (s, t) locations that are behind objects in $\Pi^{(i)}$ or that are not in the image $\Pi^{(i)}$ will make it difficult to train f_{plan}, as there is no useful information it can use to learn how to plan.

3. **Consideration 3** - The final consideration we must make is how to ensure that we can generate training data fast, in order to make train time dataset generation feasible. In particular, if we decide on a breakdown between non-colliding path examples and colliding path examples, we need to ensure that our pipeline allows for the given breakdown while remaining fast overall.

Consideration 1

Based on the experiments we perform in Chapter 5, we found that a 50/50 breakdown between non-colliding path examples and colliding path examples produces desirable paths. If we bias the generation strongly toward non-colliding path examples, f_{plan} tends to learn to predict paths which are either very 'risky' in terms of collisions or directly collide with objects in $O^{(i)}$. On the other hand, biasing the examples strongly toward colliding path examples causes f_{plan} to generate overly 'careful' paths, i.e., paths which are longer than we would like. A close to 50/50 breakdown proved to be sufficient to address the aforementioned biases. Please refer to Figure 4.6 for a visualisation of the desired breakdown.

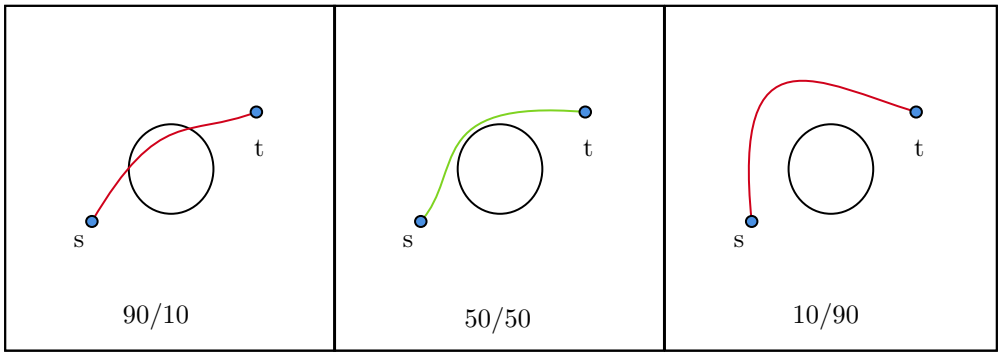


Figure 4.6: Demonstration of how training with a strong sampling bias can affect the paths that f_{plan} learns to predict

Consideration 2

In our Chapter 5 experiments, we always enforce start and target feasibility with respect to the planning problem at hand, given by $\Pi^{(i)}$. That is, for a given planning domain, we also specify a set of constraints on the generated start and target locations. For example, in terms of planning from depth images, the constraints are simply that both start and target locations should be visible by the depth camera. In practice, these constraints are simply implemented as domain-specific (s, t) example feasibility checks, and hence we only generate (s, t) examples that satisfy these checks. Figure 4.7 demonstrates an example (s, t) feasibility in the domain of planning from depth images.

Consideration 3

Once we decide on the desired breakdown p_B between colliding path examples and non-colliding path examples based on **Consideration 1**, and a set of constraints based on **Consideration 2**, we need to define a procedure that will allow us to generate start/target locations fast, in order to generate them on the fly, during training. To illustrate a possible challenge in this goal, consider the objects presented in Figure 4.8, which illustrates two scenes from our synthetic depth image dataset. As Figure 4.8 demonstrates, there exist examples which are difficult to generate both colliding path examples and non-colliding path examples for. This phenomenon arises simply due to the inherent geometries of the considered scenes. In practice, if we would wish to enforce a hard

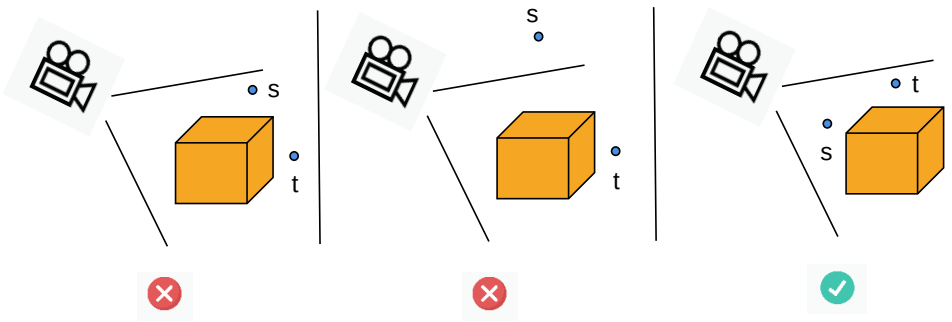


Figure 4.7: This figure illustrates examples of generated start/target locations and their feasibility with respect to the depth camera capturing the scene. On the left, the example is infeasible because it lies behind the object with respect to the depth camera. In the middle, the example is infeasible because it does not lie in the camera frame. On the right, we have a valid example, which we can use to train f_{plan}.

p_B breakdown per-batch, the point generation process becomes too slow to train networks for a reasonable number of gradient steps.

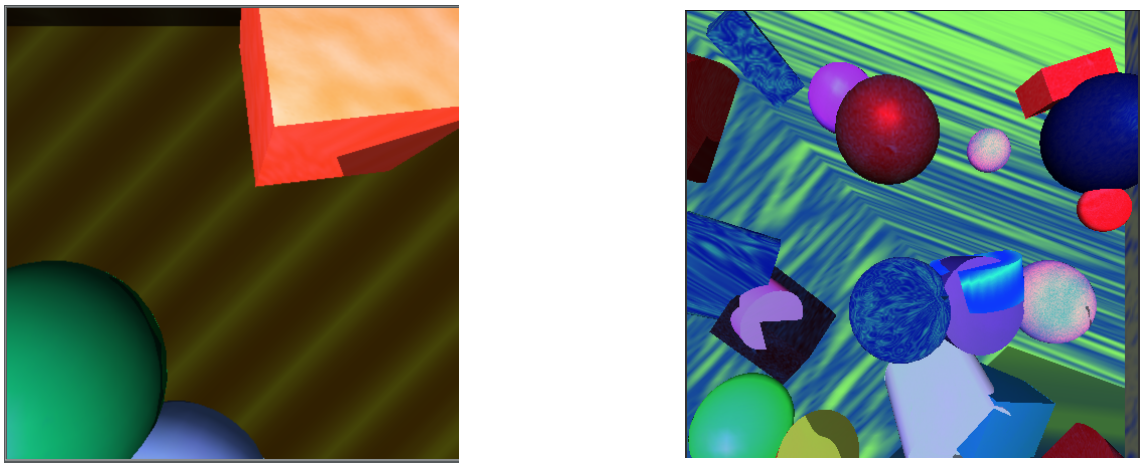


Figure 4.8: On left, we have an example scene for which it proves to be slow to generate colliding path examples, while on the right, we have an example scene for which it is difficult to generate non-colliding path examples.

To solve the problem described above, we generate both colliding path examples and non-colliding path examples using a Monte Carlo approach. This means that if we cannot generate a valid colliding path example or non-colliding path example after a set number of tries, we simply fall-back and generate an example of the opposing kind. During the generation process, we keep track of a running mean of the example breakdown called p_R, and we aim to generate each new batch so it adjusts the running mean toward the desired p_B. This way, although per-batch, we do not expect a perfect desired example breakdowns, across the whole training dataset, we can get $p_R \approx p_B$.

While our Monte Carlo method helps to ensure that our data generation pipeline does not get stuck generating examples for too long, we can further speed up our training by implementing a fast colliding path example generating routine. Instead of naively sampling start and target configurations in the scene, and checking for collisions, we uniformly sample two intermediate positions in the scene which lie in obstacles given by $O^{(i)}$. Then, we sample start and target locations along the line connecting the intermediate positions, as illustrated in Figure 4.9. This method further enhances our data generation pipeline. Now, we can comfortably iterate on our networks without significant data generation bottlenecks.

