UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient and Generalizable Motion Planning using Transformers

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Intelligent Systems, Robotics and Control)

by

Jacob J. Johnson

Committee in charge:

Professor Michael C. Yip, Chair
Professor Nikolay Atanasov
Professor Sicun Gao
Professor Nuno Vasconcelos
Professor Xiaolong Wang

2023

The Dissertation of Jacob J. Johnson is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

I want to dedicate this thesis to my family, without whom this journey wouldn't have been possible. To my acha and amma - Mr. Johnson Varughese and Dr. Shaly John for their loving support and prayers throughout this journey, to my brother and his wife, Dr. Benjamin Johnson and Mrs. Rebecca Maruthelil, for their guidance and encouragement, and to my wife, Mrs. Merin Sabu, for her love and patience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to express my heartfelt gratitude to my advisor, Dr. Michael Yip, for his support and mentorship, which have guided me throughout my doctoral program. Dr. Yip's belief in my potential and the opportunity he provided for me to pursue my Ph.D. have been pivotal milestones in my academic journey. His guidance has shaped my research and instilled in me a profound appreciation for the broader impact of the problems I am addressing, emphasizing the importance of focusing on real-world applications and societal relevance.

I also would like to sincerely thank Dr. Ahmed Qureshi for introducing me to the fascinating realm of motion planning. His expertise and guidance have been invaluable throughout my journey in this field, as he shared his profound technical knowledge and provided constructive feedback at various critical junctures of this work. Beyond his role as a mentor, Dr. Qureshi has proven to be an exceptional friend, always ready to offer support and encouragement, making my doctoral endeavor all the more rewarding.

I thank my committee members Dr. Nikolay Atanasov, Dr. Xiaolong Wang, and Dr. Nuno Vasconcelos, for their support and feedback during the different stages of my thesis— a special thanks to Dr. Sicun Gao for his feedback on the VQ-MPT work.

This journey wouldn't have been possible without the support and mentorship of different people at UCSD. I want to thank Dr. Carlos Nieto-Granda and Dr. Ruffin White for their guidance and expertise with various robotics software tools and Christopher D'Ambrosia for helping me to become a better writer. I want to thank the present and past members of ARCLab, especially - Dr. Fei Lu, Dr. Shan Lin, Nikhil Shinde, Yuheng Zhi, Jingpei Lu, Elizabeth Peiros, Xiao Liang, Shreya Saha, Dr. Dimitri Schreiber, Dr. Florian Richter, Dr. Nikhil Das, Uday Kalra, Ankit Bhatia, and Linjun Li for supporting me throughout my program. Lastly, I want to thank Dr. Henrik Christensen for his unwavering support and for organizing insightful seminars at UCSD, which have contributed significantly to my academic and professional growth.

Chapter 2, in part, is a reprint of material from J. J. Johnson, L. Li, F. Liu, A. H. Qureshi and M. C. Yip, "Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6937-6943, doi: 10.1109/IROS45743.2020.9341283. The dissertation author is the primary author of this paper.

Chapter 3, in part, is a reprint of material from J. J. Johnson, U. S. Kalra, A. Bhatia, L. Li, A. H. Qureshi, and M. C. Yip "Motion Planning Transformers: A Motion Planning Framework for Mobile Robots," on arXiv preprint arXiv:2106.02791. The dissertation author is the primary author of this paper.

Chapter 4, in part, is a reprint of material from J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Learning Sampling Dictionaries for Efficient and Generalizable Robot Motion Planning with Transformers," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2023.3322087. The dissertation author is the primary author of this paper.

Chapter 5, in part, is a reprint of material from J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Zero-Shot Constrained Motion Planning Transformers Using Learned Sampling Dictionaries," on arXiv preprint arXiv:2309.15272. The dissertation author is the primary author of this paper.

Chapter 6, in part, is a reprint of material from J. J. Johnson and M. C. Yip, "Chance-Constrained Motion Planning using Modeled Distance- to-Collision Functions," 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE) pp. 1582-1589, doi: 10.1109/CASE49439.2021.9551655. The dissertation author is the primary author of this paper.

VITA

2017	Bachelor of Technology in Electronics and Electrical Engineering with a minor in Computer Science, Indian Institute of Technology - Guwahati.
2019	Masters of Science in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California San Diego
2023	Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California San Diego

PUBLICATIONS

J.J. Johnson, A.H. Qureshi, and M.C. Yip, "Zero-Shot Constrained Motion Planning Transformers Using Learned Sampling Dictionaries," in arXiv preprint arXiv:2309.15272, 2023.

J.J. Johnson, A.H. Qureshi, and M.C. Yip, "Learning Sampling Dictionaries for Efficient and Generalizable Robot Motion Planning with Transformers," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2023.3322087.

N.U. Shinde, **J.J. Johnson**, S. Herbert, and M.C. Yip, "Object-centric Representations for Interactive Online Learning with Non-Parametric Methods." In 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), pp. 1-6. IEEE, 2023.

J.J. Johnson, U.S. Kalra, A.Bhatia, L. Li, A.H. Qureshi, and M.C. Yip, "Motion Planning Transformers: A Motion Planning Framework for Mobile Robots," in arXiv preprint arXiv:2106.02791, 2022.

J.J. Johnson and M.C. Yip, "Chance-Constrained Motion Planning using Modeled Distance-to-Collision Functions," 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, 2021, pp. 1582-1589, doi: 10.1109/CASE49439.2021.9551655.

J.J. Johnson, L. Li, F. Liu, A.H. Qureshi, and M.C. Yip, "Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6937-6943. IEEE, 2020.

A.H. Qureshi, **J.J. Johnson,**, Y. Qin, B. Boots, and M.C. Yip, "Composing ensembles of policies with deep reinforcement learning, in International Conference on Learning Representations (ICLR), 2020.

ABSTRACT OF THE DISSERTATION

Efficient and Generalizable Motion Planning using Transformers

by

Jacob J. Johnson

Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control)

University of California San Diego, 2023

Professor Michael C. Yip, Chair

Robots have played a crucial role in industries for decades, streamlining manufacturing processes and performing tasks with precision. However, with the availability of more affordable components and the rapid advancement of AI techniques, there is a growing interest in integrating robots into everyday settings. The challenge lies in the decades-old algorithms used in industrial robots often assume highly structured and controlled environments. These algorithms struggle to adapt to unstructured environments commonly found in everyday life, where obstacles and uncertainties abound. As a result, there is a need for newer, more adaptable algorithms to make robots a practical and safe part of our daily lives.

Motion planning, an age-old challenge in robotics, revolves around the task of determining a safe and efficient path for a robot to navigate an environment while evading obstacles. This problem is central in various robotics applications, from developing autonomous vehicles and healthcare robots to household robotic assistants. While existing methods have proven effective in generating trajectories and paths for pre-defined, structured environments, they face significant challenges when dealing with robots boasting higher degrees of freedom and adapting to changing environments. In these cases, the generated trajectories often necessitate further refinement before a robot can successfully execute them. In response to these limitations, there has been a recent surge of interest in using learning-based methods, which can address the shortcomings of traditional planners and enhance the generalizability and efficiency of robot motion planning algorithms.

This thesis introduces efficient planning algorithms that exhibit remarkable adaptability to various environments. Leveraging techniques from large language models, specifically the versatile Transformer architecture, we demonstrate how our planning algorithms can rapidly generate efficient trajectories while generalizing across diverse environmental contexts. Notably, we showcase the capacity of our learned models to tackle complex motion planning challenges, such as constraint planning, without the need for additional training data. By introducing a novel constraint function to encode the variabilities inherent in planning environments, we also lay the foundation for capturing and addressing different sources of uncertainties in the planning process. Looking ahead, we anticipate that our approaches will not only be readily accessible but also broadly beneficial, facilitating the seamless transfer of learned motion planners into a myriad of robotic-environment interaction scenarios and ushering in new possibilities for innovation and practical application in robotics.

Chapter 1

Introduction

Robots have long been harnessed to perform burdensome tasks, ranging from industrial welding [100] and warehouse management to ambitious endeavors like space construction. With the continuous advancement and increased affordability of robotics hardware, these systems are now more commonplace. They are being deployed to tackle complex challenges, including rescue operations [99], household robotic manipulation [123, 103, 52], and surgical automation. However, the algorithms developed for robotics in the past, primarily tailored for structured environments, often prove ineffective when confronted with the complexities of unstructured real-world settings. In light of these limitations, this thesis centers on introducing innovative motion planning algorithms that prioritize efficiency and generalizability, paving the way for more effective and adaptable robotic solutions in the modern world.

Motion planning is the task of finding a trajectory for a robot through an environment while avoiding collisions. The algorithm can also find a trajectory with additional constraints, such as the shortest or safest path. In the past few decades, a profusion of work has focused on the motion planning problem for an assortment of tasks such as autonomous vehicles [117, 61, 82], constrained robotic manipulation [123, 103], and surgical robot automation [84]. Motion planners can be broadly classified into three types - Sampling-based Motion Planners (SMP) - involves constructing trees or graphs in

the planning space by randomly sampling points [80, 68], Search-based Motion Planners - discretizes the space into grids and uses a heuristic function to find a path [49] or Optimization-based - formulates the planning problem into an optimization problem to find a trajectory [112, 34]. The most effective type that can adapt to a wide range of planning problems is SMP. Although these methods provide theoretical guarantees and effectively find a path, they do not scale well to problems with larger planning spaces and diverse environments. This is a frequent problem for robots in large warehouses and household robots.

Learning-based methods have emerged as a promising solution to the challenges of SMPs [42, 76, 105, 142, 128]. These methods leverage prior planned paths and sensor data to enhance planning efficiency. Employing a combination of network models such as convolutional neural networks, Graph Networks, and multi-layer perceptrons, these methods can accurately predict optimal sample points for constructing planning trees within the SMP framework, resulting in significantly reduced planning times and the ability to generalize to robots with higher degrees of freedom [106, 62, 76]. Nevertheless, the drawback lies in their reliance on environment-specific datasets for training. Even when attempting to extend their capabilities to larger and more complex environments, they demand copious amounts of trajectory data. This underscores the ongoing quest for automation, emphasizing the necessity for planning algorithms that can scale effectively while efficiently managing planning problem complexity and computation time.

Reducing the search space has proven to be a valuable approach for more efficient motion planning (See Fig. 1.1. This principle is effectively applied in works such as Informed-RRT* [44] and Batch Informed-RRT* [45], where hyper ellipsoids are employed to limit the exploration area. However, as we venture into higher-dimensional spaces, these parametric functions become increasingly inefficient. We propose a novel approach that leverages techniques from language models, specifically Transformer models, to identify and define sampling regions. There is a strong correlation between motion

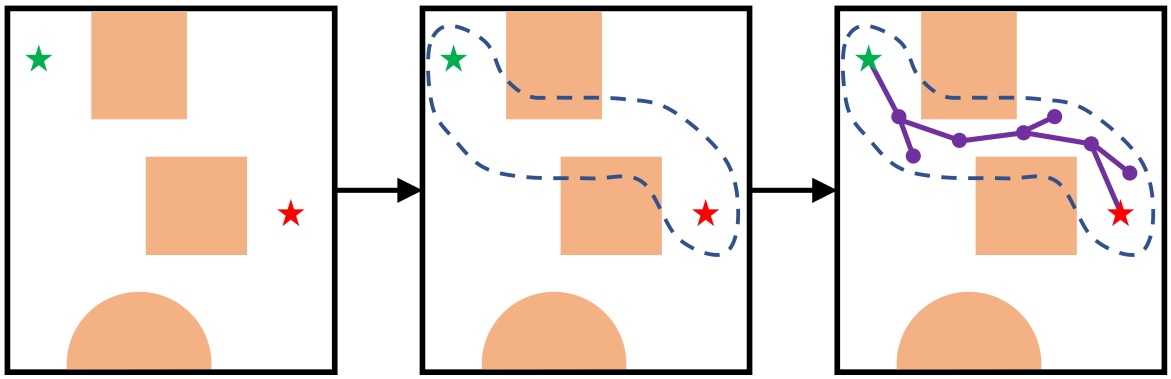

Figure 1.1. Given a planning problem (left), we propose learning-based approaches to reduce the search space for sampling-based planners. We identify regions where a valid path might exist (center). Given the proposed region, a sampling-based planner finds a collision-free path (right).

planning and natural language processing tasks; both require a good understanding of global dependencies. A sentence's syntactic and semantic structure is only inferred by reading the entire sentence. The ability of transformers to learn these long-horizon dependencies has made them the powerhouse of natural language processing [135].

Similarly, the orientation of far-away obstacles influences the construction of a local plan. Building on the success of Transformer models, we use these models for making these long-horizon correlations for path-planning problems. This innovative utilization of language models can revolutionize motion planning, making it more adaptable and effective in complex, high-dimensional spaces. In this dissertation, we build towards such planners:

1. In Chapter 2, we introduce simple ideas for improving traditional learning-based planning methods such as Motion Planning Networks (MPNet) [105]. We propose modifications to the training and planning networks that make it possible for real-time planning while improving the data efficiency of training and trained models' generalizability.

2. In Chapter 3, we introduce Motion Planning Transformers (MPT), a

transformer-based approach, which applies Vision Transformers [32] to $SE(2)$ planning problems. The model learns to restrict search spaces for simple 2D and non-holonomic robotic systems by learning to discern regions with a valid path from prior data.

3. In Chapter 4, we introduce Vector Quantized-Motion Planning Transformers (VQ-MPT) that overcome the key generalization and scaling drawbacks of MPT. VQ-MPT consists of two stages. Stage 1 is a Vector Quantized-Variational AutoEncoder model that learns to represent the planning space using a finite number of sampling distributions, and stage 2 is an Auto-Regressive model that constructs a sampling region for SMPs by selecting from the learned sampling distribution sets. By splitting large planning spaces into discrete sets and selectively choosing the sampling regions, our planner pairs well with out-of-the-box SMPs, generating near-optimal paths faster than without VQ-MPT's aid.

4. In Chapter 5, we show how our VQ-MPT can address complex challenges like constraint planning using previously trained models. We also update the prediction sampling regions closer to the constraint manifold using optimization techniques.

5. Finally, in Chapter 6, we lay the foundation to capture the variations in the distance-to-collision measurements caused by the uncertainty in state estimation techniques using a Gaussian Process (GP) model. The GP model generates paths that reduce collisions and meet optimality criteria under motion and state uncertainties.

1.1 Problem Definition

Consider the planning space defined by $\mathcal{X} \in \mathbb{R}^n$. We define a subspace $\mathcal{X}_{free} \subset \mathcal{X}$, such that all states in $\mathcal{X}_{free}$ do not collide with any obstacle in the environment and are

considered valid configuration. The objective of the motion planner is to generate a sequence of states: $\mathcal{Q} = \{q_1, q_2, \ldots, q_{n_s}\}$ for a given start state (q_1) and a goal region ($\mathcal{X}_{goal}$) such that $q_i \in \mathcal{X}_{free}, \forall i \in \{1, 2, \ldots, n_s\}$, the edge connecting q_i and q_{i+1} is also in $\mathcal{X}_{free}$, i.e., $(1-\alpha)q_i + \alpha q_{i+1} \in \mathcal{X}_{free}, \forall \alpha \in [0, 1]$, and $q_{n_s} \in \mathcal{X}_{goal}$. The sequence of states is often referred to as a trajectory or path. Additional objectives such as minimizing path length given by the expression:

$$l = \min \sum_{i=1}^{n-1} \|q_{i+1} - q_i\| \tag{1.1}$$

can also be added to the planning problem.

1.2 Acknowledgement

Chapter 1, in part, is a reprint of the following material:

- J. J. Johnson, U. S. Kalra, A. Bhatia, L. Li, A. H. Qureshi, and M. C. Yip "Motion Planning Transformers: A Motion Planning Framework for Mobile Robots," on arXiv preprint arXiv:2106.02791.
- J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Learning Sampling Dictionaries for Efficient and Generalizable Robot Motion Planning with Transformers," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2023.3322087.
- J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Zero-Shot Constrained Motion Planning Transformers Using Learned Sampling Dictionaries," on arXiv preprint arXiv:2309.15272.

The dissertation author is the primary author of these papers.

Chapter 2

Learning-based Planners

In the robotics community, learning-based methods to solve planning problems have been proposed to offer speed improvements and typically revolve around using convolution neural networks (CNN) and multi-layer perceptrons (MLP) to predict where samples should be generated to construct trees [106, 62]. Other works employ different forms of latent representation, such as gird cells [3] and Sparse Graphical Memory (SGM) [38] to generate a plan.

Recently, reinforcement learning-based methods that use neural networks have gained traction in solving motion planning problems [144, 41, 23] for non-holonomic systems. These methods often require careful fine-tuning of reward functions and significant computing resources to search for proper hyperparameters. Another class of motion planning algorithms, called neural motion planners, have emerged that learns to imitate an oracle planner and exhibits the virtues of an ideal planner during online execution [105, 105, 107]. Motion Planning Networks (MPNet) [105] is one of the first and most prominent neural motion planning methods, showing orders of magnitude performance computational speed compared to previous offline methods while producing near-optimal solutions. However, MPNet, along with its extensions [65], only considers collision-avoidance constraints and finds a viable path in the robot's configuration spaces, i.e., without considering the system's kinematic limitations. Furthermore, these methods

also require significant training data to achieve generalizability and, in their current formulation, have yet to be scaled to real-time planning scenarios involving navigating large environments. Thus, none of these methods single-handedly has the features of an ideal planner, i.e., find near-optimal/optimal paths with high, almost real-time, computational speed and exhibit completeness guarantees.

This chapter presents Dynamic Motion Planning Networks (Dynamic MPNet), which extends MPNet to plan under a broad class of non-holonomic constraints in real time. Dynamic MPNet is a deep neural network-based iterative planning algorithm. It takes the sub-goals between given start and goal states from a global C-space planner. It finds a kinematically feasible path between them with high computational speed and completeness guarantees. Real-time planning is made possible by planning during the execution of the previous plan, similar to anytime planning systems. We evaluate our framework on Dubin's car dynamical model in challenging navigation tasks where state-of-the-art classical methods fail, including simulation and real-robot experiments.

2.1 Dynamic Motion Planning Networks

Dynamic MPNet uses supervised learning to train neural networks that generate near-optimal paths through an environment given a start and goal position. The networks are trained with expert trajectories in randomized, diverse environments so that unseen environments can be planned in with near-expert level cost and with substantially less sampling than classical sample-based planners.