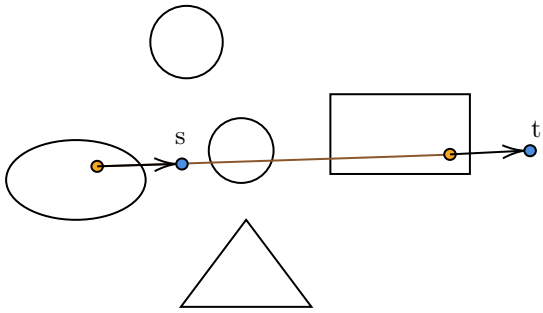


Figure 4.9: This figure captures the colliding path example generation procedure, which we use to enhance a naive approach. The intermediate points (orange) are uniformly sampled within obstacles, while the start/target locations (blue) are sampled along the straight-line path between the intermediate locations.

In this section, we introduce several considerations for our training process. While **Consideration 1** and **Consideration 2** ensure that we can train f_{plan} on sensible examples, **Consideration 3** ensures that we can get f_{plan} to converge in a timely fashion. All the considerations combined allow us to train f_{plan} in diverse settings, as we further demonstrate in Chapter 5.

4.4 Summary

This Chapter 4 introduces the two key innovations our work brings:

1. Section 4.1 focuses on formulating the core ideas behind our **scale-invariant path planning cost function** consisting of length and collision terms. Now that we removed the need for a scaling factor as in [23, 12], we can use a deep network to represent f_{plan} and train it **unsupervised** on diverse scenes, without the need of altering scene-specific parameters or generating ground truth oracle paths.

2. Section 4.2 introduces θ_{OPT}, a low-dimensional path representation allowing for **direct regression planning**. That is, planning from a start to target configuration in a single inference step. We further show how to use a θ_{OPT} parameterisation to approximate the cost function formulated in Section 4.1, so that our parameterised cost is differentiable and can be optimised using stochastic gradient descent.

3. Section 4.3 introduces a general network architecture that we use to train path planning networks. In Chapter 5, we explain how to tailor the architecture to the planning datasets at hand. Further, section 4.3 also discusses the training considerations we should follow during our experiments.

In Chapter 5, we compare our approach introduced in this chapter to state-of-the-art planners in various contexts.

Chapter 5

Evaluation

In this chapter we evaluate the ability of our approach introduced in Chapter 4 to plan in various settings. Our goal is to answer the following questions about our approach:

1. Does cost function scale invariance matter in practice?
2. Can our method generalise to plan in environments where Π captures the full state of a scene? If so, how does it compare to state-of-the-art planners?
3. Can our method generalise to plan in environments where Π is only a partial observation?
4. How does our method cope with maze-like environments, where traditionally node-based planners perform well?

An additional implicit goal is to highlight the ability of our method to plan in diverse environments. While the above list is by no means all-inclusive, applying our method in even more domains is a promising avenue for future work.

5.1 Objective comparisons

In this section, we assess the c_{OPT} scale invariance compared to the CHOMP collision objective [23], which represents a suitable gradient-based motion planning benchmark. This comparison allows us to examine whether cost function scale invariance matters in practice. While we opted to compare to the CHOMP collision objective, we could have picked objectives of other stochastic optimisation-based planning approaches, such as ITOMP[61] or STOMP[60], as they all require pre-set collision penalties. Note that CHOMP's cost is made of terms measuring path smoothness and collision costs. While we measure path lengths instead of smoothness terms, these two terms are directly comparable, as both of them need to be dominated by the collision term in the optimisation process. For a single sample point $x \in \mathbb{R}^d$ in the task space $o \in \mathbb{P}(O)$, the CHOMP collision term is as follows, where ε is a calibrated constant:

$$c_{coll-CHOMP}(x, o) = \begin{cases} -SDF(o, x) + \frac{1}{2}\varepsilon & \text{if } SDF(o, x) < 0 \\ \frac{1}{2\varepsilon}(SDF(o, x) - \varepsilon)^2 & \text{if } 0 < SDF(o, x) \leqslant \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

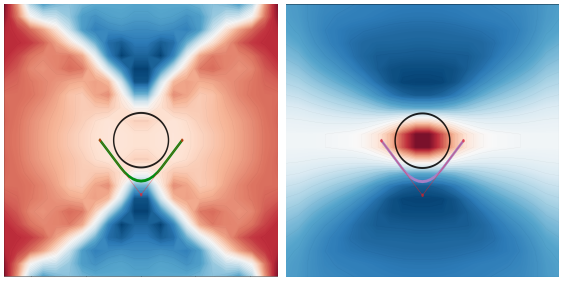
CHOMP is originally applied to robot manipulators, and while our collision cost can also directly apply to robot manipulators without any loss of generality, we choose to compare the cost functions on simple primitive shapes to make brute-force optimisation tractable for comparisons. As a first step, we calibrate λ in the CHOMP cost as defined in [23] for a simple sphere problem in our domain, as seen in Figure 5.1a. We perform this calibration so that the optimal CHOMP objective path is collision-free, with an equal length to our objective's optimal path. Then, we randomly sample a rectangle and a sphere in a 2D scene, together with a start and target position, such that a straight line path would collide with either of the objects. In this setup, as illustrated in Table 5.1, out of 150 sampled planning problems, c_{OPT} achieves a 100% success rate, while

the calibrated CHOMP objective achieves a 79.33% success rate, and an uncalibrated CHOMP objective, with default $\lambda = 1$ achieves a 40.66% success rate. These results underline our objective's innate ability for generalisation across different object scales and demonstrate that scale invariance does indeed matter in pursuit of scene generalisation.

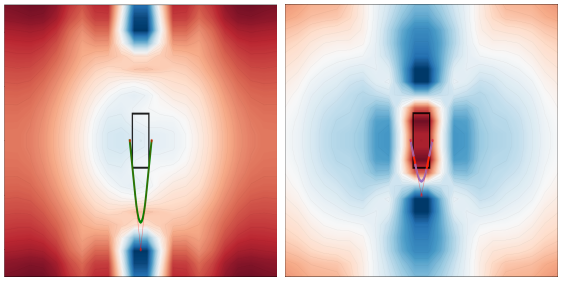
Table 5.1: Quantitative comparisons with the CHOMP objective on 150 simple 2D problems

Method	**Success rate**
CHOMP collision objective (uncalibrated)	40.66%
CHOMP collision objective (calibrated)	79.33%
Ours	**100.0**%

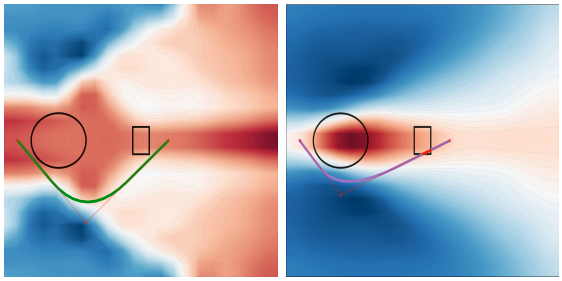
In general, the need to calibrate the CHOMP objective is a direct consequence of the **global optima property (3)**, which our objective satisfies by definition. Although the calibrated CHOMP objective performs reasonably well on these simple examples, the calibration process relies on our ability to cherry-pick difficult examples from the dataset to calibrate on, as the **global optima property (3)** needs to be satisfied across all examples in a dataset. However, without the inclusion of an object-specific size parameter in the loss function, CHOMP path lengths are necessarily compromised on *easier* examples when calibrating for the *hardest*, to guarantee no collisions across the entire dataset. For these reasons, the CHOMP objective needs to be calibrated per scene, while as we further demonstrate in experiments 5.2, 5.3, 5.4, our formulation generalises across diverse scenes and planning setups with no need for any calibration. Figure 5.1 presents examples of planning problems where our objective outperforms that of CHOMP calibrated on the example from Figure 5.1a.



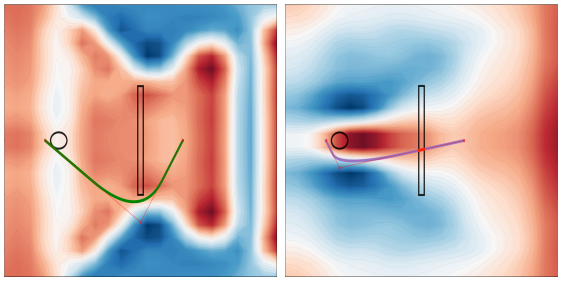
(a) Sphere planning problem that was used to calibrate the CHOMP objective.



(b) Failure case which yields collisions for the calibrated CHOMP objective.



(c) c_{OPT} successfully avoids rectangle corners.



(d) c_{OPT} successfully avoids thin rectangles.

Figure 5.1: In each image pair, we have on the left-hand side the result of optimising c_{OPT} (green), while on the right-hand the result of optimising the CHOMP collision objective (purple). In all examples, we use a single control point NURBS parameterisation. The heat maps in the background represent the values of the cost function at different control point positions, with red regions being the maxima, and blue regions being the minima. We observe that in all the cases, the paths obtained by optimising c_{OPT} are collision-free with shortest possible length for the given number of control points.

5.2 Continuous full-state planning

Continuous planning describes a setting, where planners plan in continuous spaces, rather than discretised scenes. Our formulation of f_{plan} is by nature continuous, as it does not assume a graph scene representation or a discrete action space. In this section, our goal is to assess the ability of f_{plan} to generalise over scenes, when a full description of the scene is captured in Π. This setting most directly resembles that of the learning-based planners described in Section 3.2.2.

To successfully evaluate our method, we have to decide on a viable dataset to perform planning and comparisons on. As explained in Section 3.2.2, Motion Planning Networks[11] represent a current state-of-the-art learning-based planner in continuous domains. To reiterate, in the case of MPNet's Complex 3D dataset, the MPNet network is trained on 100 scenes with 4000 ground truth paths each, generated using RRT*.

The MPNet method further strives to establish a probabilistically complete planner, so they develop several path optimisation strategies on top of the network output, which are not learning-based. The two main additional non-learning based methods MPNet uses are lazy state contraction (lsc) and replanning. In a replanning step, the MPNet method attempts to correct colliding path segments by regenerating each of them either using MPNet itself, or an oracle planner such as RRT*. In lazy state contraction (lsc), the algorithm greedily tries to connect non-consecutive path segments, if these connections are collision-free, this way pruning extraneous states which could be colliding. MPNet being a state-of-the-art continuous method using RRT* supervision and iterative planning presents a suitable baseline for our method. Figure 5.2 presents an example of a path inferred by the MPNet method on the Complex 3D dataset.

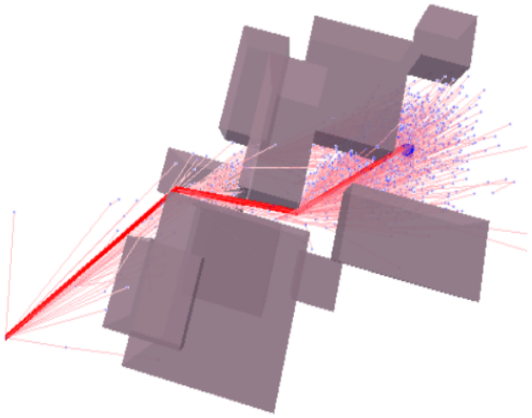


Figure 5.2: Example of a path found using MPNet[11] on the Complex 3D dataset.

5.2.1 Data generation

As mentioned above, MPNet's Complex 3D dataset presents a suitable option for full-state continuous planning. In a single Complex 3D scene, there are 10 randomly placed cuboids embedded in a scene of size 20. The MPNet method's network is trained using a point cloud scene representations, and while we could have opted for the same scene representation, we instead train f_{plan} on $\Pi \in \mathbb{R}^{10\times 6}$ vectorised scene descriptors, as each scene has 10 axis-aligned boxes with their translation and dimensions. We randomly sample Π in a scene of size 20, with each dimension of each of the boxes being either 5 or 10, just as in the Complex 3D training set. Further, to train f_{plan}, we randomly sample start and target configurations $s, t \in \mathbb{R}^3$ so that there is a 50/50 breakdown between examples which would or would not collide by simply following a straight-line path, as explained in Section 4.3.2. Both of the aforementioned samplings are done during training time, which demonstrates the flexibility of our method. Batched analytic SDFs are very simple to compute for primitive boxes, and thus we can evaluate our cost function while keeping GPU training tractable.

5.2.2 Approach setup

Now that we can sample the training data, we have to decide on a suitable architecture for f_{plan} and set the θ_{OPT} and c_{OPT} parameters. In terms of our architecture, we tailor the architecture described in Section 4.3.1 as in Table 5.6.

Network	Continuous full-state planning
Input	MLP - 2 layers, width 128
Backbone	Highway layers[26] - 10 layers, width 256
FCN	MLP - 3 layers, width 256

Table 5.2: Tailored components of the general network for the continuous planning task.

The choice of Highway layers for our Backbone as seen in Table 5.6 was simply due to the fact that we observed slightly improved performance with increasing depth. Highway layers show improved training capabilities[26] compared to standard dense layers when it comes to very deep neural networks (see 2.1). We further set the parameters of c_{OPT} and θ_{OPT} to $s = 0.05$, $p = 2$, $\delta = 5$, and $n = 10$. We observed that a $s = 0.05$ sampling step with a degree $p = 2$ NURBS parameterisation provides us with dense enough sampling to avoid fine corners. Note that although we train using $n = 10$ control points, f_{plan} does not have to use all of them, as it has a finer-grained control via the weights of each control point.

A particular detail we have not touched on yet is the inclusion of scene boundaries in our approach. In our early experiments, we noticed that our method quickly learns to avoid obstacles by simply learning the obstacle sampling distribution boundaries, and trivially avoiding the obstacles all-together. For this reason, most datasets have a set scene size, which prevents such maneuvers. We include scene boundaries by introducing shapes with so-called *inverted SDFs*. That is, we embed shapes in our cost computation which mimic the boundaries of the scene, but collisions occur when a particular path segment escapes the *inverted SDF* shape. This way, our method learns paths that respect the boundaries of the scenes that they plan in. The Complex 3D *inverted SDF* shape is a simple cube of dimensions $20 \times 20 \times 20$.

5.2.3 Path correction

The purpose of our comparison with MPNet is primarily to compare MPNet's learning-based component to our approach. As we explain at the beginning of this section, there are multiple additional techniques MPNet employs to refine paths, however, those do not reflect the network's capacity for quality path planning, as they are not used during the training of the MPNet network. Hence, the most direct comparison with MPNet is comparing purely the learning-based components with our implementation of f_{plan}.

As an extension, apart from the main comparison objective, we further implement a simple path correction procedure to compare with the inference speed of the algorithmic corrections of MPNet. In a single correction step, for an arbitrary colliding path segment, we greedily assign pairs of path points which would avoid the specific obstacle our segment collides with. We achieve this by measuring the offsets necessary for obstacle avoidance for the given colliding cuboid. The visualisation of this simple approach is demonstrated in Figure 5.3 in a two-dimensional planning case.

5.2.4 Results

Table 5.3 and Table 5.4 present the performance of each method on 2000 planning problems from the unseen Complex 3D test set. In our experiment, we measure the rate of collision-free paths, the length of the successfully predicted paths with respect to the RRT* ground truth length, and the planner's inference speed. MPNet without lazy state contraction represents the most directly comparable method, as it solely uses the output of the MPNet for planning. As seen in Table 5.3, our method outperforms the learning-based component of MPNet for an arbitrary number of MPNet's replanning attempts in terms of all measured metrics. The lazy state contraction algorithm significantly improves the performance of MPNet, as illustrated in Table 5.4. However, our method combined with the aforementioned greedy segment correction still provides a better success rate. Furthermore, our entire training time, including the procedural generation of both

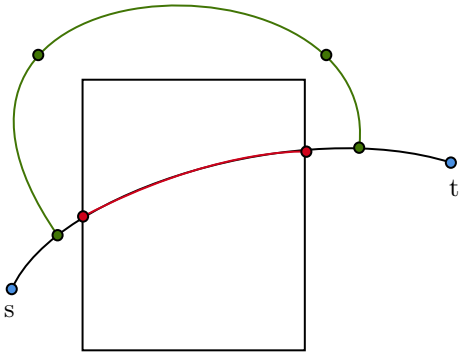


Figure 5.3: This figure illustrates the simple path correction algorithm we implement to compare with MPNet's algorithmic corrections. The red segment represents our path's collision, while the new green segment is the corrected segment obtained via our algorithm.

scenes and training targets, is only 10 hours. Assuming it takes 7 seconds to generate each ground truth path with RRT*, which is the time quoted on the Complex 3D dataset [11], MPNet's data generation process takes over a month of computing time. This is an 80× speedup over MPNet's total training time, including ground-truth dataset generation.