Dynamic MPNet consists of 3 core modules, a *transformer* that centers the local costmap with respect to the robot, an *encoder* network that takes the ego-centered obstacle map and converts it into a latent vector, and a *planner* network that takes the latent encoding, the current or predicted pose, and goal pose and returns the next feasible step. In the following section, we will go over the environment encoding, network training

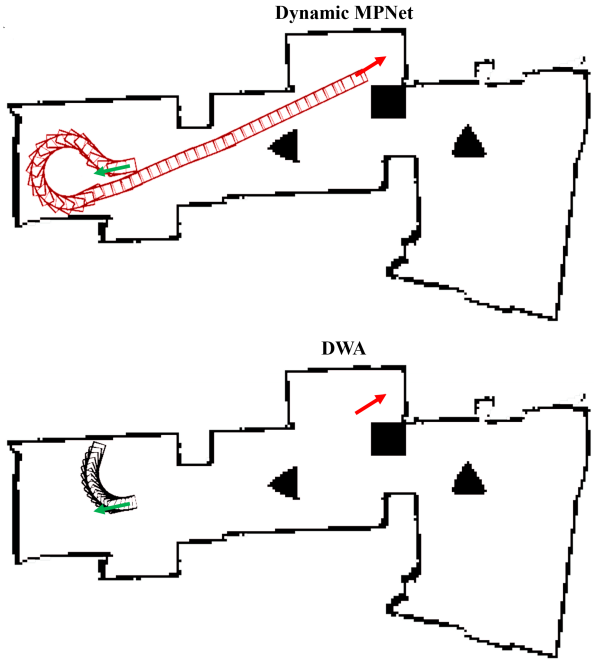

Figure 2.1. The trajectory the robot followed for a given start (green arrow) and goal (red arrow) position using, Dynamic MPNet (red), and Dynamic Windows Approach (black) as local planners respectively. Since the DWA planner has no kinematic constraints, it is difficult for the planner to generate paths with U-turns, while since the path generated from Dynamic MPNet encodes kinematic constraints, it is able to generate a successful path towards the goal.

and planning pipeline in more detail.

2.1.1 Environment Encoding

Most non-holonomic systems reside in environments that can vary significantly in size. Encoding an environment where the map could be arbitrarily large in size is infeasible. This is where a hierarchical approach to the navigation is useful. Using the global plan as a guide, kinematically feasible paths are generated for a local region using Dynamic MPNet. Fig. 2.2 is an example for the local costmap passed into the network planner. This costmap is always egocentric to the robot (see Fig. 2.2). Doing this reduces the dimensionality of the input space, as the output of the network is only a function of the current robot orientation, relative goal position, and costmap. It could also be viewed

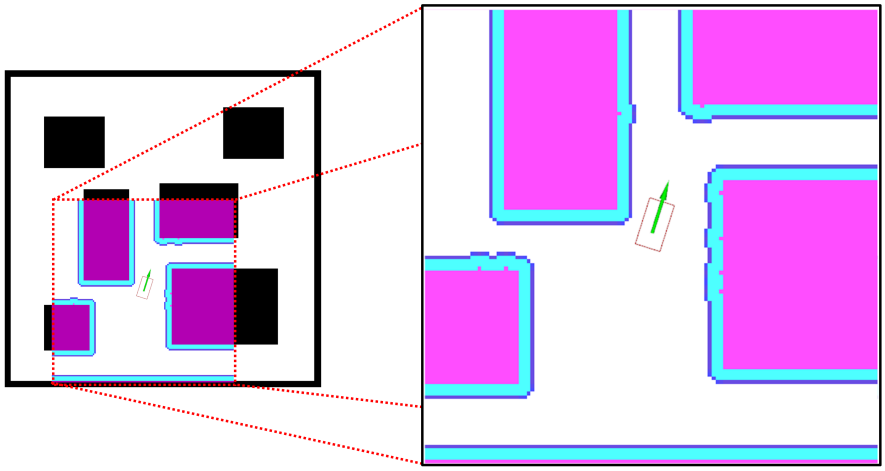

Figure 2.2. The area of the costmap passed to MPNet for planning. The local costmap used for planning is always egocentric to the robot. The blue region indicates the obstacles inflated by the robot footprint.

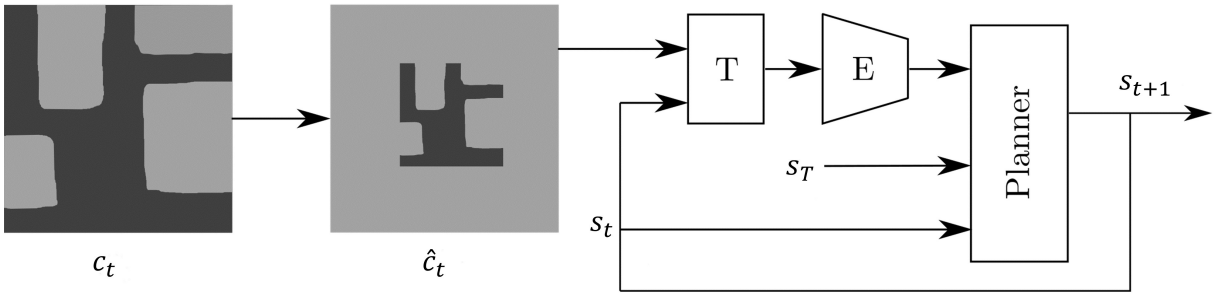

Figure 2.3. The graph describes the flow of inputs and outputs for the planner. x_t, x_{t+1} and x_g represents the current, next and target positions respectively. x_g is the sub-goal position from the global plan. C_t and, $\hat{C}_t$ is the costmap before and after padding respectively. The T block centers the padded costmap, $\hat{C}_t$, with respect to the robot position x_t. E block consists of convolution networks that encode the costmap into latent space vectors. The latent space representation of the costmap, the current robot and goal position are passed to the Planner node to generate the new target point.

as a normalization step, to disentangle the dependence of training trajectory and world map with the sampled point. It thus helps the model to generalize to the environment better with fewer training samples.

2.1.2 Training

The planner is trained with dynamically feasible trajectories such that, during prediction, it will tend to predict dynamically feasible intermediate poses. Given an expert trajectory $\{s_0, s_1, \ldots, s_T\}$ that satisfies the dynamical constraints of the robot for the planning problem, the network uses the current position (s_t), goal position (s_T) and obstacle representation ($\hat{c}_t$) to predict the next state ($\hat{s}_{t+1}$). These four elements form a training tuple $(s_t, s_T, \hat{c}_t, s_{t+1})$. The encoding and planning networks are trained by reducing the mean-squared error between the predicted state $\hat{s}_{t+1}$ and the actual state from the expert planner s_{t+1} in an end-to-end fashion using gradient descent. The loss function for N such trajectories is given by:

$$L(\theta) = \frac{1}{N\ T} \sum_{j=1}^{N} \sum_{t=1}^{T-1} ||s_{j,t} - \hat{s}_{j,t}||^2 \tag{2.1}$$

where θ represents the combined parameters of both the encoder and planner network.

2.1.3 Planning

The network starts planning using the current position, sub-goal position from the global plan, and the local costmap to generate a sequence of kinematically feasible states. For each step, if the predicted state is kinematically feasible and is collision-free, it is used as the current position for the next prediction. Fig. 2.3 shows the complete pipeline. Then, for each sampled step during the planning, an egocentric costmap encoding at this immediate (non-initial) location is required to generate the next step. This costmap is achieved by translating the initial costmap. It is necessary to pad the obstacle map such that no loss in obstacle information occurs after transforming the map to an egocentric

pose of an intermediate step of a motion plan. Given a grid of size $l \times l$, we pad it to make a new grid of size $2l \times 2l$. This way, for the planner, the world's perception remains the same, since all the padded spaces are still obstacles.

Algorithm 1 outlines the Dynamic MPNet planner, and Algorithm 2 outlines the modified path generation heuristic. The functionality of the different function calls are defined as follows:

Padding

The `Pad` function takes a costmap $c_{obs} \in \mathbb{R}^{l \times l}$, and returns a padded costmap $\hat{c}_{temp} \in \mathbb{R}^{2l \times 2l}$. Padded values are assumed to be obstacle regions to prevent the planner to find paths that would navigate this non-physical space. See Fig. 2.3 for an example.

Steering

The function `Steer` (x_1, x_2) checks if we can generate a sequence of kinematically feasible states without collision from x_1 to x_2 within a fixed time. It returns a feasible path if it exists otherwise, an empty list. For non-holonomic systems, a differential equation solver or parameterized curves along with a collision checker can be used to implement this function.

Network

The function `Net` represents both the encoder and planner neural network combined. It generates the next possible point on the path, given the current and goal position of the robot and the modified costmap. The flow of inputs is described in Fig. 2.3.

Add

The function `Add` (τ, x_1) appends the path τ with the node x_1.

Transform

Given a point x_i and a padded costmap $\hat{c}$, the function `Transform` ($\hat{c}$, x_i) translates the costmap in such a way that the costmap is egocentric to the position x_i.

Replanning

Only if, for a given start and goal location, the neural planner is not able to provide a feasible path under a suitable amount of time, the function will run classical sampling-based methods until a path is found, specifically to ensure probabilistic completeness.

Algorithm 1: $\tau \leftarrow$DynamicMPNet (x_{start}, x_{goal}, c_{obs})

$\hat{c} \leftarrow$ `Pad`(c_{obs});
$\tau \leftarrow$ `NeuralPlanner`($x_{start}, x_{goal}, \hat{c}$);
if `Empty`(τ) **then**
    $\tau \leftarrow$`Replanner`(x_{start}, x_{goal});
end
return τ

Algorithm 2: $\tau \leftarrow$NeuralPlanner (x_{from}, x_{to}, $\hat{c}$)

$\tau \leftarrow \{x_{from}\}$;
for $i = 0$ **to** N **do**
    $x_{temp} \leftarrow$ `Net`($x_{from}, x_{goal}, \hat{c}$);
    $\tau_{temp} \leftarrow$ `Steer`(x_{from}, x_{temp});
    if `NotEmpty`(τ_{temp}) **then**
        $\tau \leftarrow$`Add`(τ, τ_{temp});
        $\tau_{goal} \leftarrow$`Steer`(x_{temp}, x_{goal});
        if `NotEmpty`(τ_{goal}) **then**
            $\tau \leftarrow$`Add`(τ, τ_{temp});
            return τ
        end
        $x_{from} \leftarrow x_{temp}$;
        $\hat{c} \leftarrow$`Transform`($\hat{c}, x_{from}$);
    end
end
return $\varnothing$

2.1.4 Experiments

In this section, we describe the data collection, model architecture and experiment setup used in this paper. The neural networks model was defined and trained using the PyTorch[97] python library, and the trained model was loaded into C++ using the torch C++ API. To test the viability of our framework, we integrated the Dynamic MPNet planner to the navigation stack [1] provided by the ROS community and compared it with standard local planners used by the community. We used the default global planner from the ROS navigation stack and the mit-racecar [2] model was used as the robot for the simulations.[3]

2.1.5 Data Collection

For training the model, we collected expert trajectories using the RRT*[78] algorithm using the Open Source Motion Planning Library[124]. For the grid map (Fig. 2.4) environment, we trained the model on 10,000 RRT* trajectories by randomly sampling a start and goal location for a fixed time, while for the real-world environment, we generated 12,000 trajectories. To further augment our data set, we chose smaller sections of a path and added it to the training data. Thus if we have n points on a path, then we can choose any 2 points and create $\mathcal{O}(\frac{n^2}{4})$ trajectories. The data collection was accelerated by encapsulating the sampling code in a docker container and launching multiple containers concurrently with different seed values.

2.1.6 Model Architecture

A convolutional neural network model was used for environment encoding. Prior works in robotics have also used CNN's to process the cost map for planning [4] and path prediction [5]. The input to the encoder was an $l \times l$ dimension cost map. The CNN

[1] https://github.com/ros-planning/navigation
[2] https://github.com/mit-racecar/racecar_simulator
[3] https://youtu.be/1b3i1SSiUms

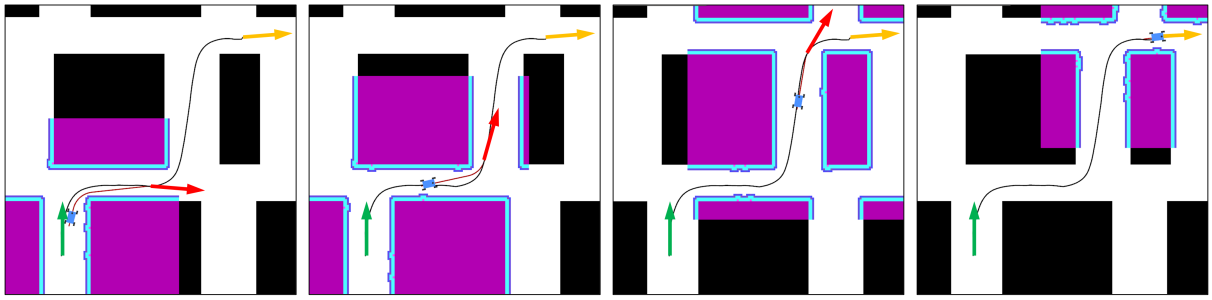

Figure 2.4. For a given start (green arrow) and goal (orange arrow) position, the plan generated by the Dynamic MPNet (red path) for a given sub-goal (red arrow). The black trajectory is the global plan. The colored region represents the local costmap used by the Dynamic MPNet.

consisted of 3 convolutional layers, with kernel size $[5, 5]$, $[3, 3]$, and $[3, 3]$, and output channels of 8,16 and 32, respectively. A maxpool and Parametric Rectified Linear Unit[130] (PReLU) layer follow the first two convolutional layers. The output of the final convolutional layer is passed through a PReLU layer to generate the output of the encoder.

The planner was a fully connected neural network with six layers. A PReLU[130] and a Dropout[122] layer follow the first four hidden layers. Dropout is used not only during training to prevent overfitting[122] but also during prediction to introduce stochasticity that tends to make motion planning networks more robust[105]. A PReLU follows the penultimate hidden layer and the output of the final hidden layer is passed through a tanh nonlinearity. Both networks were trained in an end-to-end fashion.

2.1.7 Kinematic Model of a Car-Like Robot and Dubins curve

In this paper we consider non-holonomic kinematics following the Dubins vehicle model[36] though in practice the constraint can be of another variety. The Dubins model is given by:

$$\dot{\mathbf{s}}(\mathbf{t}) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v_s \cos(\phi) \\ v_s \sin(\phi) \\ \frac{v_s}{d} \tan \phi \end{bmatrix} \tag{2.2}$$

where v_s is the speed of the car, ϕ is the steering angle and d the distance between the rear and front axle. We use Dubins curves as a steering function for Dynamic MPNet. Given the state variables for a Dubins vehicle, the shortest path is a unique path among 6-basis trajectories[36]. Each of these trajectories is represented by a sequence of left, right and straight turns. To steer between given two states, the shortest among the 6-trajectories are used.

2.1.8 Trajectory Tracking

Given a sparsely sampled path produced by Dynamic MPNet, a trajectory tracking problem is then presented during runtime to move the vehicle between the sampled points. This is necessary to ensure that due to unmodelled effects and under noise and disturbances that the vehicle follows, to the best of its capability, the solution to a dynamically feasible path provided by Dynamic MPNet.

We formulate the trajectory tracking problem as a constrained discrete optimization problem with a finite horizon. In order to follow the trajectory generated from MPNet, we sample the path nodes adaptively to proper sparsity with similar distances and apply a Nonlinear Model Predictive Control (NMPC, [48]) with the following objective function:

$$\begin{aligned} &\text{minimize} \sum_{t=0}^{N} w_s \left\|\mathbf{s(t)} - \hat{\mathbf{s}}(\mathbf{t})\right\| + w_u \left\|\phi(t)\right\|) + w_a \left\|\Delta\phi(t)\right\| \\ &\text{subject to } \Delta\mathbf{s(t)} = \begin{bmatrix} v_s \cos(\phi) \\ v_s \sin(\phi) \\ \frac{v_s}{d} \tan\phi \end{bmatrix} \Delta t \end{aligned} \tag{2.3}$$

where Δt and Δs are time step and state difference respectively, w_s is state loss weight, w_u, w_a are weights for control loss, $\hat{s}$ are sampled path nodes from Dynamic MPNet, v_s is preset as constant velocity and N is the prediction horizon. In each control cycle, the control output is computed by solving the problem with Interior Point Optimizer (Ipopt,

[10, 138]) implemented with algorithmic differentiation library CppAD [7] and the first action is taken from solution.

The prediction horizon was determined using path curvature and maximum velocity and was in sync with the control frequency to reach the targeted state. The w_a term contributes to decreasing the vibration due to maneuvering, which guarantees the smoothness of the trajectory.

2.2 Results

We evaluate the learned model in simulation on seen and unseen environments, and a real-world indoor environment using a Dubins car robot.The Dubins car is set up similar to the MIT Racecar. The local planners available for car-like robots in the ROS-navigation stack is the Dynamic-Window Approach (DWA) planner[43] and Timed-Elastic-Band (TEB)[118]. Although the TEB is the only ROS local planner for car-like robots, the optimization problem was not able to generate paths for Dubins car. As a result we compared our algorithm with DWA. We also implemented an Anytime RRT* local planner by generating the path from RRT* rather than from the Neural Planner. We compare the robot's accuracy and average speed over a batch of planning problems. Each planning problem was verified to have a solution using an offline RRT* planner. For each test case, the problem is considered solved if the robot center is able to achieve the target position within a radius of 0.2m (about half the length of the robot) and target orientation within 15°. The same threshold is used for all local planners. Each planner runs at 5Hz, giving it 200 ms to find and optimize a feasible plan. In the following sections we report our results from our experiments.

To evaluate the sampling speed of Dynamic MPNet on the trained simulated environment in terms of execution time and path length, we measured the time taken to sample n points using Dynamic MPNet is given in Table 2.1. These times indicate that

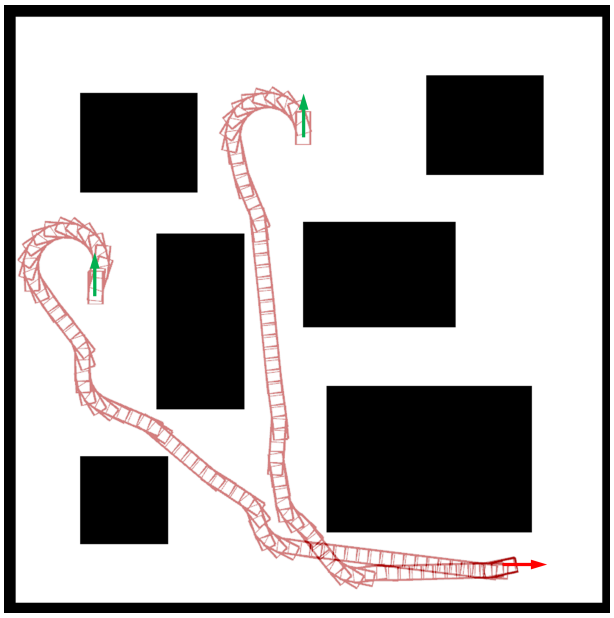

Figure 2.5. The Dynamic MPNet generating trajectories for an untrained map for two different start and end goal on a synthetic map. The planner can generate trajectories that comply with the kinematic constraints of the robot, thus achieving higher accuracy compared to DWA.

the Dynamic MPNet planner is able to generate a path within 20Hz if set to high resolution of 50 points per local path generated. Thus Dynamic MPNet is able to generate kinematically reachable points in real-time.