Table 5.3: Quantitative comparisons with MPNet on the Complex 3D w/o lazy state contraction

Method	**Success rate**	**Path length / RRT***	**Inference speed**
MPNet (0 replan)	34.7%	1.996	6.3ms
MPNet (1 replan)	42.8%	2.21	14ms
MPNet (2 replan)	45.7%	2.354	31.8ms
Ours (0 corrections)	**76.2%**	**1.947**	**1.35ms**

Table 5.4: Quantitative comparisons with MPNet on the Complex 3D w/ lazy state contraction

Method	**Success rate**	**Path length / RRT***	**Inference speed**
MPNet (0 replan, lsc)	75.2%	**1.077**	7.9ms
MPNet (1 replan, lsc)	80.3%	1.128	10.7ms
MPNet (2 replan, lsc)	85.7%	1.15	13ms
Ours (1 correction)	**86.4%**	1.948	**7.05ms**

For our next quantitative experiment, we train f_{plan} on an increased scene size of 60. The effect of this change is that there now exist more *trivial* solutions to the planning problem by simply going around multiple objects at a time. For this reason, we set $n = 3$, while keeping the other parameters the same as for the scene size 20 comparison. Unfortunately, we were unable to collect results about MPNet's performance in this setting. Table 5.5 captures the results of our method on 2000 planning problems from the unseen Complex 3D test set. We observe that our method experiences improvements in all of its metrics, due to the simplified planning setup.

Table 5.5: Quantitative results of our methods on the Complex 3D w/ scene size 60

Method	**Success rate**	**Path length / RRT***	**Inference speed**
Ours (0 corrections)	86.1%	**1.466**	**0.95ms**
Ours (1 correction)	**91.45%**	1.487	1.8ms

The results of the experiment presented in this section demonstrate the ability of f_{plan} to generalise across a diverse set of planning settings, given full-state scene descriptions. Further, we

showed that f_{plan} is capable of planning with metrics on par with state-of-the-art learning-based continuous path planners. In the following section, we examine the ability of f_{plan} to plan in cases, where we do not have access to the full scene description.

5.3 Planning from Images

Motivated by the fact that robots are never equipped with oracle knowledge, but rather with partial observations via sensors, we also train f_{plan} to predict collision-free paths conditioned on depth images of table-top-like scenes. The potential application of this approach is in the context of manipulators, which could perform direct depth image to collision-free path regression, without the need of intermediate scene reconstruction steps.

In section 5.2 we saw that f_{plan} is capable of planning from full-state scene descriptors. In this section, an additional goal is to show that we can change the architecture and the conditioning of f_{plan}, and still maintain high-quality paths. If we succeed in this experiment, we will demonstrate the general applicability of our method in drastically different planning contexts.

5.3.1 Data generation

We generated the table-top depth image dataset using CoppeliaSim[79], by randomly placing floating cuboids, cylinders, and spheres such that they intersect with the ground plane of a large bounding cuboid. They are also permitted to intersect with each other. We randomised camera positions and focal lengths for each image, with a bias to face towards the ground plane, where the objects are spawned. Analytic SDFs for these primitive shapes are also simple to compute in batch on the GPU. Figure 5.4 illustrates several examples of the synthetic scenes we can obtain using the aforementioned data generation process for more challenging synthetic table-top-like scenes.

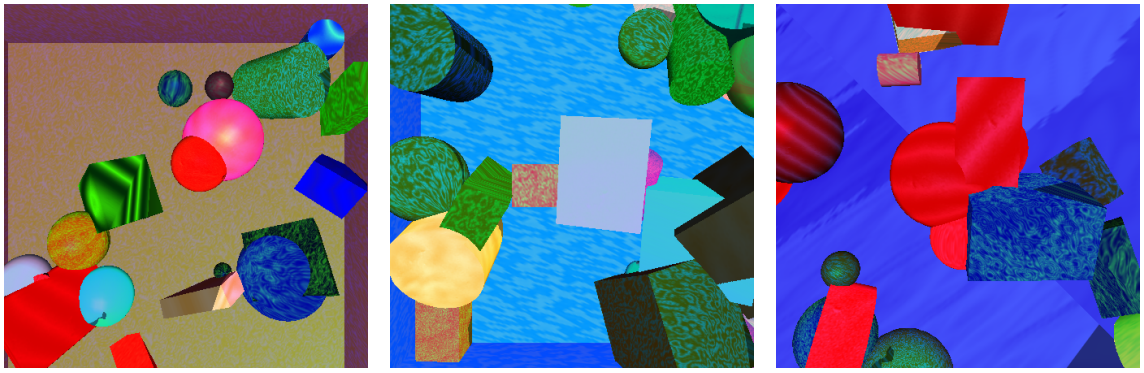


Figure 5.4: Examples of challenging synthetic scenes obtained from our scene generation process.

Similarly as in Section 5.2, to train f_{plan}, we randomly sample start and target configurations $s, t \in \mathbb{R}^3$ using O, so that there is a 50/50 breakdown between examples which would or would not collide by simply following a straight line path, as explained in Section 4.3.2. Note that each depth image $\Pi \in \mathbb{R}_{>0}^{448\times448}$ is taken using a camera with a calibration matrix $C \in \mathbb{R}^{3\times3}$ and an extrinsic camera matrix $E \in \mathbb{R}^{4\times4}$. Hence, we do not directly condition the network on s and t, as they are sampled in the world frame, but rather we transform both s and t to the camera frame. We achieve this simply by multiplying with the extrinsic matrix as $s^{(c)} = Es^{(h)}$ and $t^{(c)} = Et^{(h)}$, where $s^{(h)}, t^{(h)} \in \mathbb{R}^4$ are homogeneous coordinates equivalent with s and t, respectively. In practice, we convert $s^{(c)}$ and $t^{(c)}$ back to Cartesian coordinates, to simplify the input to f_{plan}.

5.3.2 Approach setup

Architecture and θ setup

Similarly as in Section 5.2, we tailor the architecture described in Section 4.3.1 as in Table 5.6.

The choice of Conv3, 2x as the input part of the network is mainly to downsample the high resolution depth images that we are using to train f_{plan}. In terms of the backbones, we also experimented with VGG-16[27] and MobileNet[80], but ResNet-50[28] proved to provide the best

Network	Planning from images
Input	Conv3, 2x
Backbone	ResNet-50 [28]
FCN	MLP - 4 layer, width 256

Table 5.6: Tailored components of the general network for the planning from images task.

performance. We further set the parameters of c_{OPT} and θ_{OPT} to $n = 3$, $s = 0.05$, $p = 2$, $\delta = 0$. Compared with the task described in Section 5.2, in this part, we expect to require fewer control points to find quality paths.

Note that unlike in the previous section, f_{plan} is trained to predict θ_{OPT} in the camera frame of the given scene, denoted as $\theta^{(c)}_{OPT}$. Since we store O in the world frame, $\theta^{(c)}_{OPT}$ is then converted to the world frame for SDF computations.