Table 2.1. Planning Time of Dynamic MPNet vs. Sampling Resolution

Number of samples	5	10	25	50
Compute Time (ms)	11.33	15.29	22.07	44.67

The Dynamic MPNet was trained on a synthetic grid world environment. One of the paths generated by the trained planner is shown in Fig. 2.4. To evaluate the generalizability of the planner, we created a synthetic map that is different from the training map, but shares a lot of common obstacle features such as 90° turns from the original map. Paths generated on this environment is shown in Fig. 2.5. Table 2.2 compares the percentage of planning problems solved with standard planners. Dynamic MPNet is able to plan 34% more planning problems compared to DWA on the unseen

Table 2.2. Percentage of Planning Problems Successfully Completed

Environment	DWA	Anytime RRT*	Dynamic MPNet
Unseen map	47%	74%	80%
Real world map	44%	58%	76%

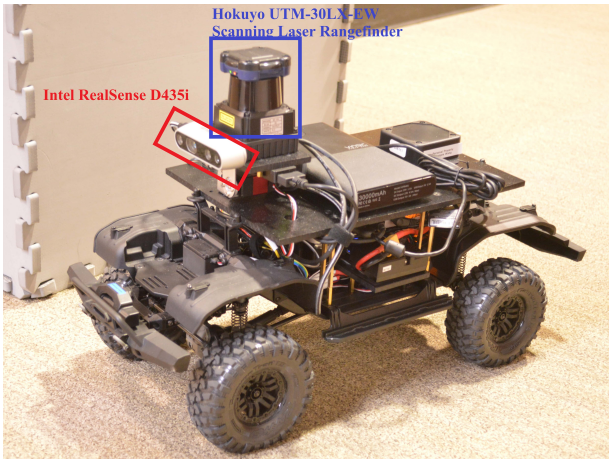

Figure 2.6. The RC robot used for this work. On board LIDAR (Hokuyo UTM-30LX-EW) and IMU (RealSense D435i) sensors were used for localization.

map. Hence we were able to achieve generalizabilty with much fewer training paths.

In addition to the simulation experiments, real-world experiments with an RC Dubins Car (see Fig. 2.6) in one of our mapped office buildings were used. Fig. 2.8 compares one of the trajectories planned by DWA and Dynamic MPNet for the same start and goal point. The red path given by Dynamic MPNet is 9.43m long while the DWA path is 10.85m long. Since Dynamic MPNet is trained on RRT* paths, the local paths generated would be near optimal. In Fig. 2.7 we compare the distance of each trajectory and time take to complete the planning problems solved by both DWA and Dynamic MPNet for the unknown and real world maps. A linear regression model was fit to both the models to estimate the average speed of the robot. Table 2.3 summarizes the results. Dynamic MPNet is able to solve planning problems faster compared to DWA.

In Fig. 2.1 we can observe one of the biggest drawbacks of the DWA planner which is that it does not take the kinematics constraint of the robot into consideration for generating a plan. Since the Dynamic MPNet generates kinodynamically feasible paths, it

Table 2.3. Average Vehicle Speed of Dynamic MPnet v.s. DWA

Environment	DWA Speed (m/s)	Dynamic MPNet Speed (m/s)
Unseen map	0.211 ± 0.016	0.340 ± 0.006
Real world map	0.263 ± 0.014	0.336 ± 0.003

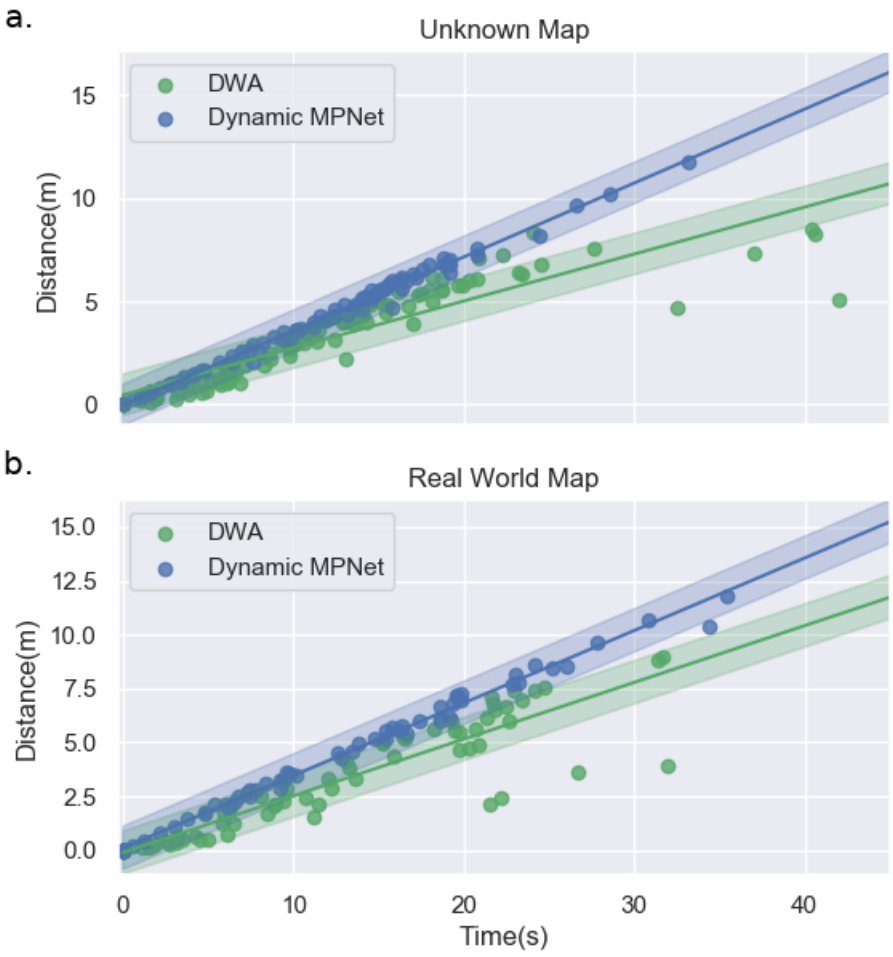

Figure 2.7. Total time and total distance taken by Dynamic MPNet and DWA planners on the same set of planning problems for an (a) unknown map (b) real world map. A linear regression model was fit to each dataset to evaluate the average speed of the car for the local planners. The shaded region represents standard error in the estimates. Dynamic MPnet not only moves faster and more consistently than DWA, it solves problems that require longer planning distances as well.

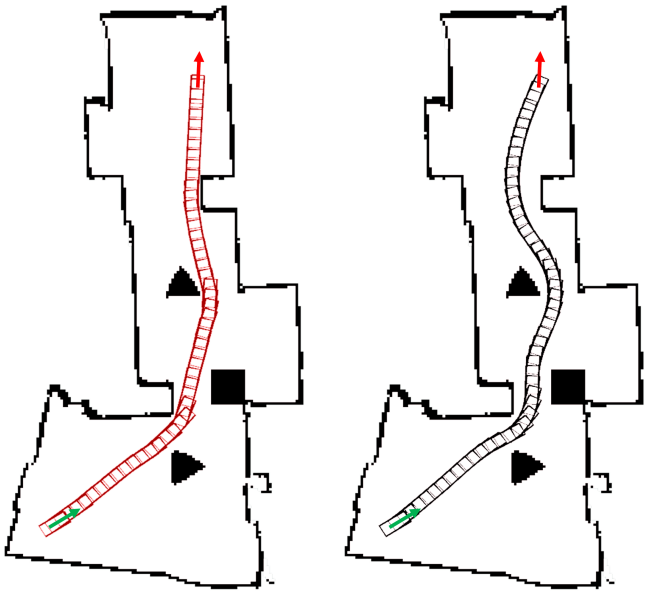

Figure 2.8. The trajectory the robot followed for a given start (green arrow) and goal (red arrow) position for DWA (black) and Dynamic MPNet (red). The path Dynamic MPNet takes is much closer to the obstacle compared to DWA and hence shorter.

is able to plan for the given goal position and orientation. As a result, MPNet is able to solve a larger percentage of planning problems compared to standard planners for both simulated and real world maps.

2.3 Discussion

In this chapter, we looked at some of the recent learning-based planners. We introduced Dynamic Motion Planning Networks, a neural motion planner to generate paths for non-holonomic robots successfully, and reduced data for training the model using ego-centric maps. We achieved this by introducing a new framework to facilitate real-time planning for non-holonomic robots. Compared to traditional sampling-based methods, Dynamic MPNet can achieve faster average speeds with higher accuracy.

While Dynamic MPNet exhibits the advantage of requiring significantly fewer data samples for training, its limitations become apparent when faced with the complexity of real-world environments. This shortcoming is partially due to its dependence on a global

planner to provide a rough trajectory for the local planner to follow. To address these issues and pave the way for a more versatile and efficient planner, we explored techniques from natural language processing. In the following chapters, we integrate such methods into the planning framework.

2.4 Acknowledgement

Chapter 2, in part, is a reprint of material from J. J. Johnson, L. Li, F. Liu, A. H. Qureshi and M. C. Yip, "Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6937-6943, doi: 10.1109/IROS45743.2020.9341283. The dissertation author is the primary author of this paper.

Chapter 3

Planning Using Transformers

Transformers have become the powerhouse of natural language processing and recently found use in computer vision tasks. Their effective use of attention can be used in other contexts as well, and in this chapter, we introduce a transformer-based approach for efficiently solving complex motion planning problems. There is a strong correlation between motion planning and language translation tasks; both require a good understanding of global dependencies. A sentence's syntactic and semantic structure is only inferred by reading the entire sentence. Similarly, the orientation of far-away obstacles influences the construction of a local plan. The ability of transformers to learn these long-horizon dependencies has made them the powerhouse of natural language processing [134]. Building on this success, we use transformer models for making these long-horizon correlations for planning problems.

In this chapter, we introduce Motion Planning Transformer (MPT), a transformer-based model to reduce search space for planning algorithms. Traditional neural network-based motion planning uses convolutional networks to encode the planning space, but these methods are limited to fixed map sizes, which is often not realistic in the real world. Our approach first identifies regions on the map using transformers to provide attention to map areas likely to include the best path and then applies traditional planners to generate the final collision-free path. We validate our method on various

randomly generated environments with different map sizes, demonstrating a reduction in planning complexity and achieving comparable accuracy to traditional planners.

3.1 Related Works

The most relevant work to our transformers-based region proposal network for motion planning is perhaps the guided sampling-based motion planning methods. They analytically or through learned heuristics determine a subset in robot space that probably contains a path solution. For instance, [109, 127] employ Artificial Potential Fields (APF) within sampling-based methods such as RRT* [66] and Bidirectional RRT* [108] to guide a subset of random samples towards promising regions that possibly contain an optimal path solution. In contrast, Informed-RRT* (IRRT*) [44] depends on an initial path from an RRT* algorithm to compute an ellipsoidal region probably containing an optimal path solution. However, in most planning problems, finding an initial path solution is itself challenging. Similarly, Batch Informed Trees (BIT*) [45] begins from an elliptical region formed by a straight line path, ignoring all obstacles and incrementally expanding it until an initial path solution is found. Once an initial path is determined, it is further optimized by adapting the precomputed ellipsoid and generating new samples within that space. Despite all advancements, these methods only consider geometric planning, i.e., path planning under collision-avoidance constraints. They are yet to be evaluated in practical problems with complex constraints, such as kinodynamic or non-holonomic constraints for autonomous car navigation tasks.

Many have also used learning-based methods to reduce search spaces. [137] used the REINFORCE algorithm to guide the underlying SMP planner on a discretized workspace, others use a learned value function to guide the graph search [24, 20], while [76] learned a generative model to sample points along bottleneck regions. Similarly, Value Iteration Networks (VIN) [128] also discretizes the space and learns a value map to guide path

planning. Universal Planning Networks (UPN) [121] extends VIN to continuous control spaces. [18] proposed one of the few works that use transformers to learn a value function. Other works employ latent representation of state spaces to generate a plan [3, 38, 55]. These methods are often difficult to train and interpret, and many of them are yet to be evaluated in real-world navigation tasks.

Although there exist random sampling-based approaches such as RRT* [66, 2] and SST [85] that explore the robot state space and satisfy advance constraints, they suffer from the curse of dimensionality and take a considerable computational time in cluttered spaces. Neural Motion Planning [105, 54] has recently emerged as a promising tool for solving a wide range of planning problems under various task constraints, ranging from non-holonomic [62, 83] to advanced manifold kinematic constraints [102], with high computational speed. These methods learn sampling distributions from expert demonstrations and, on execution, generate samples for an underlying planner, forming a subset that potentially contains a path solution. However, these approaches assume a fixed-size input environment map and often require redefining network architectures and retraining for different map sizes. However, recent developments in deep learning, primarily through Transformers [32, 90], have provided us with ways to relax such assumptions. Our proposed approach leverages these developments and introduces a region proposal framework that can work with variable map sizes and enhances underlying motion planners to solve complex problems in cluttered environments.

3.2 Motion Planning Transformers

Given a start (x_s) and goal (x_g) state, the objective is to propose a sequence of states $x_i \in \mathcal{X}$ for $i \in \{0, 1, \dots n\}$ such that $x_0 = x_s$, $x_n \in \mathcal{X}_{\text{goal}}$ and the trajectories joining these states should be both kinematically feasible and collision free. Here, $\mathcal{X}$ represents the robot state space and $\mathcal{X}_g \triangleq \{x : \|x - x_g\| \leq \epsilon, x \in \mathcal{X}\}$ for a user defined threshold (ϵ).

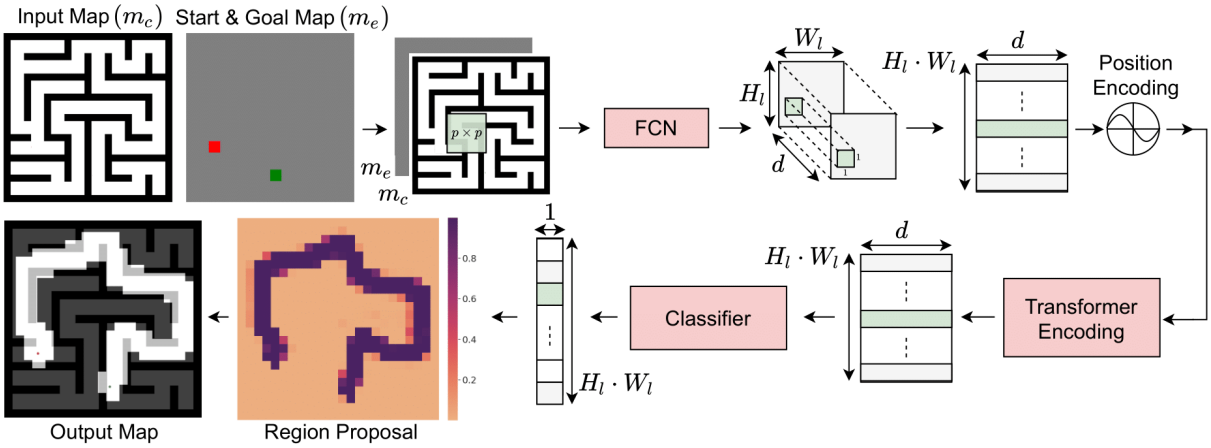

Figure 3.1. Overview of MPT Module for planning in $\mathbb{R}^2$. (Start from the top left and move clockwise) The input map and the start (green) and goal (red) encoded map are concatenated and passed as inputs to the model. The Fully Convolution Network (FCN) passes a sliding window of size $p \times p$ over the input and encodes each patch into latent vectors of dimension d. After reshaping, fixed positional encodings are added to the latent vectors to inject spatial location. The transformer module uses the modified latent representation to identify patches through which a path might exist. This information is encoded into a latent vector of size d, which a classifier uses to provide the probability that a path might pass through the patch. The patches with a probability greater than 0.5 are used to create the mask for the map. The light green shading highlights the flow of information for a single patch from the input to the output of the model.

The set of motion planning problems we will focus on in this paper is 2D navigation hence $\mathcal{X} \in \{\mathbb{R}^2, SE(2)\}$. MPT proposes promising regions in $\mathcal{X}$ where the underlying planners can search for path solutions. In the following sections, we describe our method in detail.

The MPT module is a region proposal network that uses a transformer network to identify regions of interest. An overview of the model is shown in Fig. 3.1. The input to the model is a representation of the planning scene and an encoding of the start and goal points. The planning scene is represented using an occupancy matrix, $\mathbf{m_c} \in \mathbb{R}^{H \times W}$, where an element with 1 indicates an occupied space and 0 denotes free space. In some navigation problems, these representations are also called costmap and have additional cost terms associated with safety and robot constraints. The start-goal encoding of the planning problem is formed by highlighting patches of size $p \times p$ on a tensor of size

$H \times W$ with values -1 and 1 for the start and goal points respectively. For $SE(2)$ space, we add two additional matrices to represent the robot orientation which is represented using sine and cosine values, respectively. These matrices are concatenated to form a tensor $\mathbf{m}$ and passed to the feature extractor.

3.2.1 Feature Extractor:

The feature extractor is a Fully Convolution Network (FCN) that encodes the environment and the given planning problem into a latent space. As shown in prior works [32, 114], the feature extractor reduces the dimensionality of the input space by using a series of convolution, ReLU, and MaxPool layers. The FCN passes a sliding window of size $p \times p \times i$ over $\mathbf{m}$ to generate an output of size $H_l \times W_l \times d$, where H_l and W_l is determined by the size of the costmap and the FCN, and d is the latent dimension of the transformer encoder. i is 2 for the $\mathbb{R}^2$ space and 4 for the $SE(2)$ space. The patch size p defines the size of local structures that the model encodes into the latent vector and the smallest area the model can mask. Larger patch sizes will have a lower resolution of masking; thus, the resulting planner would perform similarly to traditional planners.

The output of the FCN is reshaped row-wise to size $(H_l \cdot W_l) \times d$ and fed to the position encoder. Hence each latent vector at index (i_F, j_F) is mapped to the row $i_F * W_l + j_F$ where $i_F \in \{0, 1, \ldots, H_l - 1\}$ and $j_F \in \{0, 1, \ldots, W_l - 1\}$. Each row vector of this matrix corresponds to a $p \times p$ patch that the sliding window moves over and is referenced by an anchor point similar to Faster R-CNN [114]. We choose the 2D coordinate corresponding to the center pixel of the patch as the anchor point.

3.2.2 Position Encoding:

Transformer and convolutional models are agnostic to the spatial location of their inputs [32]. A common solution is to add learned or fixed vectors to encode the position of each input [46]. We used fixed vectors to encode the position and used the following for

testing,

$$PE(j, 2i) = \sin\left(\frac{j}{10000^{2i/d}}\right) \tag{3.1}$$

$$PE(j, 2i+1) = \cos\left(\frac{j}{10000^{(2i+1)/d}}\right) \tag{3.2}$$

$j \in \{0, 1, \ldots, H_l - 1, \ldots i_F * W_l + j_F, \ldots, H_l * W_l - 1\}$, and $i \in \{0, 1, 2 \ldots, d-1\}$ similar to [135]. The maximum value j could take was set at $H_{max}^2 - 1$. For maps larger than the training data, we observed that models trained using the position encoding in Eqn. 3.1 and 3.2 created a bias that prevented MPT from selecting regions of the state space outside the training map size. One way to resolve this overfitting is to train on maps of various sizes to ensure that the network observes different position encoding. However, the data collection is often computationally expensive, especially for larger maps. Instead, to overcome this bias, we leverage the fact that a proposed plan is invariant to linear translation of the state space and train our model by randomly shifting the position encoder. Hence for training we randomly sample $(i_R, j_R) \sim U[0, H_{max} - 1]^2$ where $U[0, H_{max} - 1]$ is a uniform distribution over discrete set of integers from 0 to $H_{max} - 1$ and use the following position encoder for each map:

$$PE(j, 2i) = \sin\left(\frac{j+k}{10000^{2i/d}}\right)$$

$$PE(j, 2i+1) = \cos\left(\frac{j+k}{10000^{(2i+1)/d}}\right)$$

where if $j = i_F * W_l + j_F$ then $k = H_{max} * (i_R + i_F) + j_R$. This effectively translates the map to different position encoding values during training and ensures that the size of the training map does not bias the model. The modified latent vector is then passed to the transformer encoder.

3.2.3 Transformer Encoder:

The transformer module is responsible for learning the connections between the different local regions on the map for a given planning problem. It infers these connections by passing the latent vectors through a series of multi-headed self-attention (MSA) and multi-layer perceptron (MLP) blocks. Between every MSA and MLP block, we apply Dropout, Layer Norm, and residual connections similar to [135] and [32]. We also add gradient checkpoints [22] after the MSA blocks to be more memory efficient. Hence by attending to all patches, the network encodes the importance of each patch to the given planning problem in its output.

3.2.4 Classifier:

Using the encoded importance, the classifier predicts if each anchor point is of interest to the current planning problem. This is efficiently implemented using a 1×1 convolution layer, similar to Faster R-CNN [114]. A mask is generated by setting $p \times p$ patch, centered around selected anchor points, to 1 on a matrix of size $H \times W$.

3.2.5 Orientation Prediction:

For the $SE(2)$ space, each positive anchor point is assigned an orientation generated by this layer. Like the classifier, this is implemented efficiently using a 1×1 convolution layer.

3.2.6 Path Planning

Any traditional or learning-based planner can find the path by searching the masked region. For the point robot, we sample uniformly across a square grid centered around each anchor point, while for the Dubins car, we sample uniformly across an ellipse whose major axis is oriented along the predicted orientation. In this work, we use variations of the Rapidly Exploring Random Trees (RRT) algorithm that guarantee optimality

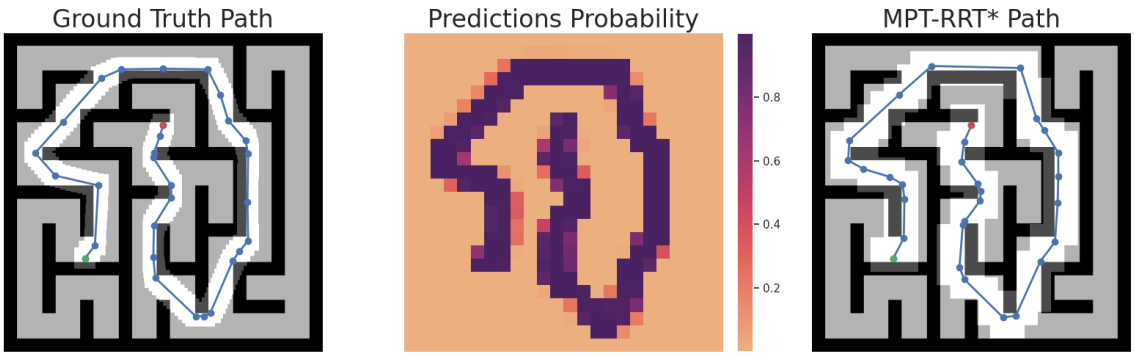

Figure 3.2. Planned path for the Maze environment. Left: The ground truth path for a given start (green) and goal (red) point from the validation data. Center: The prediction probability from the MPT. Right: Masked map and planned trajectory for the given start (green) and goal (red) point using MPT-RRT*.

[66, 85, 44] to find the path using the sampled nodes.

Occasionally, MPT-guided planners fail to generate a solution due to misclassifications. We alternate between searching the map's masked (exploitation) and unmasked (exploration) regions to overcome these errors. We show that this technique can preserve the benefits of MPT-guided planners while achieving higher efficiency in planning.

3.3 Experiments

We evaluated the planning capabilities of MPT aided SMPs and compared them with both traditional and learning-based planners. The following section will review our environment and robot setup, model training, and results.

3.3.1 Setup

Model Architecture: This section details the network architecture used for MPT in our experiments. The Transformer architecture is similar to the ones proposed in [135]. We used 6 layers of encoder block, each consisting of 3 heads. The dimension of the keys and queries was set at 512, and the dimension of the value was set at 256. The

architecture of our feature extractor is given in Table 3.1. For the convolution layer, the dimensions in the brackets represent [Input Channel Size, Output Channel Size, Kernel Size, Stride], and for the Maxpool layer, it represents the Kernel Size.

Table 3.1. Network architecture of the Feature Extractor

Layer	Dimension
2D Convolution	[2, 6, 5, 0]
2D Maxpool	[2]
ReLU	
2D Convolution	[6, 16, 5, 0]
2D Maxpool	[2]
ReLU	
Convolution	[16, 512, 5, 5]

The UNet architecture we use is similar to the one used by [115]. The network consists of 4 blocks of down convolution and 4 blocks of up convolution. Each down convolution block consists of a 3×3 convolution, followed by batch norm, ReLU, and a 2×2 of max pool layer. The up convolution block consists of a bilinear upsampling block, followed by a 3×3 convolutions, batch norm, and ReLU. The final layer is a 1×1 to classify each latent vector. For more details on the exact channel size used for each convolution.

Environments: To test the planning capabilities of our method, we evaluated the model on randomly generated maps from two different classes of environments. The first environment is called the Random Forest, where circular and square objects are randomly placed on the map. It simulates real world scenes with narrow passages and crowded spaces. The second environment is called the Maze environment. Each map from this environment is a perfect maze, generated using a randomized depth-first search. A characteristic of a perfect maze is that any start and goal pairs on this map are reachable by a collision-free path. The maze environment mimics long-horizon planning problems because even if the start and goal are geometrically close, the planner would have to take

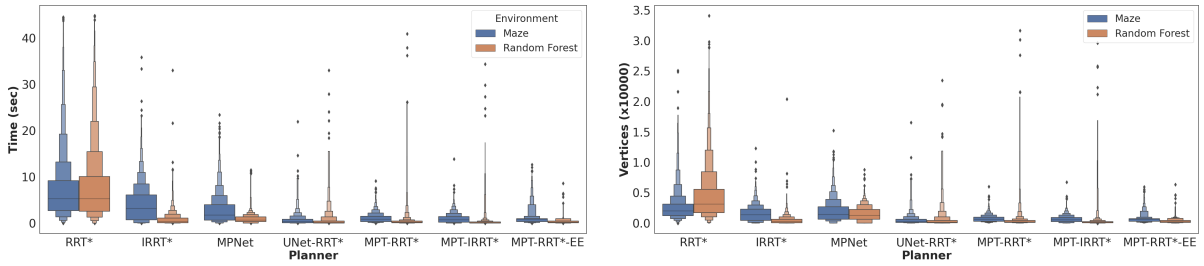

Figure 3.3. Planning statistics for the Point Robot Model. Left: The planning time for traditional and learning-based planners. Right: Number of vertices in the planning tree for traditional and learning-based planners. MPT aided planners consistently reduce the planning time and the vertices in the planning tree, resulting in a lower variance of these statistics for these planners.

a circuitous path to reach the goal. For all environments, we use an occupancy map of 5cm per pixel to check if the robot is in collision.

Robot Models We test our algorithm on two robotic systems encapsulating a large set of mobile systems. The first is a simple Point Robot Model that can move in any direction in its state-space $\mathcal{X} = \mathbb{R}^2$. Solutions for this model can be easily extended to many indoor and outdoor robots. For such robots, taking the Minkowski sum of the obstacles and the robot's footprint reduces it to a point robot [91]. The other robot we tested is a Dubins Car Model with state-space $\mathcal{X} = SE(2)$.

3.3.2 Training

The MPT model is trained in an end-to-end fashion under supervision similar to [47]. Each mini-batch is formed from a single planning problem containing positive and negative anchor points. We define positive anchor points as those within 0.7m distance to the trajectory, while all other anchor points are considered negative samples. For training, negative anchor points are randomly chosen to have a ratio of 1:1 of positive and negative samples. For the Dubins Car Model, each positive anchor point had an orientation associated with it. This orientation was the average of all the trajectory orientations within 0.7m of the path.

We trained the network by minimizing the cross-entropy loss for the anchor points and maximizing the cosine similarity measure for the orientation using the Adam optimizer [71] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 1e^{-9}$. We varied the learning rate as proposed in [135] with warm-up steps 3200. Each model was trained for 100 epochs with a batch size of 128. The models were trained on one machine with 4 NVIDIA 2080GTX graphics cards. The MPT model for the Point Robot took 21 hours, and the Dubins Car took 12 hours to train.

Datasets: We trained an MPT model for the Point Robot on 1750 randomly generated maps from each environment mentioned in Section 3.3.1. For each map, 25 paths were generated using the RRT* planner that terminated after 90 seconds. Similarly, we trained an MPT model for the Dubins Car on 900 randomly generated maps from the Random Forest environment. We collected 25 paths from each map using the RRT* planner using Dubins curve [35] as the node connector. All the maps used were of size 480×480.

3.3.3 Point Robot Model

For the Point Robot Model, we compared MPT-aided planners with traditional and learning-based planners. We also looked at the capabilities of other image segmentation approaches such as UNet [115] to highlight the area where a potential path might exist. We choose UNet because, like MPT, it can generalize to maps of different sizes. We call the aided planners Y-X, where Y is the underlying method used to generate the mask and X is the SMP planner. MPT-RRT*-EE represents the MPT aided RRT* planner with the exploration and exploitation strategy (hybrid planning).

The first set of experiments examined the network's ability to generalize to unseen maps of the same dimension as the training data. We compared the planners on 2500 random start and goal pairs for maps from the Maze and Random Forest environment. For each planner, we report the 1. Accuracy - the percentage of planning problems the

Table 3.2. Comparing planning accuracy, and median time and vertices for the Point Robot Model on unseen environments of the same size as the training data for Random Forest.

Algorithm	Accuracy	Time (sec)	Vertices
RRT*	100%	5.448	3228
IRRT*	100%	0.425	267
BIT*	100%	0.477	819
UNet-RRT*	30.27%	0.167	168
UNet-RRT*-EE	100%	2.58	1913
MPNet	92.35%	0.296	63
MPT-RRT* (ours)	99.40%	0.194	233
MPT-IRRT* (ours)	99.40%	0.087	136
MPT-RRT*-EE (ours)	100%	0.211	247

Table 3.3. Comparing planning accuracy, and median time and vertices for the Point Robot Model on unseen environments of the same size as the training data for Maze.

Algorithm	Accuracy	Time (sec)	Vertices
RRT*	100%	5.364	2042
IRRT*	100%	3.139	1394
BIT*	100%	2.870	2002
UNet-RRT*	21.4%	0.346	277
UNet-RRT*-EE	100%	4.133	2139
MPNet	71.76%	1.727	1409
NEXT-KS	28.27%	3.021	387
MPT-RRT* (ours)	99.16%	0.870	626
MPT-IRRT* (ours)	99.16%	0.784	566
MPT-RRT*-EE (ours)	100%	0.869	585

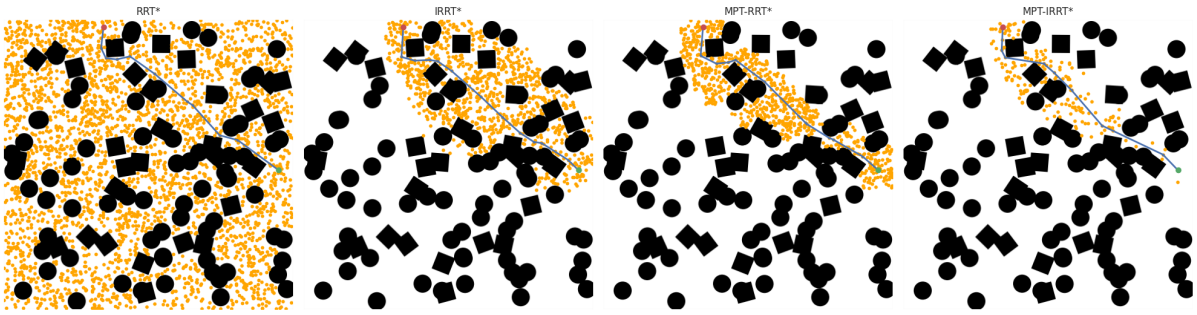

Figure 3.4. (From left) Path planned by RRT*, IRRT*, MPT-RRT*, and MPT-IRRT* for the same start (green) and goal (red) positions for the Random Forest environment. MPT-aided planners can significantly reduce the number of vertices (orange) required to search for a path.

planner solves, 2. Time (sec) - the amount of time it takes to generate the mask (if applicable) and plan a path shorter or of equal length to a path from the RRT* planner searching for 90 seconds, and 3. Vertices - the number of collision-free states sampled by the planner to construct the planning tree. The summary statistics of the experiment are reported in Table 3.2 and 3.3. We see that MPT aided planners reduce the planning time and vertex count of the planning tree substantially. In Fig. 3.2, we show an example of a planned path from the maze environment.

To better understand the advantages of MPT, we visualize the distribution of the planning time and vertices in Fig. 3.3 for the Random Forest and Maze environment using Letter-value plots. These plots help to observe the tail of the distribution of the metrics. Naive RRT* has a heavier tail distribution than MPT-RRT* because, for start and goal pairs further away from each other, the planner needs to generate a denser graph to search a larger space, requiring more time and vertices. On the other hand, MPT-aided planners focus their search near regions highlighted by the model, and as a result, they plan faster with fewer vertices. We see a thin tail distribution for MPT-RRT* and MPT-IRRT* planners for the random forest environment. This is because the planner only terminates when the cost of the path is below a certain threshold. Even if a solution is found, MPT planners continue to search to reduce the path length, resulting in longer

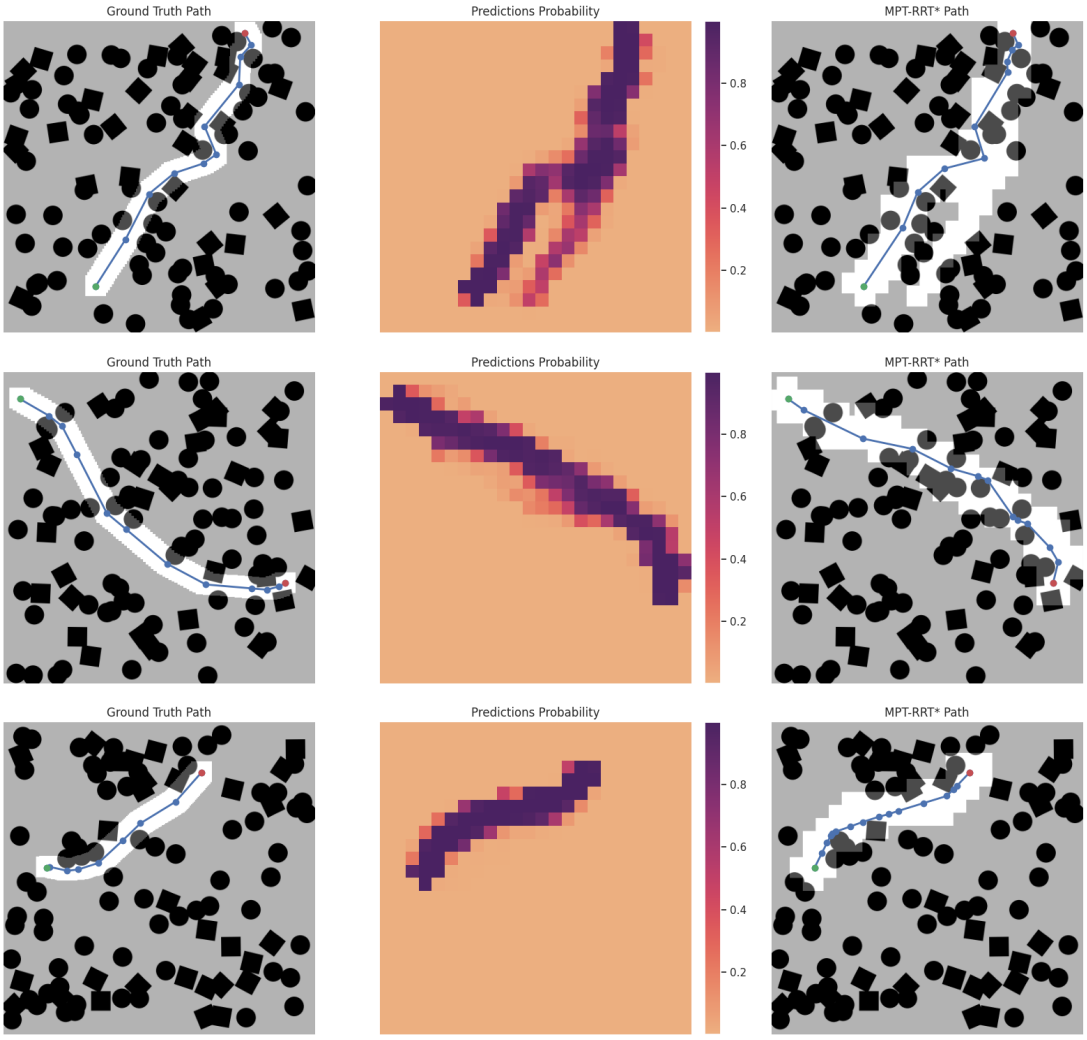

Figure 3.5. Three different trajectories planned successfully using MPT on the Random Forest environment. Left Column: The ground truth path in the validation dataset. Middle Column: The probability estimate made by the MPT. Right Column: The planned path using RRT* on the region proposed by the MPT.

planning times and vertex count for a few trajectories. The number of such problems accounts for less than 0.75% of the planning problems.