Network conditioning

Although we could directly condition the network as presented in Section 4.3.1, based on our experiments, for planning from images, it is preferable to transform $(\Pi, s^{(c)}, t^{(c)})$ so that for every coordinate in Π, we have some information about where $s^{(c)}$ and $t^{(c)}$ are. For this reason, we transform an arbitrary pixel in the depth image Π_{ij}, to its camera frame correspondent as:

$$\Pi^{(c)}_{ij} = C^{-1}[i, j, 1]^T * \Pi_{ij} \tag{5.2}$$

Then, for each pixel we can compute offset vectors $\Delta s^{(c)}_{ij} = \Pi^{(c)}_{ij} - s^{(c)}$ and $\Delta t^{(c)}_{ij} = \Pi^{(c)}_{ij} - t^{(c)}$, which provide us with information about the direction to $s^{(c)}$ and $t^{(c)}$, respectively. Hence, when we train f_{plan}, we condition it on $(\Pi, s^{(c)}, t^{(c)})$ transformed to $(\Pi^{(c)}, \Delta s^{(c)}, \Delta t^{(c)})$.

Further, to increase the ability of f_{plan} to generalise to real-world depth images, we apply noise to the images before using them for training. We found that this approach makes the training process robust and reduces generalisation error. Figure 5.5 illustrates our depth image before and after the noise is added.

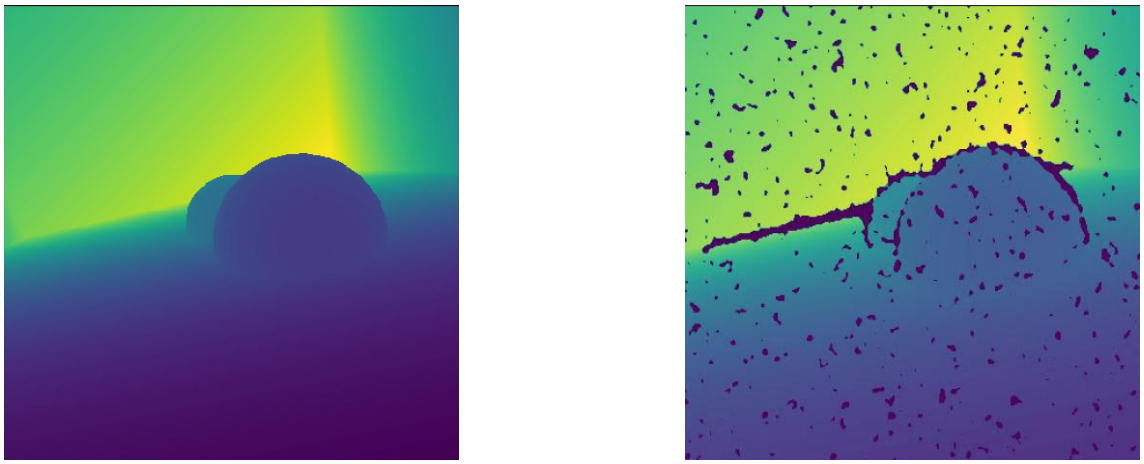


Figure 5.5: On left, we have an example depth image, while on the right, we have the same depth image with the introduced noise.

5.3.3 Results

Quantitative results

In our vision-based experiments, we focus on measuring the planning success rates on examples where we would expect a straight-line start-to-target path to collide, and path lengths on examples where we would expect a start-to-target path to be the optimal, collision-free path. On 2000 unseen examples from our synthetic dataset, our method achieves a 89.05% **success rate** on problems

where a straight-line path is expected to collide, and 1.39 **times longer than start-to-goal distance** on problems where a straight-line path is optimal.

Qualitative results

In this section, we aim to visualise some of the f_{plan} planned paths on our table-top synthetic test set and our more complex synthetic dataset. Figure 5.6 demonstrates planned paths on scenes from our synthetic test set, which we used to report the results in the **Quantitative results** section. We observe that the paths comfortably avoid obstacles while keeping reasonable path lengths.

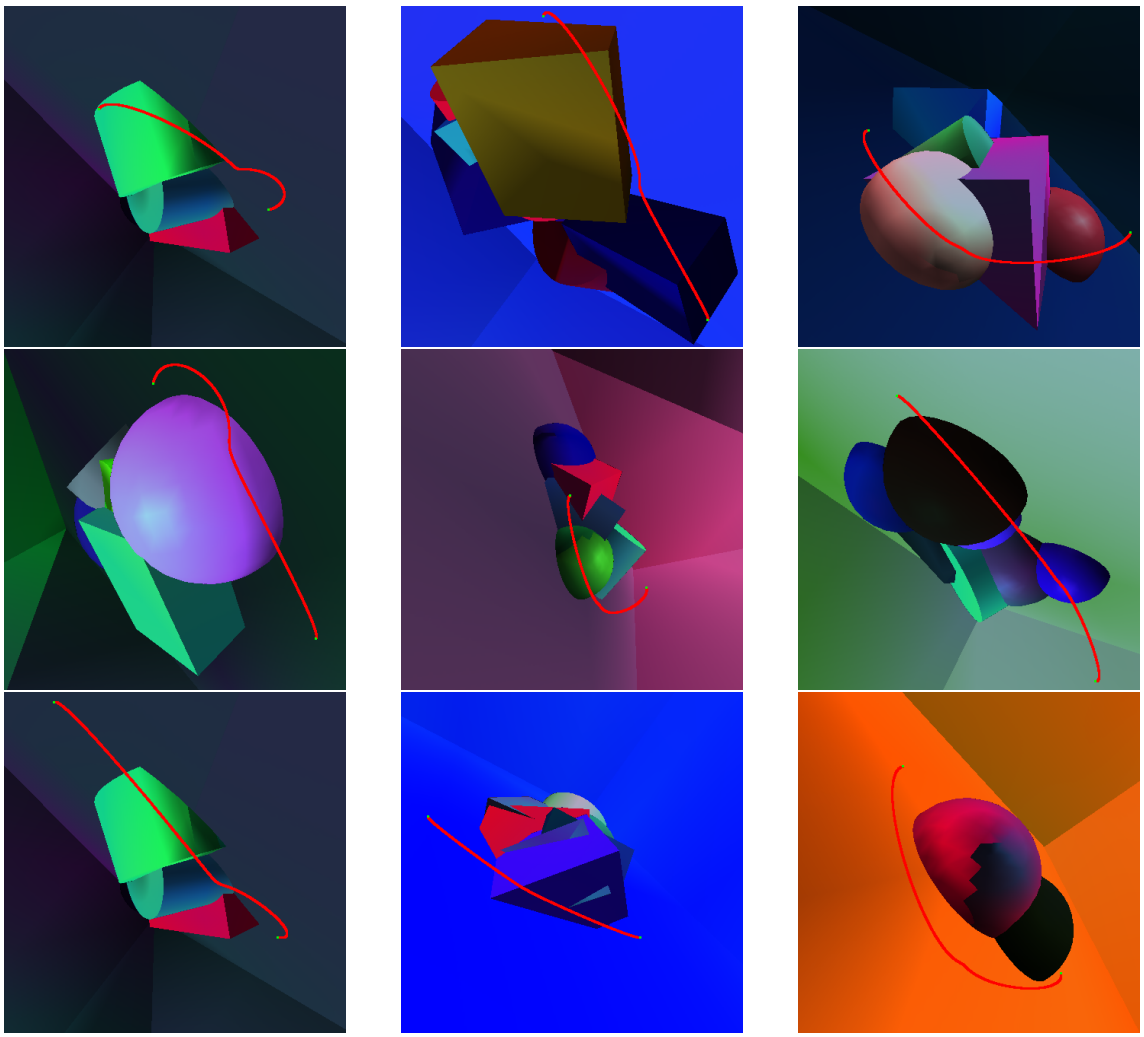


Figure 5.6: Examples of f_{plan} paths on synthetic scenes.

Figure 5.7 demonstrates planned paths on scenes from our smaller, more complex, synthetic dataset. We observe that the paths are capable of navigating more complex planning problems based on depth images alone.

Summary

The results presented above demonstrate that our method is general enough for f_{plan} to plan in contexts with partial observation. We successfully transformed the architecture of f_{plan}, together with the full planning setting, and still were able to plan high-quality paths using our regression network.