We also observe that IRRT* and BIT* achieve similar planning time and planning tree vertices compared to the aided planners. This is because these planners, like MPT, reduce the planning search space once an initial solution is found by bounding the initial path with an Ellipse. Such heuristics do not work for long-horizon problems like the Maze

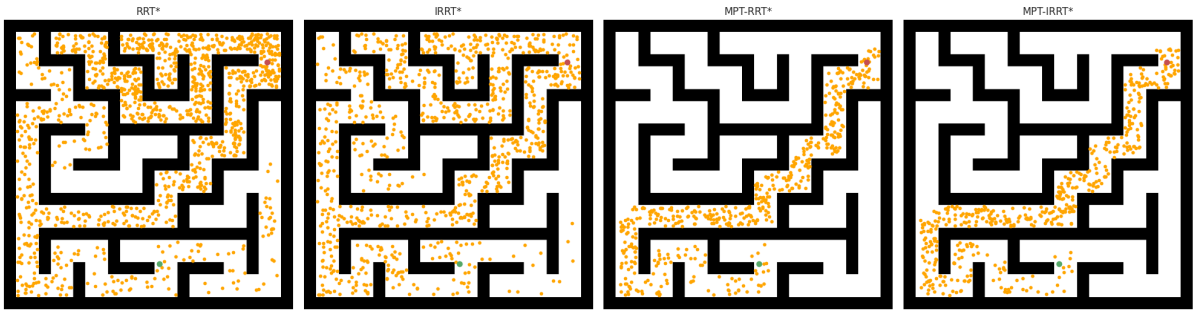

Figure 3.6. (From left) Path planned by RRT*, IRRT*, MPT-RRT*, and MPT-IRRT* for the same start (green) and goal (red) positions for the Maze environment. MPT aided planners can significantly reduce the number of vertices (orange) required to search for a path. The IRRT* planner must search the entire space for these maps.

environment, and MPT aided planners outperform traditional methods.

The MPT planners also outperform other learning-based approaches. Planners that used UNet to propose patches performed poorly. We believe this is because the convolution layers can only learn the connections between local patches, and deeper networks would be required to learn global connections. As a result, it fails to highlight the area of interest for the given planning problem. We also tested the recently proposed Exploration-Exploitation Tree with kernel smoothing (NEXT-KS) [20] on the Maze environment. The same validation set from Table 3.3 was used, but the map size was reduced from 480×480 pixels to 16×16 pixels. The drop in performance can be attributed to the value function proposing samples that are locally optimum. We do not compare the planners performance in the forest environment because reducing the map size reduces the distance resolution from 0.05meter/pixel to 0.75meter/pixel, which if applied to the Forest environment, would oversimplify collision free regions. MPNet, on the other hand, performs considerably better than UNet-RRT* for both environments but not as well as MPT planners. We attribute the weak generalization to the lack of training data. In [105], the authors used nearly 400k trajectories to train their model, whereas we only provided around 88k trajectories for all our models.

Due to classification errors, MPT fails on around 1% of maps. Nevertheless, by

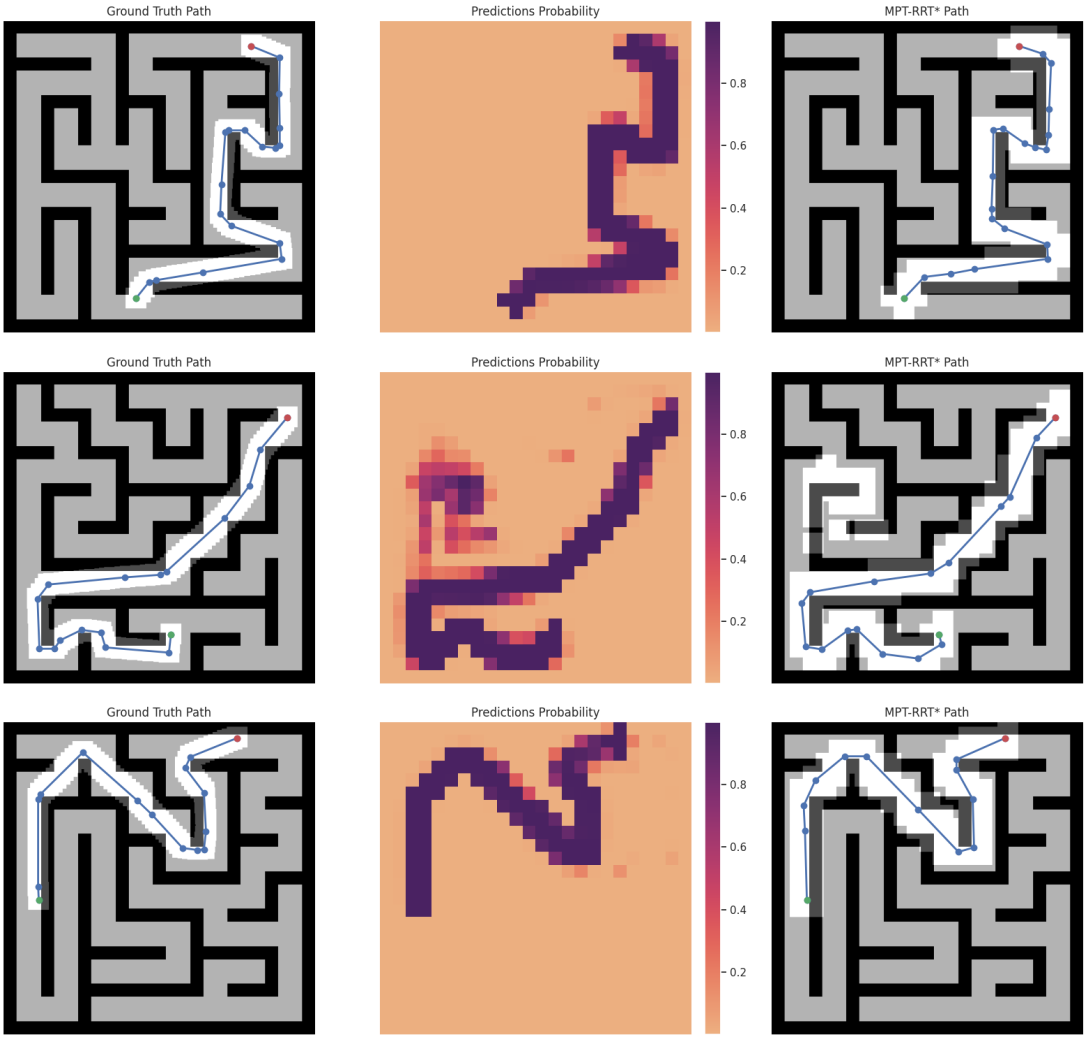

Figure 3.7. Three different trajectories planned successfully using MPT on the Maze environment. Left Column: The ground truth path in the validation dataset. Middle Column: The probability estimate made by the MPT. Right Column: The planned path using RRT* on the region proposed by the MPT.

randomly exploring the map for a few samples outside the segmented region, MPT-RRT*-EE can find valid trajectories without additional costs. UNet-RRT*-EE, on the other hand, has similar statistics to RRT*, implying that the planner relies on RRT* alone to generate a path. The distribution of planning time and vertices for MPT-RRT*-EE are much tighter than traditional planners and even MPT planners without exploration.

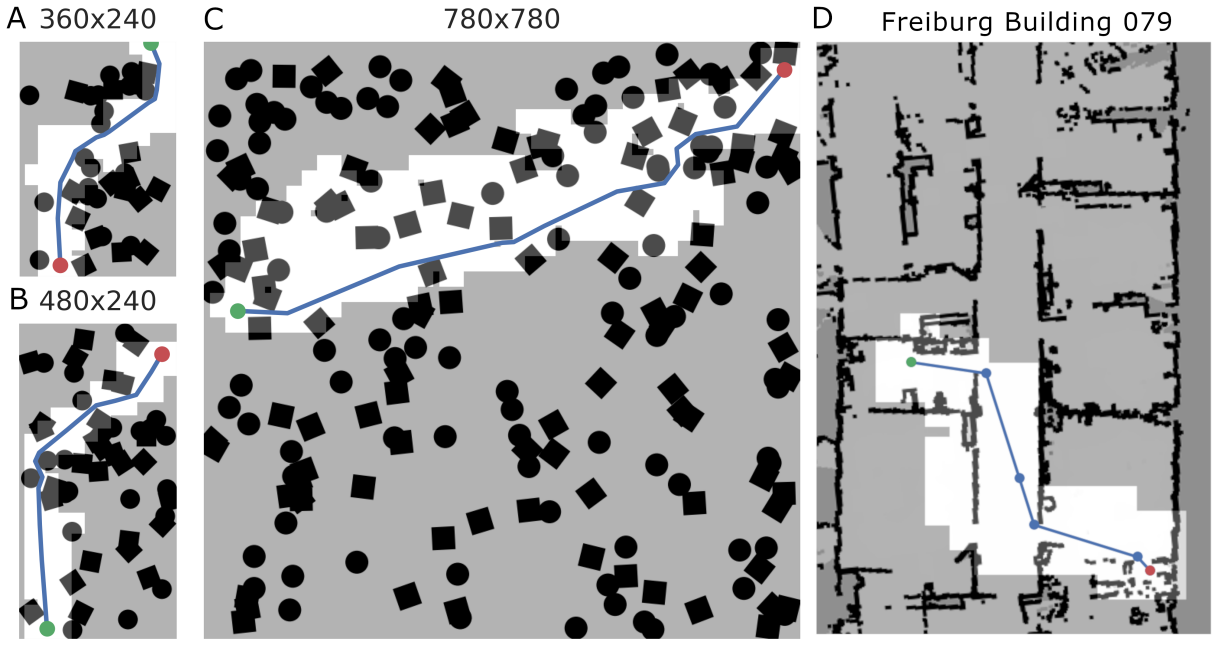

Figure 3.8. Plots of MPT aided planning for out-of-distribution environments. A, B, C: Plot of paths for Random Forest environments of different sizes. The architecture of the MPT Model allows flexibility in planning for environments of different sizes. D: Path generated by MPT-RRT* on a random start and goal pair on the map of building 079 at the University of Freiburg.

The following experiment we conducted evaluated MPT's ability to generalize to map sizes that were not part of the training data. We tested the model on four different map sizes, 360×240, 480×240, 560×560, and 780×780 of the Random Forest environment on 1000 randomly generated maps while maintaining the density of obstacles. We used the same MPT model without re-training or fine-tuning and compared the metrics with traditional planning techniques. We did not compare against learning-based planners because these planners either performed poorly in our previous experiment or were not generalizable to maps of different sizes without modifications. The planning statistics are summarized in Table 3.4, and three successfully planned paths using MPT are shown in Fig. 3.8 (A, B, C). Training the model by randomly shifting position encoding was instrumental in improving the accuracy of the planner for maps of larger sizes. Results for the model trained with fixed position encoding (F.E.) from [135] are given in Table 3.4.

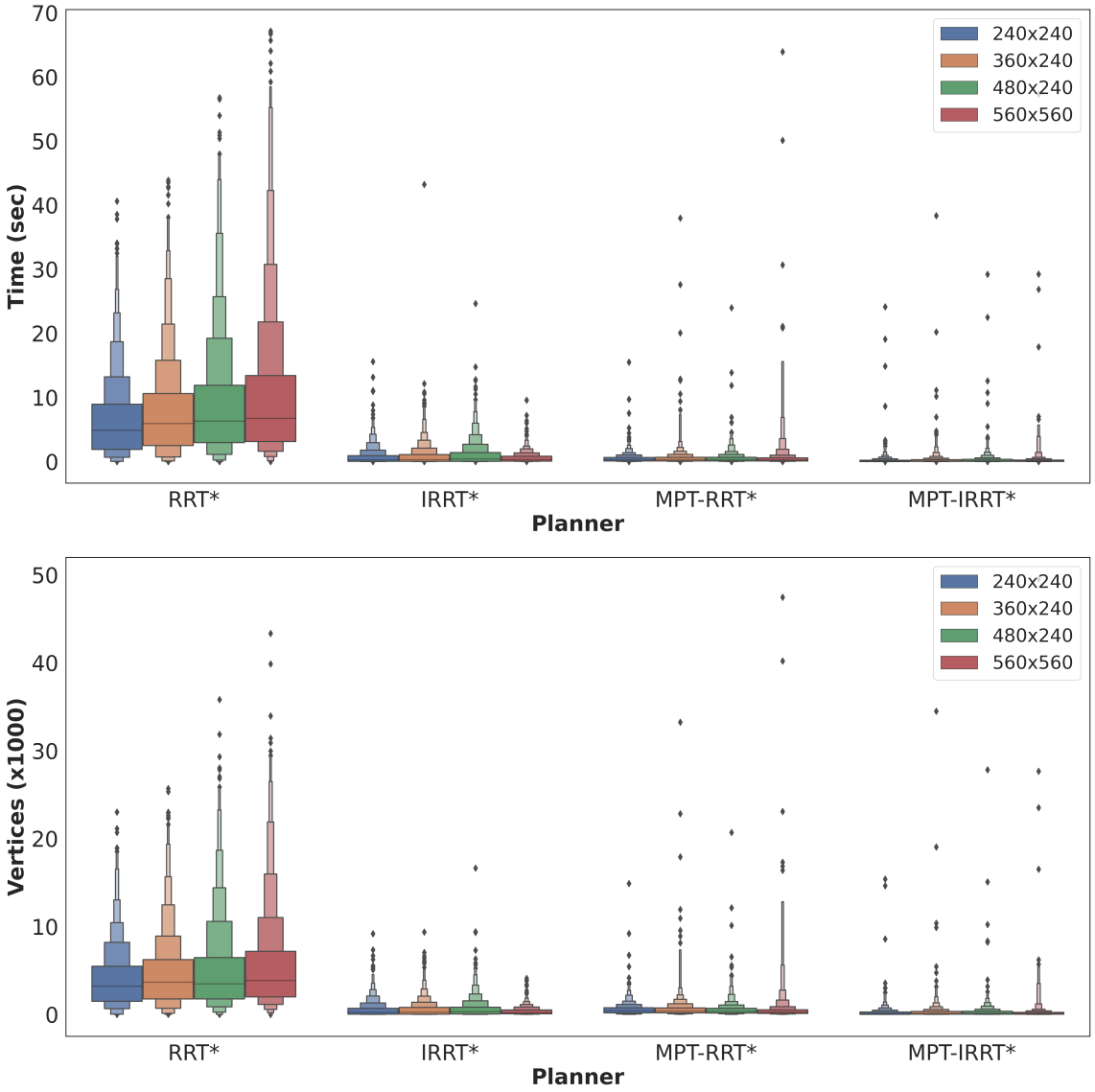

Figure 3.9. The distribution of metrics for maps of different sizes. Top: The distribution of planning time for different algorithms for maps of different sizes. Bottom: The distribution of the number of vertices in the planning tree for different algorithms for maps of different sizes. MPT aided planners can achieve faster planning times and lower variance in the metrics across maps of different sizes, which were not part of the training data.

We observe that the MPT model trained with randomized position encoding achieves nearly 61% more accuracy on larger maps while improving 1-4% on smaller maps.

We notice that the MPT aided planners, similar to our previous experiments, achieve lower planning time and vertices count than traditional planners without additional

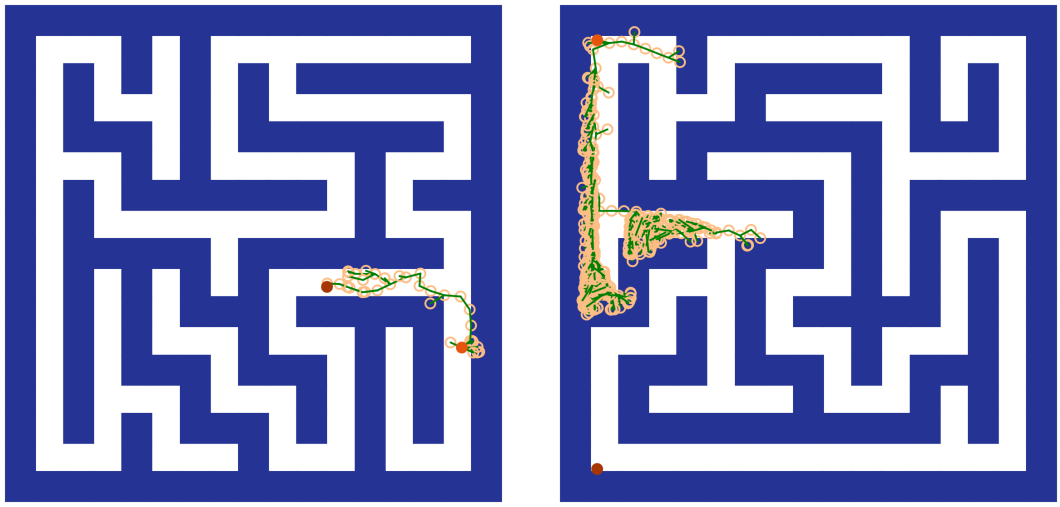

Figure 3.10. Plots of two paths for the modified maze environment using the NEXT-KS planner. The yellow circles indicate the sampled points by the planner. The planner can solve simple problems (Left), while for long-horizon problems, it gets stuck in local minimums (Right).

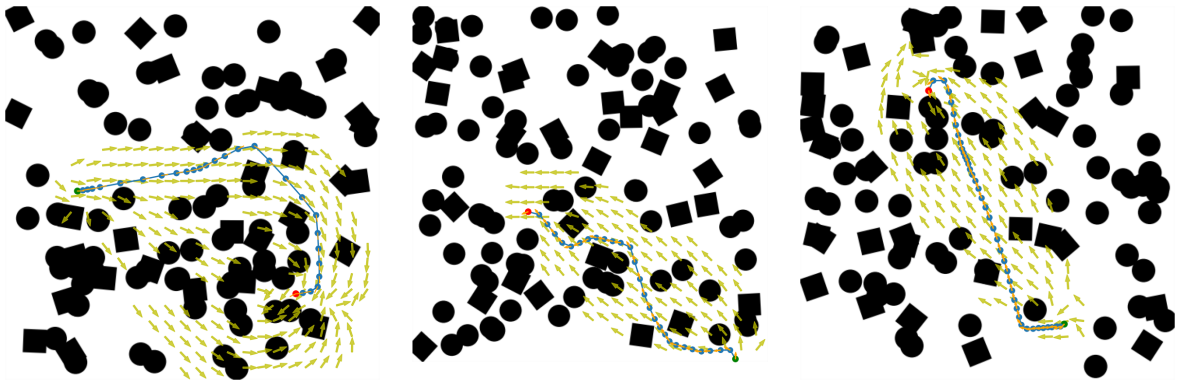

Figure 3.11. MPT can also be trained to aid SMP planners for non-holonomic robots. Planned paths on Random Forest environment using MPT-RRT*. MPT identifies regions in $SE(2)$ through which a non-holonomic path exists.

training or fine-tuning for larger maps. IRRT* and BIT* achieve similar performance to MPT-aided planners for smaller-sized maps but as the map sizes grow, the time taken by IRRT* grows because of the more prominent search space. While for larger maps, MPT aided planners can find a solution faster and outperform IRRT* and BIT*.