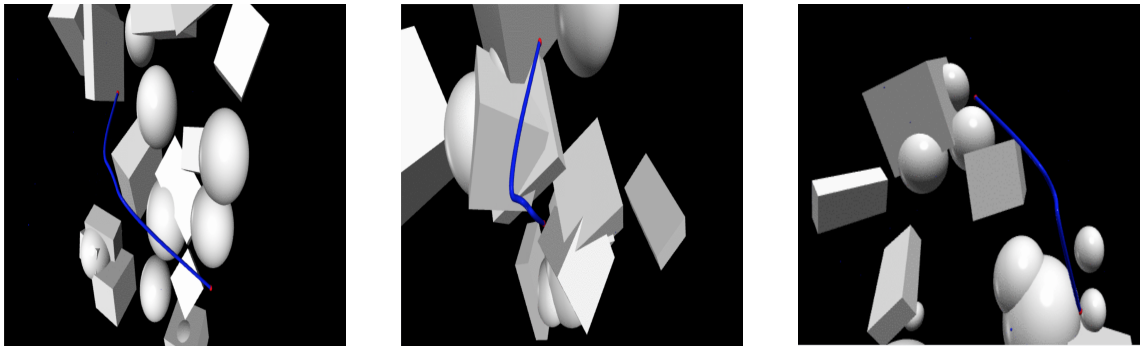


Figure 5.7: Examples of f_{plan} paths on more complex synthetic scenes.

5.4 Current limitation: maze-like environments

For our final evaluation, we consider how our method copes with maze-like environments, where discrete planners (see 3.1.1) would be typically employed. In the previous sections, we exposed our method mostly to scenes, which were suitable for fast, continuous planning. This experiment, on the other hand, allows us to assess how increasing scene complexity affects the performance of our planner.

Maze-like environments are more commonly generated in two dimensions, so we reach for two-dimensional learning-based path planning literature for comparison options. The current state-of-the-art discrete method in this domain are Value Iteration Networks (VIN)[10]. Note that while VIN focuses on discrete planning in grid-world environments, VIN is mostly an innovative network architecture, trained using Dijkstra's shortest paths. Our method is also concerned with planning, however, our main goal is to establish an unsupervised cost function. Hence, our approach and VIN's approach are in many ways orthogonal. Further, note that while VIN plans iteratively, by guiding a reactive agent from a start to a target location, we perform continuous planning in the given scene, just as in the previous sections. This is an important distinction, as VIN is known to have limitations in a continuous setting [10].

5.4.1 Data generation

VIN employs a scene generation procedure, where images are randomly filled with rectangles of random sizes. All the scenes have a default boundary, and we can control how many rectangles to generate, together with their maximal dimension. VIN is originally trained on 3 different scene sizes, 8×8, 16×16 and 28×28. For our purposes, we focus on 28×28 scenes with 3 varying complexity classes. Hence, we generate three datasets as follows:

1. **Hard** - Contains 50 random rectangles with a maximum dimension of 2.
2. **Medium** - Contains 12 random rectangles with a maximum dimension of 4.
3. **Easy** - Contains 2 random rectangles with a maximum dimension of 8.

Figure 5.8 presents an example of a **hard** VIN scene. For our purposes, we store the analytical description of each rectangle in O in order to compute the SDFs, and we represent Π just as a two dimensional binary mask, as in Figure 5.8 .

5.4.2 Approach setup

For planning from the binary masks, we tailor the architecture introduced in Section 4.3.1 similarly as in our Planning from Images 5.3 experiment. Table 5.7 captures the components of our architecture.

An important distinction to make is that in the Planning from Images 5.3 experiment, the images were projections of three-dimensional scenes, while in this two-dimensional case, the images capture the full scene information. We further set the parameters of c_{OPT} and θ_{OPT} to $s = 0.05$, $p = 2$, $\delta = 2$, and $n = 5$.

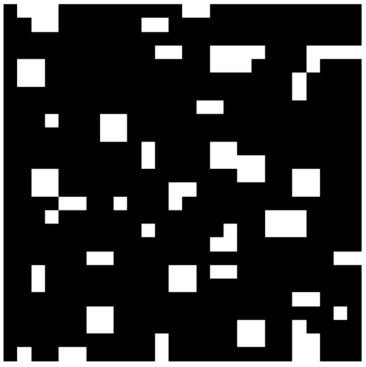


Figure 5.8: An example VIN 28×28 scene with 50 randomly placed rectangles.

Network	Planning in mazes
Input	Conv3
Backbone	ResNet-50 [28]
FCN	MLP - 4 layer, width 256

Table 5.7: Tailored components of the general network for the planning in maze-like environments.

5.4.3 Enforcing physical constraints

In the continuous VIN experiments, VIN model obstacles as objects the reactive agent can 'bounce' away from. Hence, the continuous VIN success rate is not necessarily measured as (1−collision rate) as ours is, however, it is measured as the rate at which the agent was able to reach the goal location or the average final distance of the agent from the goal. In our case, we have a somewhat inverse setup, as due to our parameterisation, we always reach obstacles. On the other hand, we do not assume continuous bouncy obstacles, but rather measure collisions and count them in our success rate metrics. To make the two methods more comparable, we implement a simple procedure to enforce physical constraints on our paths.

The procedure very much resembles the one we introduced in section 5.2, where we simply project the colliding path segments marginally beyond the boundaries of the respective colliding objects. In this case, however, we perfectly follow the contours around the colliding obstacle, this way ensuring that our projected segment does not collide with other rectangles. Now that we have such a procedure implemented, we can more reasonably compare the two methods.

5.4.4 Results

In this section, we present the results of our experiments in comparison to the discrete version of VIN on the grid-world environments as presented in [10].

Optimising per-scene

The first step in our evaluation is to see whether our method can solve a planning problem in a single maze. In the context of deep learning, this can be seen as overfitting on a single scene, however, in practice, this is how much of the optimisation-based planners infer paths. We perform this sanity check to verify that our cost function performs well in these settings. Hence, we generate a random **hard** scene together with a start and a target location. Figure 5.9 demonstrates that our method can reliably find a short, collision-free path when optimising on a single maze-like scene.

Quantitative results

As described in the data generation part 5.4.1, we randomly generated datasets for 3 environments of varying difficulty. In our comparisons, we focus on the following metrics:

1. **Collision rate** [coll rate] - Ratio of paths that collide with objects in the scene. In the case of VIN, this is always 0.

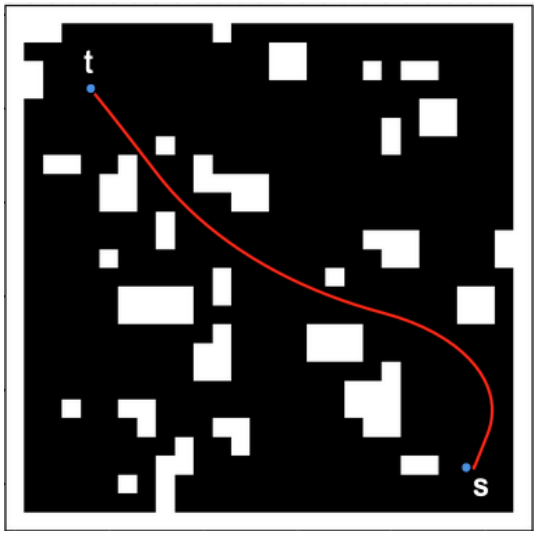


Figure 5.9: Our method reliably finds a short, collision-free path when optimising per-scene on complex two dimensional maze-like environments.

2. **Success rate** [succ rate] - Ratio of paths, in which the two methods reach the target location. In our case, this is always 1.
3. **Path length (successful)** [len succ] - Path lengths of paths that reach the goal. In our case, this are all the paths.
4. **Path length (collision-free)** [len free] - Path lengths of paths that are not colliding. In our case, we measure the length of the paths when transformed as explained in section 5.4.3.
5. **Inference speed** [speed] - Time to generate a full path.