We also tested the MPT model on a map of building 079, University of Freiburg from

Table 3.4. Comparing planning accuracy and median time and vertices for Point Robot on maps of the different sizes.

Planner		Map Size (# Obstacles) 360×240 (35)	480×240 (50)	560×560 (100)	780×780 (200)
RRT*	Accuracy	100 %	100%	100%	100%
	Time (sec)	5.926	6.308	6.725	8.095
	Vertices	3660	3480	3854	4292
IRRT*	Accuracy	100%	100%	100%	100%
	Time (sec)	0.286	0.394	0.283	0.476
	Vertices	257	291	203	255
BIT*	Accuracy	100%	100%	100%	100%
	Time (sec)	0.625	0.590	0.397	0.542
	Vertices	1069	1061	810	974
MPT-RRT* (F.E.)	Accuracy	97.4%	96.3%	75.6%	37.9%
	Time (sec)	0.265	0.268	0.253	0.274
	Vertices	377	348	262	284
MPT-IRRT* (F.E.)	Accuracy	97.4%	96.3%	75.6%	37.9%
	Time (sec)	0.062	0.072	0.082	0.095
	Vertices	118	130	101	142
MPT-RRT*	Accuracy	99.20%	98.5%	99.7%	99.5%
	Time (sec)	0.248	0.265	0.181	0.297
	Vertices	354	319	218	241
MPT-IRRT*	Accuracy	99.2%	98.5%	99.7%	99.5%
	Time (sec)	0.054	0.073	0.083	0.152
	Vertices	106	131	112	161
MPT-RRT*-EE	Accuracy	100%	100%	100%	100%
	Time (sec)	0.297	0.302	0.217	0.285
	Vertices	382	362	237	238

publicly available 2D lidar scan data. We used the MPT model from our previous experiments without any additional training or fine-tuning. The results are summarized in Table 3.5. We plot one of these trajectories in Fig. 3.8 (D). The ability of MPT to generalize to real world maps without any additional training or fine tuning is exciting because it can be trained on simulated environments and transferred to real world

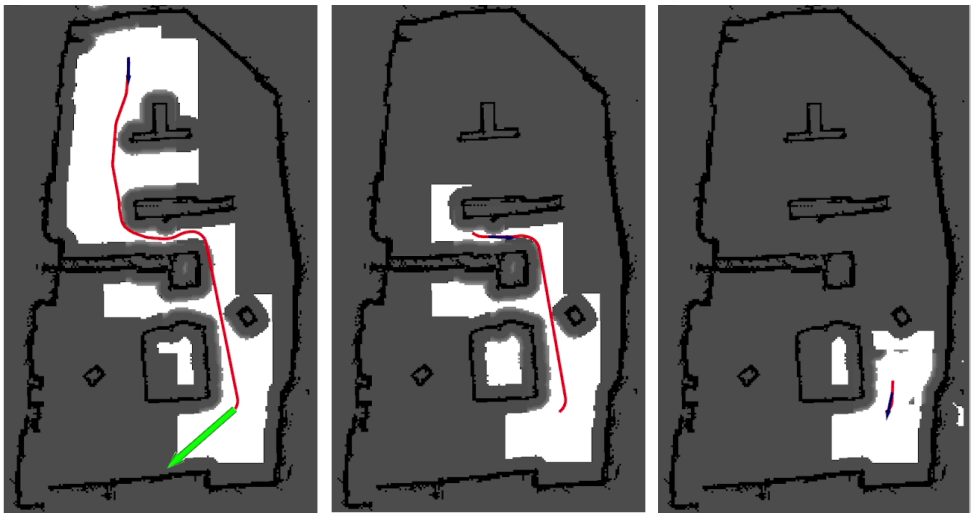

Figure 3.12. Execution of trajectory from an MPT aided planner using the Nav2 stack. For a given goal position (green arrow), the MPT model proposes a region from the current position of the robot (blue arrow) to the goal, and the Hybrid A* planner generates a global path (red). The MPT model can aid the underlying planner in real time.

applications with minimal effort.

3.3.4 Dubins Car Model

To evaluate the performance of MPT for the Dubins car model, we compared our method with RRT* and SST planners. For the geometric planning, we used Dubins curves [35] as edge connectors, while SST nodes were constructed by propagating wheel velocity and steering angle. The parameters for SST were set as reported in [86]. The metrics we report are the same as the point robot. The results are summarized in Table 3.6. We see that MPT-aided planners are able to reduce planning time and sampled points by half, while the hybrid planning strategy is able to achieve 100% accuracy without adding additional vertices.

Fig. 3.11 shows two paths planned using MPT for the Dubins car. We can observe that the MPT model can predict regions through which a kinematically feasible path can pass. By narrowing the search space, the planner can generate optimal paths with fewer samples than the unaided RRT* planner. A comparison for points generated between

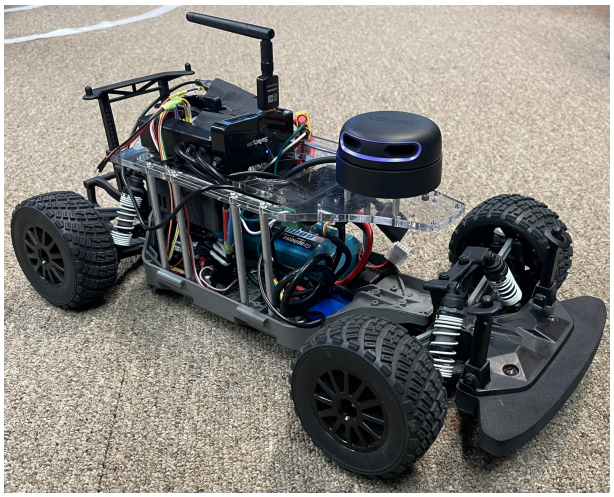

Figure 3.13. The robot car used for the real world experiment. We used the F1Tenth platform for our robot system [96] and Navigation 2[93] planning stack for planning.

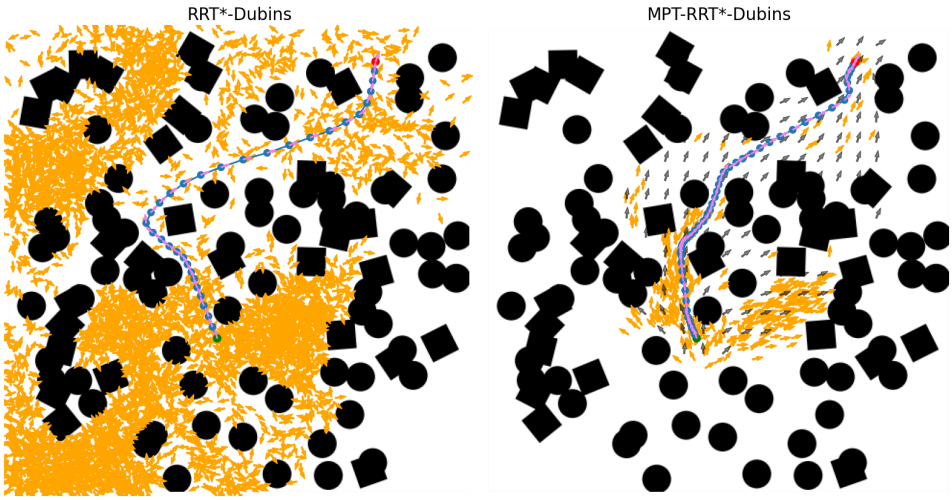

Figure 3.14. Vertices used by RRT* (left) and MPT-RRT* (right) for the same start (green) and goal (red) positions in the $SE(2)$ space. By guiding the sampling around anchor points (gray), the planner can achieve a shorter path with far fewer samples.

RRT* and MPT-RRT* is shown in Fig. 3.14. On the other hand, the SST planner takes more time and vertices to generate a valid path because more samples are required to traverse through the narrow sections of the map.

Table 3.5. Comparing planning accuracy, and median time and vertices for the Point Robot Model on real world map.

Planner	Accuracy	Time (sec)	Vertices
RRT*	20/20	2.507	1630
MPT-RRT*	17/20	0.538	455
MPT-RRT*-EE	20/20	0.946	804

Table 3.6. Comparing planning accuracy, and median time and vertices for Dubins Car Model for the Random Forest Environment.

Planner	Accuracy	Time (sec)	Vertices
RRT*	100%	0.357	95
SST	100%	4.880	710
MPT-RRT*	95.15%	0.176	59
MPT-RRT*-EE	100%	0.197	60

3.4 MPT Navigation2 Plugin

We also provide the MPT model as a global planner plugin to the Navigation2 [93] (Nav2) navigation stack. The model is integrated in such a way that, any of the already existing planners in Nav2 could be used to generate a plan in the accentuated area proposed by the MPT model. We tested the plugin on an RC Dubins car in an indoor mapped environment. The model used by the plugin was the one trained using just the synthetic data for the point robot. The planning frequency was set at 10Hz, and the Hybrid A* planner with Dubins motion model was used to generate the final global plan. A simple DWB local planner was used to follow the planned global path. An example of a complete execution of a planning problem is shown in Fig. 3.12.

This experiment also highlights the sim-2-real generalization of the MPT model. Previous learning models would have to gather task-specific data [81, 54] or actively learn from failed trajectories[105] in order to adapt to new environments. The ability of MPT to generalize will also benefit the entire robotics community by making it easier to use previously trained models.

3.5 Discussions

Graph-based search methods such as A* can also be used to solve the given planning problem using the costmap, but in order to do real time planning, these methods often require sub-sampling of the costmap to provide solutions in real time. Further hand-tuned heuristics are required to restrict the planning space for kinematically constrained systems [31] otherwise they get stuck in local minima [80]. We have shown through our experiments that the MPT aided planners require no such sub-sampling, and can learn to restrict search spaces by leveraging data. Since MPT is agnostic to the underlying planner, graph-based searches can also be used to search the highlighted space.

Prior learning based methods that predicts a value such as VIN [128], are shown to be difficult to train for larger maps [95] because of the depth of layers that is required to propagate the reward. Unlike the other transformer based planner [18], as we have shown randomizing the positional encoding is critical to be able to generalize the planner to maps of different sizes. The map sizes that are used in our method is nearly 10 times larger than the ones shown use in [18]. Moreover, MPT combines best of both worlds by learning from data the space where a plan exists and using an optimal planner get the exact path.

In the next chapters, we extend MPT to higher-dimensional robots such as robotic arms or drones. For manipulation systems in 3D since the task and joint space do not overlap, the extension of the method is more challenging.

3.6 Acknowledgement

Chapter 3, in part, is a reprint of material from J. J. Johnson, U. S. Kalra, A. Bhatia, L. Li, A. H. Qureshi, and M. C. Yip "Motion Planning Transformers: A Motion Planning Framework for Mobile Robots," on arXiv preprint arXiv:2106.02791. The dissertation author is the primary author of this paper.

Chapter 4

Generalizing Transformers for Planning

In chapter 3 we introduced Motion Planning Transformers, which only solves for 2D mobile robots because, inherently, their network models follow those used in image understanding in 2D discrete spaces. Since these models have to discretize the entire planning space, extending this method to higher dimensional, continuous planning spaces would exponentially increase training and memory costs. Furthermore, these planners require the space in which the path is constructed (planning space) to overlap with the space in which the environment is represented (task space). For example, for a 14-degree-of-freedom bi-manual robot arm setup, the environment is represented using point clouds which is $\mathbb{R}^3$, while the planning space is $\mathbb{R}^{14}$.

In this chapter, we introduce Vector Quantized-Motion Planning Transformers (VQ-MPT), a scalable transformer-based model that accelerates SMP by narrowing the sampling space. VQ-MPT uses a Vector Quantized (VQ) model to discretize the planning space. VQ models are generative models with an encoder-decoder architecture similar to Variational AutoEncoder (VAE) models but with the latent dimension represented as a collection of learnable vectors referred to as dictionaries. A transformer model selects a subset of these learned vectors to generate the search region for the given planning problem. We describe in this paper how the VQ approach can be used in the context of

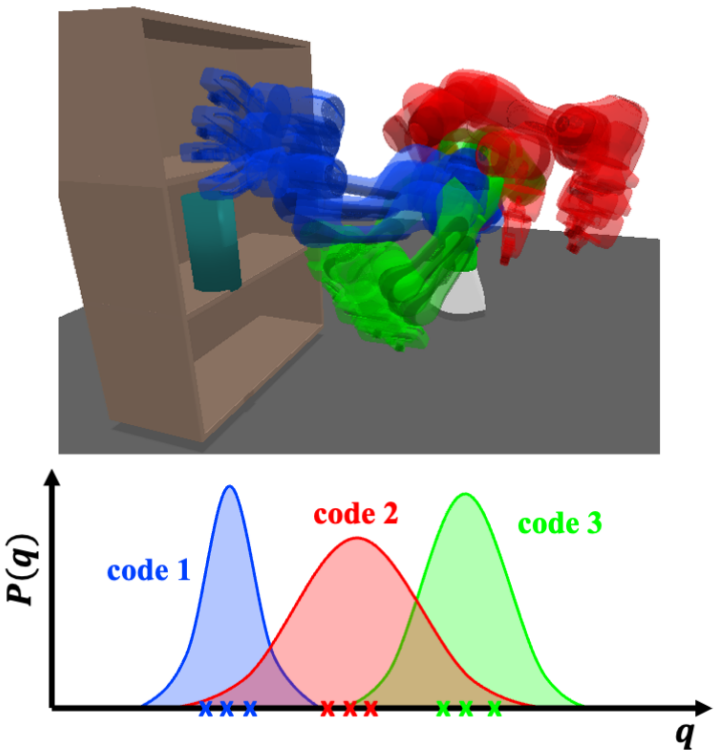

Figure 4.1. VQ-MPT can efficiently split high-dimensional planning spaces into discrete sets of distributions. Each distribution is represented using a latent variable called code or dictionary value. Given a planning problem, the model selects a subset of codes and samples from the associated distributions to construct the trajectory. By sampling efficiently, VQ-MPT reduces planning times by 2-6× compared to previous planners.

motion planning, leading to the following major advantages:

1. Reduces planning times by 2-6× compared to traditional planning algorithms such as BIT* and by 3-6× compared to learned planners such as MPNet.

2. Scales to 14-dimensional planning spaces without compromising planning performance.

3. Learns efficient quantization of high dimensional planning space without increasing the dictionary size.

4. Generalizes to unseen in-distribution and out-of-distribution environments more successfully than learned planners such as MPNet.

4.1 Related Works

For efficient sampling, prior works have reduced the search spaces through hand-crafted heuristics or parametric functions, decreasing planning time. The current state-of-the-art motion planners leverage goal-directed heuristics; Informed-RRT* (IRRT*) [44] and Batch Informed Trees (BIT*) [45] search for a path in an ellipsoidal region between the start and goal location. In [109, 127], Artificial Potential Fields (APF) guide random samples toward regions with an optimal solution. Sampling-based A* [80] extends the A* search algorithm to sampling-based planning and uses heuristics to sample from selected vertices. But for higher dimensional spaces, sampling with these heuristics still leaves many samples unused for constructing a trajectory.

On the other hand, learning-based methods leverage data from prior planned data to accelerate planning in similar environments [81, 17, 55, 76]. Motion Planning Networks (MPNet) [105] was the first neural planner to generate the full motion planning solution through a recurrent sampling of its networks, given the current and goal position of the robot as well as the environment representation. MPNet considerably reduces planning time for higher dimensions, but these models do not generalize to larger environment representations [60]. Other neural planners [20, 142] have also explored using neural networks for planning.

Transformer models are an ideal candidate for solving the planning problem because of their ability to make long-horizon connections [134]. Advances in large language models, such as BERT [30], and GPT [14], have inspired similar efforts in solving planning tasks using transformer models [21, 58]. These models make better control decisions in robotic quadrupedal walking tasks by attending to proprioceptive and visual sensor data [141]. Although these works support the possibility of using transformer models for decision-making, it is difficult to interpret the policy's future control actions and provide any form of guarantee for the underlying planner.

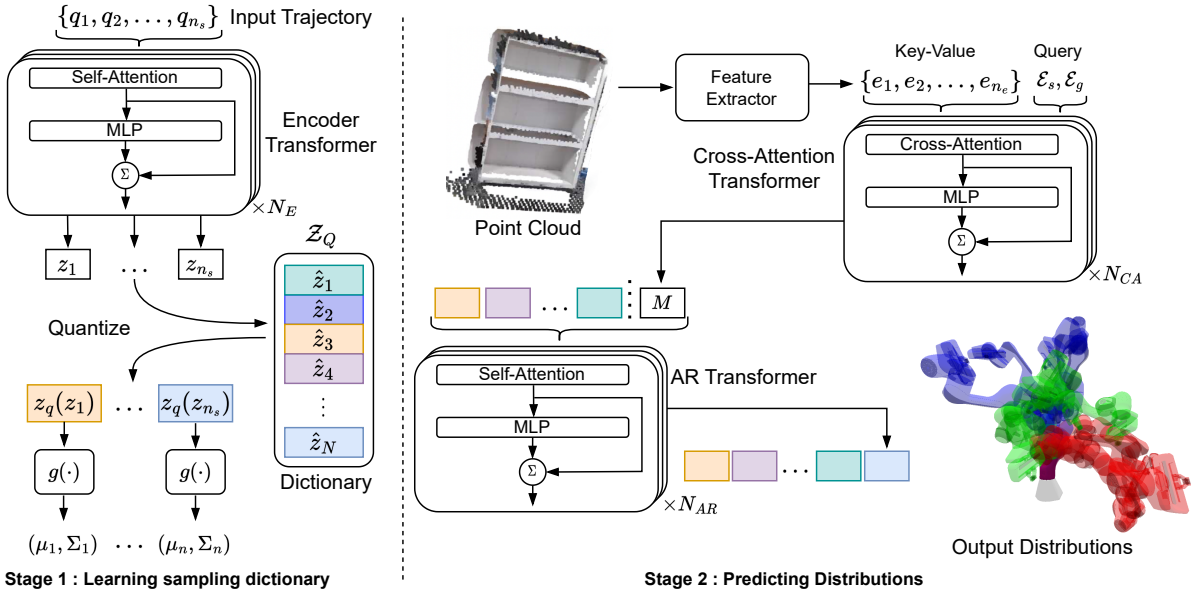

Figure 4.2. An outline of the model architecture of VQ-MPT. Stage 1 (Left) is a Vector Quantizer that learns a set of latent dictionary values that can be mapped to a distribution in the planning space. By encoding the planning space to discrete distributions, we can plan for high-dimensional robot systems. Stage 2 (Right) is the Auto-Regressive (AR) model that sequentially predicts the sampling regions for a given environment and a start and goal configuration. The cross-attention model transduces the start and goal embeddings given the environment embedding generated using a feature extractor. The output from the AR Transformer is mapped to a distribution in the planning space using the decoder model from Stage 1.

4.2 Background

4.2.1 Vector Quantized Models

The VQ-VAE model has been shown to compress high-dimensional spaces such as images and audio without posterior collapse observed in VAE models [133]. We utilize a VQ-VAE in a similar manner to compress the robot planning space $\mathcal{X}$. The VQ model encodes input $q \in \mathbb{R}^n$ using a function f to a latent space $\mathcal{Z}$, and is quantized to a set of learned vectors $\mathcal{Z}_Q = \{\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_N\}$. The vectors in $\mathcal{Z}_Q$ are often called codes or dictionary values in literature. The function g decodes the closest vector in $\mathcal{Z}_Q$ to $f(q)$ back to the input space. The parameters of f and g and the set of vectors in $\mathcal{Z}_Q$ are

estimated using self-supervised learning by minimizing the following error,

$$\mathcal{L} = \mathcal{L}_{recon} + \|\text{sg}[f(q)] - \hat{z}\| + \beta\|f(q) - \text{sg}[\hat{z}]\|, \tag{4.1}$$

where $\hat{z}$ is the quantized vector and sg[ ] stands for the stop gradient operator [133], which has zero partial derivatives, i.e. $\nabla \text{sg}(x) = 0$, preventing the operand from being updated during training. $\mathcal{L}_{recon}$ is the main AE reconstruction loss (we will derive this later). The second term is used to update the latent vectors in $\mathcal{Z}_Q$ while keeping the encoder output constant, and the last term is called the commitment loss and updates the encoder function while keeping the latent vectors constant. This prevents the output of the encoder from drifting away from the current set of latent vectors. Yu et al. [143] proposed two further improvements in representing the codes to help improve the training stability, code usage, and reconstruction quality of VQ-VAE models for images.

Factorized Codes

The output from the encoder function is linearly projected to a lower dimensional space. For example, if the encoder output is a 1024-d vector, it is projected to an 8-d vector. The authors in [143] show that using a lower dimensional space improves code usage and reconstruction quality.

Normalized Codes

Each factorized codes, $\hat{z}_i$, are l_2 normalized. Hence all the dictionary values are mapped onto a hypersphere. This improves the training stability and reconstruction quality of the model.

4.2.2 Transformer Models

Transformer models are transduction models that consist of self-attention [87] and fully connected layers. They have been shown to efficiently model sequence data for

language and image tasks [134, 33], hence an ideal encoder model. The self-attention layer is a Scaled Dot-product Attention [134] that takes three matrices - query ($Q \in \mathbb{R}^{n_s \times d_q}$), value ($V \in \mathbb{R}^{n_s \times d_v}$), and key ($K \in \mathbb{R}^{n_s \times d_q}$) vectors to generate the attention output

$$\text{Atten}(Q, K, V) = \text{softmax}\left(\gamma^{-1} Q K^T\right) V, \tag{4.2}$$

where n_s is the sequence length, d_q is the dimension of the query space, d_v is the dimension of key and value space, and $\gamma = \sqrt{d_v}$ is a scaling factor. Rather than doing a single attention function, these models linearly project the query, key, and value vectors multiple times using different learned weights and is called the multi-headed attention model. This enables the model to attend to different features present in the data. The final output is a linear combination of individual attention values evaluated on each projected set. The pooled output is passed through deep residual multilayer perceptron (MLP) networks. In [140], the authors introduce Prenorm-Transformer where the inputs to the attention and MLP layers are normalized as this makes training the model more stable.

4.3 Vector Quantized-Motion Planning Transformers

The VQ pipelines in image generation [113, 143] consist of a quantization stage and a prediction stage. We adapt this pipeline for sequence generation and represent the planning space as a collection of distributions (Fig. 4.2). Below, we describe the two stages and objectives used for training.

4.3.1 Stage 1: Vector Quantizer

The first stage learns to represent the planning space using a set of distributions. It does not take any sensor data such as costmap or pointcloud. We use a VQ model similar

to VQ-VAE [133] with a transformer network as the encoder and propose a maximum likelihood-based reconstruction loss to learn the set of distributions. The encoder network takes in a trajectory, $\mathcal{Q} = \{q_1, q_2, \ldots, q_{n_s}\}$, and outputs a set of latent vectors, $\mathcal{Z} = \{z_1, z_2, \ldots, z_{n_s}\}$. The decoder model, an MLP model, maps the quantized encoder output to a sequence of parameterized distributions, $\{P(\cdot\ ;\ \theta_1), P(\cdot\ ;\ \theta_2), \ldots, P(\cdot\ ;\ \theta_{n_s})\}$, in the planning space. We define our reconstruction loss as follows:

$$\mathcal{L}_{recon} = -\sum_{j=1}^{n_s} \log(P(q_j\ ;\ \theta_j)) - \lambda \sum_{j=1}^{n_s} \mathbb{E}_{q\sim\mathcal{X}}[-\log(P(q;\ \theta_j))] \tag{4.3}$$

where λ is a scaling constant. The first term maximizes the likelihood of observing the input trajectory, while the second term maximizes the differential entropy. The entropy term prevents the distribution from overfitting to each batch of data because a small batch size does not cover the entire planning space. In the following paragraphs, we provide further details of our models.

The encoder model transforms each state in the trajectory into an efficient representation by learning patterns in the sequence. Each input state, q_j, to the encoder is linearly projected to a latent space $\mathbb{R}^d$, and fixed position embedding [134] is added to the projected output. The resulting vector is passed through multiple blocks of Prenorm-Transformer described in Section 4.2.2 to obtain the set $\mathcal{Z}$. Each latent vector $z_j \in \mathcal{Z}$ is quantized to a vector from the set $\mathcal{Z}_Q = \{\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_N\}$ using the function $z_q(\cdot)$ defined by:

$$z_q(z) = \hat{z}_i \quad \text{where} \quad i = \underset{k\in\{1,\ldots,N\}}{\arg\min} \|z - \hat{z}_k\| \tag{4.4}$$

where $\hat{z}_i$ is the quantized vector corresponding to q_i. We prepend and append the transduced set with static encodings z_s and z_g to indicate the start and end of the sequence, respectively. Hence the robot trajectory $\mathcal{Q}$ is transduced to $\hat{\mathcal{Z}} = \{z_s, z_q(z_1), z_q(z_2), \ldots z_q(z_{n_s}), z_g\}$.

The decoder model maps each quantized vector, $z_q(z_i)$, to the parameterized distribution $P(\cdot\ ;\ \theta_i)$. We choose the output distribution as Gaussian, but any parametric distribution, such as Gaussian Mixture Models, Exponential distributions, or Uniform distributions, can be chosen. The decoder model outputs the mean and the covariance matrix of the Gaussian distribution ($\mathcal{N}(\mu, \Sigma)$); hence it is a function of the dictionary value $z_q(z_j), \forall j \in \{1, \ldots, n_s\}$, and is represented by $\mu(z_q(z_j))$ and $\Sigma(z_q(z_j))$ respectively. We will refer to these variables as μ_j and Σ_j for simplicity.

To ensure that the covariance matrix always remains positive definite during training, we decompose Σ_j using Cholesky decomposition as in previous works [51, 88]:

$$\Sigma_j = L_j D_j L_j^T \tag{4.5}$$

where L_j is a lower triangle matrix with ones along the diagonal, and D_j is a diagonal matrix with positive values. The output from the penultimate MLP layer is passed through separate linear layers to obtain μ_j and L_j, while for D_j, it is passed through a linear and soft-plus layer [37] to ensure values are positive. Using the soft-plus layer improves the stability of training the model.

4.3.2 Stage 2: Auto-Regressive (AR) Prediction

The second stage generates sampling regions by predicting indexes from the dictionary set $\mathcal{Z}_Q$ for a given planning problem and sensor data. It comprises two models - a cross-attention model to embed start and goal pairs and the environment embedding into latent vectors (M), and a Transformer-based Auto-Regressive (AR) model to predict the dictionaries indexes, $\mathcal{H} = \{h_1, h_2, \ldots h_{n_h}\}$. Both models are trained end-to-end by reducing the cross entropy loss using trajectories from an RRT* planner:

$$\mathcal{L}_{CE} = \mathbb{E}[-\sum_{j=1}^{n_h}\sum_{i=1}^{N+1} \delta_i(h_j) \log(\pi(h_j = i|\hat{z}_{h_1}, \cdots, \hat{z}_{h_{j-1}}, M))] \tag{4.6}$$

where $\delta_i(\cdot)$ is the Kronecker delta function, $\pi(\cdot)$ is the output of the AR model, and $\hat{z}_{h_i}$ corresponds to the latent dictionary vector associated with the ground truth index h_i, and the expectation is over multiple trajectories. We provide more details of the models in the following section.

The environment representation (i.e., costmap or point cloud data) is passed through a feature extractor to construct the environment encodings $\mathcal{E} = \{e_1, e_2, \ldots, e_{n_e}\}$ where $e_i \in \mathbb{R}^d$. The feature extractor reduces the dimensionality of the environment representation and captures local environment structures as latent variables using convolutional layers for costmaps and set-abstraction layers for point clouds. The start and goal states (q_s and q_g) are projected to the start and goal embedding ($\mathcal{E}_s \in \mathbb{R}^d$ and $\mathcal{E}_g \in \mathbb{R}^d$) using a MLP network. The cross-attention model is a Prenorm-Transformer model that uses the environment embedding, $\mathcal{E}$, and the start and goal embedding, $\{\mathcal{E}_s, \mathcal{E}_g\}$ to generate latent vectors M. The cross-attention model learns a feature embedding that fuses the given start and goal pair with the given planning environment. It uses the vector in $\mathcal{E}$ as key-value pairs, and $\mathcal{E}_s$ and $\mathcal{E}_g$ as query vectors to generate M.

We use an AR Transformer model, $\pi(\cdot)$, to predict the dictionary indexes $\mathcal{H}$. A Transformer-based AR model was chosen because of their ability to make long-horizon connections. For each index h_j, the model outputs a probability distribution over $\mathcal{Z}_Q \cup \{z_g\}$ given dictionary values of previous predictions $\{\hat{z}_{h_1}, \hat{z}_{h_2}, \ldots, \hat{z}_{h_{j-1}}\}$ and the planning context M:

$$\pi(h_j = i | \hat{z}_{h_1}, \ldots, \hat{z}_{h_{j-1}}, M) = p_i \quad \text{where} \sum_{i=1}^{N+1} p_i = 1 \tag{4.7}$$

Using the learned decoder from Stage 1, we can convert each of the predicted dictionary values, $\hat{z}_{h_j}$, into a Gaussian distribution ($\mathcal{N}(\mu_{h_j}, \Sigma_{h_j})$) in the planning space.

Table 4.1. Model and environment parameters for each robot

Robot	Environment Representation	d	Dictionary Keys	d_k	d_v
2D	Costmap	512	1024	512	256
7D	Point Cloud	512	2048	512	256
14D	Point Cloud	512	2048	512	256

4.3.3 Generating Distributions for Sampling

With stage 1, we have efficiently split the planning space into a discrete set of distributions represented using a set of latent vectors, and with stage 2, we have provided a means to select a subset of distributions from the dictionary. Given a new planning problem, we use the trained Stage 2 models to generate a sequence of dictionary indexes $\mathcal{H} = \{h_1, \dots h_{n_h}\}$. Since each index can take N values, we pick the sequence $\mathcal{H}$ that maximizes the following probability:

$$P(h_1, \dots, h_{n_h}|M) = \prod_{i=1}^{n_h} \pi(h_i|h_1, \dots, h_{i-1}, M) \tag{4.8}$$

where h_{n_h} is the goal index and π is the probability from Eqn. 4.7. We apply a beam-search algorithm to optimize for Eqn. 4.8 as done before in language model tasks [30].

The decoder model from Stage 1 is used to generate a set of distributions, $\mathcal{P}$, from the dictionary values, $\{\hat{z}_{h_1}, \hat{z}_{h_2}, \dots, \hat{z}_{h_{n_h-1}}\}$, corresponding to the predicted indexes $\{h_1, h_2, \dots, h_{n_h-1}\}$. We define this set as a Gaussian Mixture Model (GMM) with uniform mixing coefficients:

$$\mathcal{P}(q) = \sum_{i=1}^{n_h-1} \frac{1}{n_h - 1} \mathcal{N}(\mu(\hat{z}_{h_i}), \Sigma(\hat{z}_{h_i})) \tag{4.9}$$

An example of this distribution is in Fig. 4.3 for a 2D robot.

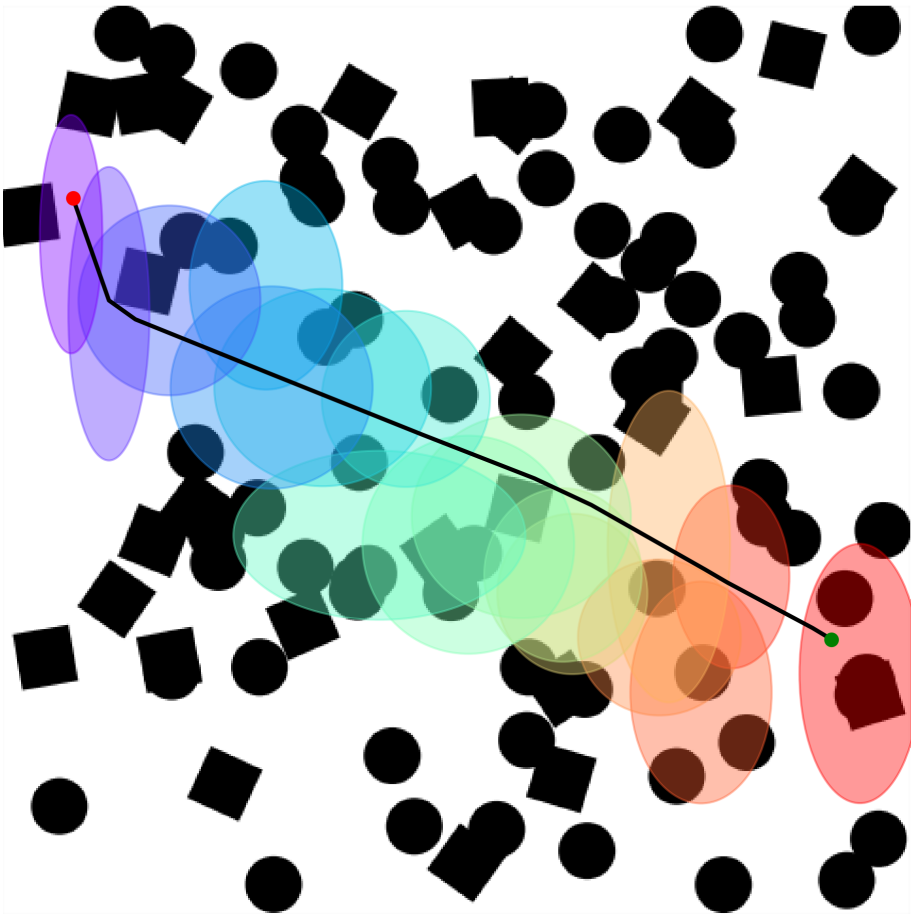

Figure 4.3. A trajectory (black) planned using VQ-MPT for the 2D robot and the corresponding GMM used for sampling. Each ellipse represents the distribution encoded by the dictionary values. The shaded region represents the 2 standard deviation confidence interval region. The dictionary values can encode the planning space using a finite number of vectors.

Table 4.2. Comparing accuracy and mean planning time and vertices in In-Distribution environments

Planner		Robots 2D	Robots 7D	Robots 14D
RRT*	Accuracy	94.8%	52.80%	11.80%
	Time (sec)	1.588	49.35	1.80
	Vertices	1195	683	9
RRT*(50%)	Accuracy	·	95.20 %	32.00%
	Time (sec)	·	10.51	15.03
	Vertices	·	149	94
IRRT*	Accuracy	97.40%	89.00%	21.80%
	Time (sec)	0.244	54	52.84
	Vertices	195	63	45
IRRT*(50%)	Accuracy	·	94.80 %	40.40%
	Time (sec)	·	15.03	29.16
	Vertices	·	71	77
BIT*	Accuracy	96.00%	72.20%	30.80%
	Time (sec)	0.297	7.58	9.56
	Vertices	457	826	384
BIT*(50%)	Accuracy	·	97.40 %	43.40%
	Time (sec)	·	5.26	39.09
	Vertices	·	640	2021
MPNet	Accuracy	92.35%	94.20%	92.20%
	Time (sec)	0.296	5.18	17.46
	Vertices	**63**	147	117
VQ-MPT	Accuracy	**97.60%**	**97.40%**	**99.20%**
	Time (sec)	**0.147**	**0.929**	**2.62**
	Vertices	306	**45**	**18**

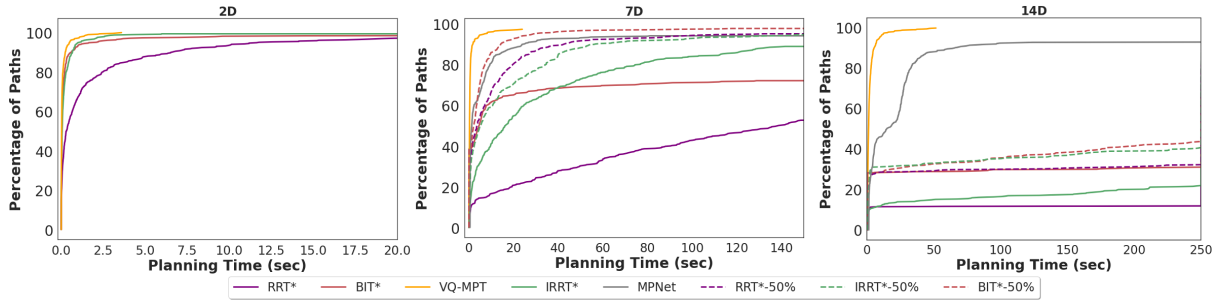

Figure 4.4. Plots of planning time and percentage of paths successfully planned on in-distribution environments for the 2D (Left), 7D (Center), and 14D (Right) robots. VQ-MPT can solve problems faster than other SMP planners by reducing the planning space and scales to higher dimensional problems.

Algorithm 3: VQMPTPlanner(q_s, q_g, $\mathcal{P}$, K, b)

$\tau \leftarrow \{q_s\}$;
for $k \leftarrow 0$ **to** K **do**
  $q_{rand} \leftarrow$ SAMPLE($\mathcal{P}$);
  $q_{near} \leftarrow$NEAREST(q_{rand}, τ);
  if *CONNECT*(q_{rand}, q_{near}) **then**
    $\tau \leftarrow \tau \cup \{q_{rand}\}$;
  if *rand()*$> b$ **then**
    $q_{gn} \leftarrow$NEAREST(q_g, τ);
    if *CONNECT*(q_{gn}, q_g) **then**
      $\tau \leftarrow \tau \cup \{q_g\}$;
      break;
  SIMPLIFY(τ);
  return τ

4.3.4 Planning

To generate a trajectory, any SMP can be used to generate the trajectory by sampling from the distribution given in Eqn. 4.9. We use Algorithm 3, to generate a path using samples from the distribution in Eqn. 4.9. The *VQMPTPlanner* function takes the start and goal state (q_s and q_g), the number of samples to generate (K), and a threshold value (b) to sample the goal state and returns a valid trajectory. This function is a modified RRT algorithm, where instead of *CONNECT* extending the current node by a small range, it checks if a valid path exists between the current and sampled node.