Tables 5.8, 5.9, 5.10 present the results of both methods on the hard, medium, and easy datasets respectively. For evaluation, **Our** method sampled 2000 random planning problems randomly in previously unseen scenes. In case of **VIN**, the positions were random as well, but were selected from a 28×28 grid.

Table 5.8: Quantitative comparison of our method to discrete VIN on **hard** dataset

Method	**[coll rate]**	**[succ rate]**	**[len succ]**	**[len free]**	**[speed]**
VIN	**0%**	**100.0%**	14.85	**14.85**	58ms
Ours	72.25%	**100.0%**	**14.452**	21.89	**8.8ms**

Table 5.9: Quantitative comparison of our method to discrete VIN on **medium** dataset

Method	**[coll rate]**	**[succ rate]**	**[len succ]**	**[len free]**	**[speed]**
VIN	**0%**	**100.0%**	**14.63**	**14.63**	57ms
Ours	43.10%	**100.0%**	15.03	18.61	**8.8ms**

Table 5.10: Quantitative comparison of our method to discrete VIN on **easy** dataset

Method	**[coll rate]**	**[succ rate]**	**[len succ]**	**[len free]**	**[speed]**
VIN	**0%**	**100.0%**	**14.34**	**14.34**	56ms
Ours	11.97%	**100.0%**	14.7	15.69	**8.8ms**

Summary

The results presented in Tables 5.8, 5.9, 5.10 underline the differences between two drastically different approaches to path planning. While VIN is trained using Dijkstra shortest path supervision, performs iterative planning, and has a discrete action space, our method is trained unsupervised, performs regression planning, and has a continuous planning space. In the case of the two-dimensional discrete maze-like environments, all these factors seem to play in favour of VIN. However, note that our regression planning setup allows us to plan approximately 6.5× faster with comparable successful path lengths. Hence, in rapid planning settings where collisions are not fatal, our approach may be a preferable choice. We speculate that some weak form of supervision, or perhaps multiple planning iterations per scene, might aid our approach to perform better in a maze-like context. This exploration presents a suitable option for future work.

From the perspective of the ablation of our method on maze-like two-dimensional settings, it is clear that a reduced planning complexity does indeed improve our performance. One hardship with optimising our method on very hard planning settings is that our optimisation cost field is dependent on the scenes themselves. With more obstacles, the optimisation problem contains an increasing number of local minima. A promising avenue for future work is investigating this phenomenon by doing an ablation with various optimisers and training parameters on scenes of varying difficulty.

5.5 Common failure modes

In general, the most common failure cases occur when our start or goal positions are very close to objects. In such cases, f_{plan} needs to place the edge anchor points with additional care to ensure safe obstacle avoidance. This is a similar setting as planning in challenging grid-world environments we examined in Section 5.4, where control points need to be placed perfectly to ensure collision-free paths. This is a difficult task for a planner without iteration.

Regarding the lengths of our paths, there are two main limiting aspects. Firstly, our collision cost component has a repelling effect beyond the boundaries of objects, which slightly compromises path length in favour of safe obstacle avoidance. Secondly, a portion of the network predictions are miss-classifications for the image-based planning 5.3, where f_{plan} unnecessarily attempts to avoid an obstacle as a precaution.

5.6 Summary

In the beginning of this Chapter 5, we outlined a series of questions to aid us in assessing our method. The following conclusions can be made based on our experiments:

1. We saw in Section 5.1, that scale invariance matters in pursuit of scene wise generalisation of unsupervised path planning. Our cost function outperformed a calibrated CHOMP[23] objective by more than 20% in the process of demonstrating the general applicability of our cost.

2. In Section 5.2, we established that deep neural networks trained on full-state scene descriptions using our method can generalise to plan in unseen environments. On top of generalisation capabilities, our method outperforms recent learning-based planning methods in all measured metrics on complex 3D datasets.

3. We used Section 5.3 to show that our method is f_{plan} architecture agnostic and that it can be applied in a different planning setting. Moreover, we demonstrated that our method is robust enough to plan in setting with partial scene observations.

4. In Section 5.4 we pointed out that maze-like environments do pose a challenge to our method and we proposed several options for tackling the associated issues.

In parallel, we also successfully demonstrated the implicit goal of planning in diverse settings. In our following Chapter 6, we consider how to take our proposed approach further.

Chapter 6

Conclusion

In this work, we present two key innovations to deep learning-based approaches for shortest path planning.

1. We introduce a scale-invariant cost formulation with theoretical guarantees about minima in collision-free and shortest paths. Our cost formulation allows us to train networks on diverse planning problems unsupervised, without the requirement of ground truth optimal paths.

2. We devise a regression-based approach for shortest path planning which greatly improves planning efficiency by requiring paths to be sampled using only a single forward pass of a network.

To validate our method, we first showcase the importance of our novel cost formulation by comparing our objective to standard objectives used in the gradient-based planning literature. On top of this, we train a total of 3 planning networks for path planning tasks using our proposed method. Our full-state planning networks beat state-of-the-art performance in complex 3D environments, our partial state planning networks showcase the ability to plan from depth images, and finally, our two-dimensional grid-world planning networks hint at some of the shortcomings of our approach.

All in all, the results demonstrate that our unsupervised method is capable to generalise and achieve fast path planning in previously unseen environments. We were able to deploy our method in diverse planning contexts with good performance, and our new approach opens several exciting avenues for future work.

6.1 Future work

We believe that there exist several options for both extending our current formulation and for using it in previously unexplored ways. The following list represents a selection of the most important ideas that could lead to a publication or a future dissertation.

Weak supervision

Much of the problems that we encountered when working with hard maze-like environments in Section 5.4 boiled down to problems with fully optimising our networks due to the increased cost field complexity. We believe that this could be mitigated by incorporating some form of weak supervision in our pipeline, such as occasional ground truth oracle planner paths. By letting our method optimise on a single scene for an extended period, the oracle paths themselves could be generated by our method.

Actor-Critic planning

Before deriving our cost formulation, we spent a considerable amount of time on a planning setting where we had a single step planning actor (our regression network), whose paths were evaluated by a critic network (a collision detection network). We did not solely rely on a direct Reinforcement Learning formulation, but we tried to incorporate the critic in our training pipeline with a

downscaled unsupervised cost. While this approach showed some early promise, we had a tough time training the two networks in parallel in more challenging planning settings. In the context of regression planning, we believe that this avenue is worth exploring further as it showed very promising early results in our experiments.

Planning faster

Our work mostly focuses on showing that it is possible to train path regression networks in an unsupervised fashion. As a result, we did not put a lot of effort into optimising our networks for inference speed. A possible investigation would be exploring the minimal network capacity necessary to perform high-quality path planning using our method.

Hierarchical RL

A key drawback of RL-based learning approaches is costly training time. For example, UNREAL takes 25 million iterations and roughly four days to train [81, 82]. This is partly due to the incremental short-horizon nature of action selection, which can cause agents to explore low-reward regions of the state space for extended periods during training. Furthermore, optimal agent motions are typically hierarchical, requiring both long-horizon and short-horizon components, which can be difficult to learn for a network outputting actions at a constant frequency, even if high reward regions are discovered. Many works have demonstrated the improved sample efficiency of hierarchical decomposition for action selection, either using pre-defined high-level goals and representations [83], or allowing these to arise implicitly through the learning process [84]. Learning to predict complete collision-free paths directly in the agent action space is a promising candidate for a useful sub-controller in a more hierarchical RL setup.

Manipulator control

In this dissertation, we performed experiments for planning in two and three-dimensional settings. A natural extension of these experiments is adjusting our approach to solve higher dimensional planning problems for robotic control. This work will involve sampling splines in a robot's joint space and likely adding cost terms for respecting kinodynamic constraints such as acceleration, torque, and velocity bounds.

Bibliography

[1] Facundo Bre, Juan M Gimenez, and Víctor D Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441, 2018.

[2] Adit Deshpande. A beginner's guide to understanding convolutional neural networks. 2016.

[3] Subhrajit Bhattacharya. Illustration of dijkstra. License: Creative Commons BY 3.0.

[4] Subhrajit Bhattacharya. Illustration of a*. License: Creative Commons BY 3.0.

[5] Javed Hossain. Illustration of rrt. License: Creative Commons BY-SA 3.0.

[6] Michael A. Goodrich. Potential fields tutorial. 2002.

[7] Mogens Graf Plessen, Pedro F Lima, Jonas Mårtensson, Alberto Bemporad, and Bo Wahlberg. Trajectory planning under vehicle dimension constraints using sequential linear programming. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[8] Masaya Inoue, Takahiro Yamashita, and Takeshi Nishida. Robot path planning by lstm network under changing environment. In *Advances in Computer Communication and Computational Sciences*, pages 317–329. Springer, 2019.

[9] Holger Banzhaf, Paul Sanzenbacher, Ulrich Baumann, and J Marius Zöllner. Learning to predict ego-vehicle poses for sampling-based nonholonomic motion planning. *IEEE Robotics and Automation Letters*, 4(2):1053–1060, 2019.

[10] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[11] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[12] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018.

[13] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.

[14] Namik Kemal Yilmaz, Constantinos Evangelinos, Pierre FJ Lermusiaux, and Nicholas M Patrikalakis. Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *IEEE Journal of Oceanic Engineering*, 33(4):522–537, 2008.

[15] Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *autonomous robots*, 8(2):161–171, 2000.

[16] KS Yakovlev, DA Makarov, and ES Baskin. Automatic path planning for an unmanned drone with constrained flight dynamics. *Scientific and Technical Information Processing*, 42(5):347–358, 2015.

[17] Bailin Cao, GI Doods, and George W Irwin. Time-optimal and smooth constrained path planning for robot manipulators. In *Proceedings of the 1994 IEEE international conference on robotics and automation*, pages 1853–1858. IEEE, 1994.

[18] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. Path planning for the mobile robot: A review. *Symmetry*, 10(10):450, 2018.

[19] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[20] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016.

[21] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[22] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[23] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[26] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[31] Carl de Boor. Subroutine package for calculating with b-splines. *Los Alamos Scient. Lab. Report LA-4728-MS*, 1971.

[32] John Fisher, John Lowther, and Ching-Kuang Shene. If you know b-splines well, you also nnow nurbs! *SIGCSE Bull.*, 36(1):343–347, March 2004.

[33] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, 74(250):603–627, 2005.

[34] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[35] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016, 2016.

[36] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[37] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[38] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[39] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[40] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2), 2010.

[41] Fahad Islam, Jauwairia Nasir, Usman Malik, Yasar Ayaz, and Osman Hasan. Rrt∗-smart: Rapid convergence implementation of rrt∗ towards optimal solution. In *2012 IEEE International Conference on Mechatronics and Automation*, pages 1651–1656. IEEE, 2012.

[42] Olzhas Adiyatov and Huseyin Atakan Varol. Rapidly-exploring random tree based memory efficient motion planning. In *2013 IEEE International Conference on Mechatronics and Automation*, pages 354–359. IEEE, 2013.

[43] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[44] Nathan A Wedge and Michael S Branicky. On heavy-tailed runtimes and restarts in rapidly-exploring random trees. In *Twenty-third AAAI conference on artificial intelligence*, pages 127–133, 2008.

[45] Peng Cheng and Steven M LaValle. Reducing metric sensitivity in randomized trajectory design. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 1, pages 43–48. IEEE, 2001.

[46] Jürgen Guldner and Vadim I Utkin. Sliding mode control for gradient tracking and robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 11(2):247–254, 1995.

[47] Min Gyu Park and Min Cheol Lee. A new technique to escape local minimum in artificial potential field based path planning. *KSME international journal*, 17(12):1876–1885, 2003.

[48] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 1936–1941. IEEE, 2002.

[49] Sterling J Anderson, Steven C Peters, Tom E Pilutti, and Karl Iagnemma. An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios. *International Journal of Vehicle Autonomous Systems*, 8(2-4):190–216, 2010.

[50] Cedric S Ma and Robert H Miller. Milp optimal path planning for real-time applications. In *2006 American Control Conference*, pages 6–pp. IEEE, 2006.

[51] Gorkem Erinc and Stefano Carpin. A genetic algorithm for nonholonomic motion planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1843–1849. IEEE, 2007.

[52] Mayur J Bency, Ahmed H Qureshi, and Michael C Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. *arXiv preprint arXiv:1904.11102*, 2019.

[53] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv preprint arXiv:1706.09829*, 2017.

[54] Jing Xin, Huan Zhao, Ding Liu, and Minqi Li. Application of deep reinforcement learning in mobile robot path planning. In *2017 Chinese Automation Congress (CAC)*, pages 7112–7116. IEEE, 2017.

[55] Leonid Butyrev, Thorsten Edelhäußer, and Christopher Mutschler. Deep reinforcement learning for motion planning of mobile robots. *arXiv preprint arXiv:1912.09260*, 2019.

[56] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017.

[57] Ping Wu, Yang Cao, Yuqing He, and Decai Li. Vision-based robot path planning with deep learning. In *International Conference on Computer Vision Systems*, pages 101–111. Springer, 2017.

[58] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. *arXiv preprint arXiv:1710.05627*, 2017.

[59] Divya Davis and P Supriya. Implementation of fuzzy-based robotic path planning. In *Proceedings of the Second International Conference on Computer and Communication Technologies*, pages 375–383. Springer, 2016.

[60] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[61] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[62] Fiorato Nicola, Yasutaka Fujimoto, and Roberto Oboe. A lstm neural network applied to mobile robots path planning. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 349–354. IEEE, 2018.

[63] Joshua Allen Shaffer. *Expanding Constrained Kinodynamic Path Planning Solutions through Recurrent Neural Networks*. PhD thesis, 2019.

[64] Ni Bin, Chen Xiong, Zhang Liming, and Xiao Wendong. Recurrent neural network for robot path planning. In *International Conference on Parallel and Distributed Computing: Applications and Technologies*, pages 188–191. Springer, 2004.

[65] Luca Caltagirone, Mauro Bellone, Lennart Svensson, and Mattias Wahde. Lidar-based driving path generation using fully convolutional neural networks. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[66] Noé Pérez-Higueras, Fernando Caballero, and Luis Merino. Learning human-aware path planning with fully convolutional networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE, 2018.

[67] Tom Jurgenson and Aviv Tamar. Harnessing reinforcement learning for neural motion planning. *arXiv preprint arXiv:1906.00214*, 2019.

[68] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[69] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807. IEEE, 1993.

[70] Oliver Brock and Oussama Khatib. Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031–1052, 2002.

[71] Maxim Garber and Ming C Lin. Constraint-based motion planning using voronoi diagrams. In *Algorithmic Foundations of Robotics V*, pages 541–558. Springer, 2004.

[72] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[73] Kevin Judd and Timothy McLain. Spline based path planning for unmanned air vehicles. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4238, 2001.

[74] Evgeni Magid, Daniel Keren, Ehud Rivlin, and Irad Yavneh. Spline-based robot navigation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2296–2301. IEEE, 2006.

[75] John Connors and Gabriel Elkaim. Analysis of a spline based, obstacle avoiding path planning algorithm. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 2565–2569. IEEE, 2007.

[76] Luigi Biagiotti and Claudio Melchiorri. B-spline based filters for multi-point trajectories planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 3065–3070. IEEE, 2010.

[77] Kwangjin Yang, Sangwoo Moon, Seunghoon Yoo, Jaehyeon Kang, Nakju Lett Doh, Hong Bong Kim, and Sanghyun Joo. Spline-based rrt path planner for non-holonomic robots. *Journal of Intelligent & Robotic Systems*, 73(1-4):763–782, 2014.

[78] Tim Mercy, Ruben Van Parys, and Goele Pipeleers. Spline-based motion planning for autonomous guided vehicles in a dynamic environment. *IEEE Transactions on Control Systems Technology*, 26(6):2182–2189, 2017.

[79] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[80] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[81] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[82] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[83] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

[84] Alexander C Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. *arXiv preprint arXiv:1906.05862*, 2019.