4.4 Experiments

We evaluated our framework on three environments - a 2D point robot, a 7D Franka Panda Arm, and a 14D Bimanual Setup. Our experiments compare the use of VQ-MPT coupled with RRT (Algorithm 3) with traditional and learning-based planners on a diverse set of planning problems. All planners were implemented using the Open Motion Planning Library (OMPL) [124].

4.4.1 Setup

We trained a separate VQ-MPT model for each robot system and chose feature extractors based on environment representations. For costmaps, we used the Fully Convolutional Network (FCN) as in [60], while for point cloud data, we used two layers of set-abstraction proposed in PointNet++ [101]. We chose these architectures because they are agnostic to the environment size and can generate latent embeddings for larger-sized costmaps or point clouds. The same transformer model architecture was used for the Stage 1 encoder, the cross-attention network, and the AR model. Each transformer model consisted of 3 attention layers with 3 attention heads each. Table 4.1 details the latent vector dimensions and the dictionary size used for each robot. A larger key size was used for the 7D and 14D robots because of the larger planning space. We observed that increasing the dictionary size further did not reduce the reconstruction loss.

All models were trained using data collected from simulation. We collected two sets of trajectories.

Trajectories without obstacles

This set consisted of trajectories in an environment without obstacles and was used to train Stage 1 of the model. These trajectories were free from any form of self-collision and covered the whole planning space of the planner. For each robot, we collected 2000 trajectories of this type.

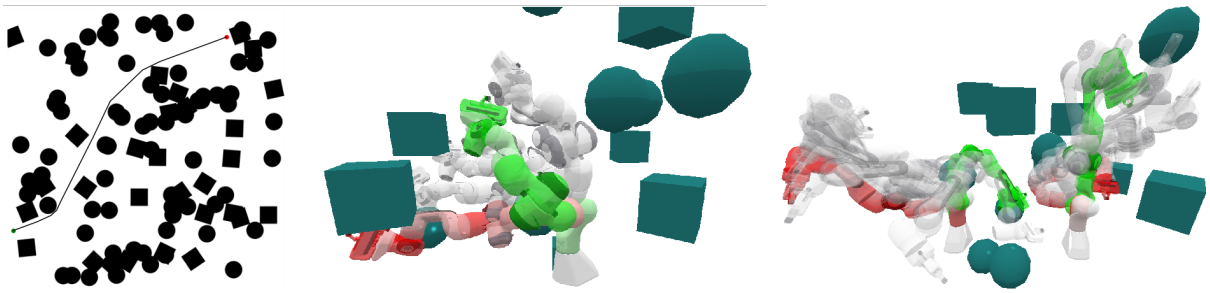

Figure 4.5. Sample paths planned by the VQ-MPT planner for different robot systems (Left) 2D robot, (Center) 7D robot, and (Right) 14D robot on in-distribution environments. The red and green color represents the start and goal states of the robot, respectively. Given an environment with crowded obstacles, VQ-MPT can sample efficiently from learned distributions to find a trajectory.

Figure 4.6. Snapshots of a trajectory planning using VQ-MPT for physical panda robot arm for a given start and goal pose on a shelf environment. On the top-right of each image, we show the point cloud data captured using Azure Kinect cameras. We used markerless camera-to-robot pose estimation to localize the captured point cloud in the robot's reference frame. VQ-MPT can generalize to real-world sensor data without additional training or fine-tuning.

Trajectories with obstacles

This set consisted of valid trajectories collected from environments where obstacles were placed randomly in the scene. It was used to train Stage 2 of the model. For each robot, we collected 10 trajectories for 2000 randomly generated environments.

We trained Stages 1 and 2 using the Adam optimizer [71] with $\beta_1 = 0.9$, $\beta = 0.98$ and $\epsilon = 10^{-9}$ and a scheduled learning rate from [134].

4.4.2 Unseen In-Distribution Environments

We compared our framework against traditional and learning-based SMP algorithms for each robot system on a trajectory from 500 different environments. To quantify

Table 4.3. Comparing accuracy and mean planning time and vertices in Out-of-Distribution Environments

Planner		Robots 7D	14D	7D (Real)
RRT*	Accuracy	8.60%	6.00%	·
	Time (sec)	107.75	4.92	·
	Vertices	1338	39	·
RRT*(50%)	Accuracy	66.60%	18.60%	·
	Time (sec)	22.75	7.61	·
	Vertices	279	67	·
IRRT*	Accuracy	44.60%	10.60%	100%
	Time (sec)	55.12	20.72	30.68
	Vertices	215	20	607
IRRT*(50%)	Accuracy	59.20%	17.80%	·
	Time (sec)	23.94	10.57	·
	Vertices	**72**	34	·
BIT*	Accuracy	37.80%	12.20%	100%
	Time (sec)	75.32	30.07	26.42
	Vertices	5147	1673	2852
BIT*(50%)	Accuracy	88.60%	30.00%	·
	Time (sec)	11.86	40.58	·
	Vertices	896	2889	·
RRT	Accuracy	84.20 %	75.00%	100%
	Time (sec)	8.88	19.75	1.69
	Vertices	477	179	21
MPNet	Accuracy	53.20%	80.40%	30%
	Time (sec)	10.14	23.91	2.23
	Vertices	310	104	7
VQ-MPT	Accuracy	**92.20%**	**98.60%**	100%
	Time (sec)	**3.24**	**6.21**	**1.17**
	Vertices	306	**70**	34

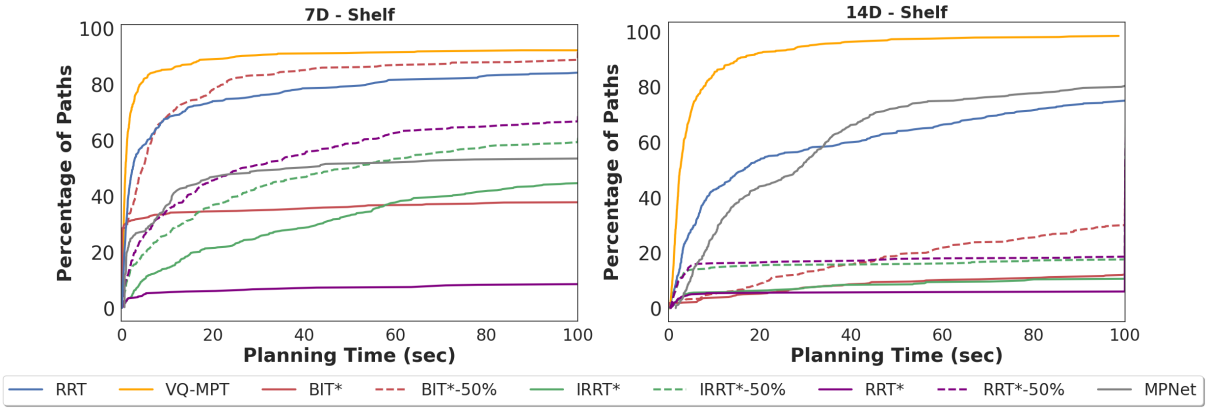

Figure 4.7. Plots of planning time and percentage of paths successfully planned for the 7D (Left) and 14D (Right) robots on environments different from ones used for training. VQ-MPT can reduce the planning space in unseen environments, enabling efficient planning in challenging environments.

planning performance, we measured three metrics: planning time - the time it takes for the planner to generate a valid trajectory; vertices - the number of collision-free vertices required to find the trajectory and accuracy - the percentage of planning problems solved before a given cutoff time. We chose to measure vertices because checking the validity of a vertex imposes a significant cost on most SMPs [27]. Since optimal planners do not have termination conditions, for fair comparisons, we stopped planning when the constructed trajectory, $\{q_1, q_2, \ldots, q_n\}$, satisfied the following condition:

$$\sum_{i=0}^{n-1} \|q_{i+1} - q_i\|_2 \leq (1+\epsilon) \sum_{j=0}^{m-1} \|q^*_{j+1} - q^*_j\|_2 \tag{4.10}$$

where $\mathcal{Q}^* = \{q^*_1, \ldots, q^*_n\}$ is the path planned by VQ-MPT and $\epsilon \geq 0$ is a user-defined threshold. If VQ-MPT could not generate a path for the trajectory, we used a path from RRT* running for 300 seconds (s) to generate $\mathcal{Q}^*$. For optimal planners like RRT*, IRRT*, and BIT*, we used $\epsilon = 0.1$ and $\epsilon = 0.5$. In our tables, planners that used $\epsilon = 0.5$ are reported by 'X (50%)', where X is the planner. The planning time reported for VQ-MPT also includes the time taken for model inference. All results are summarized in Table 4.2 and the percentage of planning problems solved vs planning time is shown in Figure 4.4.

We first tested our framework on a simple 2D robot. An example of the path planned by the VQ-MPT framework is shown in Fig. 4.5 (Left). The cutoff time set was 20 seconds. VQ-MPT showed efficient sampling of points in the planning space and found trajectories faster than traditional planners.

VQ-MPT can also use 3D environment representations such as point clouds to generate sampling regions. We evaluated the framework on a 7D panda robot arm with a point cloud environment representation. The dictionary encodings can capture diverse sets of valid configurations in 7D space (Fig. 4.2). An example of the trajectory planned by the VQ-MPT framework is shown in Fig. 4.5 (Center). The cutoff time set was 100 s. VQ-MPT planner generates a trajectory nearly 5× faster with fewer vertices than the next best accurate planner. MPNet performs poorly compared to VQ-MPT. The rigid feature encoding of MPNet potentially prevents it from generalizing to larger point cloud data environments. VQ-MPT, in contrast, learns to identify suitable regions to sample in the joint space using point cloud data of different sizes.

We also tested the framework in a bi-manual panda arm setup with 14D. An example of a VQ-MPT trajectory is shown in Fig. 4.5 (Right). Stage 1 captures the planning space with the same 2048 dictionary values used in the 7D panda experiment. The cutoff time was 250 s. While BIT* performed relatively well compared to traditional planners for the 2D and 7D problems, performance and accuracy decreased due to the high-dimensional planning space. Since Stage 1 of the VQ-MPT framework encodes self-collision-free regions, it's easier for the planner to generate feasible trajectories in Stage 2, resulting in faster trajectory generation with fewer vertices.

4.4.3 Out-of-Distribution Environments

Our next set of experiments evaluated VQ-MPT's performance for the 7D and 14D robots in environments very different from the training environments. We test our framework on different planning scenes resembling real-world scenarios (Fig. 4.1). We test

the model for each robot on 500 and 10 start and goal locations for simulation and real-world environments, respectively. The cutoff time for each planner was set at 100 s. The results of the experiments are summarized in Table 4.3, and the plot of the percentage of paths solved across planning time is given in Fig. 4.7. Higher dimensional 7D and 14D spaces are challenging. The environment is even more challenging because of the goal location inside the shelf since it reduces the number of feasible trajectories in the same way a narrow passage eliminates feasible trajectories in mobile robots [13]. Even non-optimal planners like RRT solve only 75-91% of trajectories. Existing optimal SMP planners cannot achieve the same accuracy as VQ-MPT even after relaxing path length constraints.

To evaluate the performance of VQ-MPT on physical sensor data, we tested a trained model in a real-world environment (Fig. 4.6). The environment was represented using point cloud data from Azure Kinect sensors, and collision checking was done using the octomap collision checker from Moveit [1]. Camera to robot base transform was estimated using markerless pose estimation technique [92]. Our results show that the model can plan trajectories faster than RRT with the same accuracy. We observed that VQ-MPT trajectories are also shorter than RRT trajectories, which can be clearly seen in some of the attached videos. This experiment shows that VQ-MPT models can also generalize well to physical sensor data without further training or fine-tuning. Such generalization will benefit the larger robotics community since other researchers can use trained models in diverse settings without collecting new data or fine-tuning the model.

4.5 Discussion

VQ-MPT can plan near-optimal paths in a fraction of the time required by traditional planners, scales to higher dimension planning space, and achieves better generalizability than previous learning-based planners. Our approach will be beneficial for

[1] https://moveit.ros.org/

planning multi-arm robot systems like the ABB Yumi and Intuitive's da Vinci®Surgical System. It is also helpful for applications where generating nodes and edges for SMPs is computationally expensive, such as for constrained motion planning [64]. In the next chapter, we will explore the application of VQ-MPT to constraint planning.

4.6 Acknowledgement

Chapter 4, in part, is a reprint of material from J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Learning Sampling Dictionaries for Efficient and Generalizable Robot Motion Planning with Transformers," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2023.3322087. The dissertation author is the primary author of this paper.

Chapter 5

Planning with Constraints

Constraint motion planning (CMP) is a fundamental challenge in robotics that involves finding a collision-free path between a start and goal configuration while satisfying certain constraints. Constraints may include kinematic constraints on the robot's joints [62, 64], dynamic constraints like torque limits [99], and task constraints like maintaining end-effector orientation [52]. Trajectories that adhere to these constraints are relevant in fields such as home robotics - to move containers without spilling their content, medical robotics - constrain end-effector torques to interact with humans safely [99], and industrial robotics - articulate objects such as levers and pulleys [100].

The most common algorithms for finding such trajectories are sampling-based motion planners (SMP) [79]. These algorithms build a discrete search graph of collision-free robot states by random sampling in the planning space to connect the start and goal states. They have been particularly successful at finding solutions for robotic systems with high degrees of freedom, such as manipulators [45]. Previous works have extended SMPs to plan trajectories that satisfy given constraints [56]. However, SMPs can be highly inefficient, especially for higher dimensional spaces. With the added complexity of satisfying constraints, the valid search region shrinks, exacerbating inefficiency.

In Chapter 4 we introduced, Vector Quantized-Motion Planning Transformers (VQ-MPT), which has shown promising results in improving the sampling efficiency of

SMP planners by narrowing down the search region. VQ-MPT has shown promising generalization capabilities in that trained models can reduce planning times for environments outside the training data. This chapter introduces Constraint VQ-MPT (CVQ-MPT), a fast and efficient neural planner for kinematic and task-specific constraints. Unlike previous learning-based approaches, our model requires no additional task-related data for training or finetuning the model and uses a pre-trained VQ-MPT model for planning.

5.1 Related Works

Numerous methods have been proposed to solve the CMP. Broadly, they can be categorized into optimization-based and sampling-based approaches.

Optimization-based approaches formulate the entire planning problem, including constraints, as an optimization problem. In [34], the authors apply Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [112], a method that uses covariant gradients to solve the unconstraint planning problem to construct unconstraint trajectories and project them onto the constraint manifold. TrajOpt [120] improves optimization-based planning by utilizing Sequential Convex Programming (SCP) and incorporates constraint functions as penalties within the optimization problem to solve CMP. Bonalli et al. [12] extend SCP methods by lifting the manifold constraints to Euclidean spaces. Howell et al. [50] use Augmented Lagrangian- Iterative Linear Quadratic Regulators (AL-iLQR) to build a rough solution and a projection-based method to refine the coarse solution to solve the constraint problem. However, these techniques require hand-tuning penalty functions for different tasks, often resulting in local minima solutions.

On the other hand, sampling-based approaches build discrete search graphs by random sampling in the planning space to connect the start and goal states. SMP

planners such as Rapidly Exploring Random Trees (RRT) [80] and its variants [108] have shown to be particularly effective in solving planning problems for robotic systems with higher degrees of freedom, such as manipulators and also easily adapt to a wide range of tasks [73]. Sampling-based methods can solve constraint planning by generating random samples on the constraint manifold mainly through projection-based and continuation-based methods.

The projection-based approach uses a projection operator to map sampled points onto the constraint manifold. The projection operators usually involve first-order gradients to iteratively move the sampled point toward the constraint manifold using the Jacobian of the constraint function. [9, 40] use projection-based methods for solving CMP. On the other hand, continuation-based approaches approximate constraint manifolds at local regions and use this approximation to sample points and construct local trajectories. Methods such as AtlasRRT [57] and Tangent-Bundle RRT (TB-RRT) [125, 70] simultaneously approximate the constraint manifold using tangent spaces at adhering points and uses a BiRRT to construct a tree between the start and goal points on this approximated manifold. However, the underlying optimization routines and constructing atlas's can make planning computationally expensive.

To improve the efficiency and speed of sampling-based planners, recent methods have utilized learning-based techniques to solve the planning task [53, 105]. Constraint Motion Planning Networks (CoMPNet) [110, 104] was the first of these methods that used a fully connected neural network to generate configurations near constraint surfaces. By selectively sampling points, these methods reduced planning times considerably. Similarly, in [69], a deep neural network models the constraint manifold using prior trajectory data and can compute trajectories quickly. Liu et al. [89] propose using reinforcement learning to find a policy that always satisfies the constraints, which allows the agent to explore the space efficiently and removes the need for a projection operator. Although these methods improve the efficiency and speed of constraint planning, they lack generalizability to new

environments and require access to task-specific demonstrations. In this work, we address previous CMP drawbacks without collecting any task-specific data.

5.2 Problem definition

Consider an n dimensional planning space defined by $\mathcal{C} \in \mathbb{R}^n$. The space is split into two subspaces $\mathcal{C}_{free} \subset \mathcal{C}$ and $\mathcal{C} - \mathcal{C}_{free}$ such that all states in $\mathcal{C}_{free}$ do not collide with any obstacle in the environment and are considered valid configurations. The objective of the motion planner is to generate a continuous path (or trajectory), σ, such that it connects the given start state (q_s) and a goal region ($\mathcal{C}_{goal}$), and all states in σ are in $\mathcal{C}_{free}$. For CMP, all the points in σ must also satisfy a constraint function $F : \mathcal{C} \rightarrow \mathbb{R}^k$, where k is the number of constraints. For a state $q \in \mathcal{C}_{free}$ to satisfy the constraint, $F(q) = \mathbf{0}$, where $\mathbf{0}$ is a vector of zeros. This work focuses on the constraint function defined on joint configurations, $\mathcal{C}$, and not on the robot kinematics or dynamics.

5.3 Background

5.3.1 Task-Space Regions

Introduce TSRs, their benefits, and uses. A task space for manipulators is the space spanned by possible end-effector poses. For example, the task space for the Franka Panda arm is the $SE(3)$ space. For many applications, such as moving a cup without spilling its content, opening doors for shelves, and gasping objects from a table, constraints can be defined with respect to the task space. Task-Space Regions (TSR) [9] is a common method to define such constraints. TSR represents constraints as relative pose between the robot end-effector and a target pose. The poses are represented using a homogenous

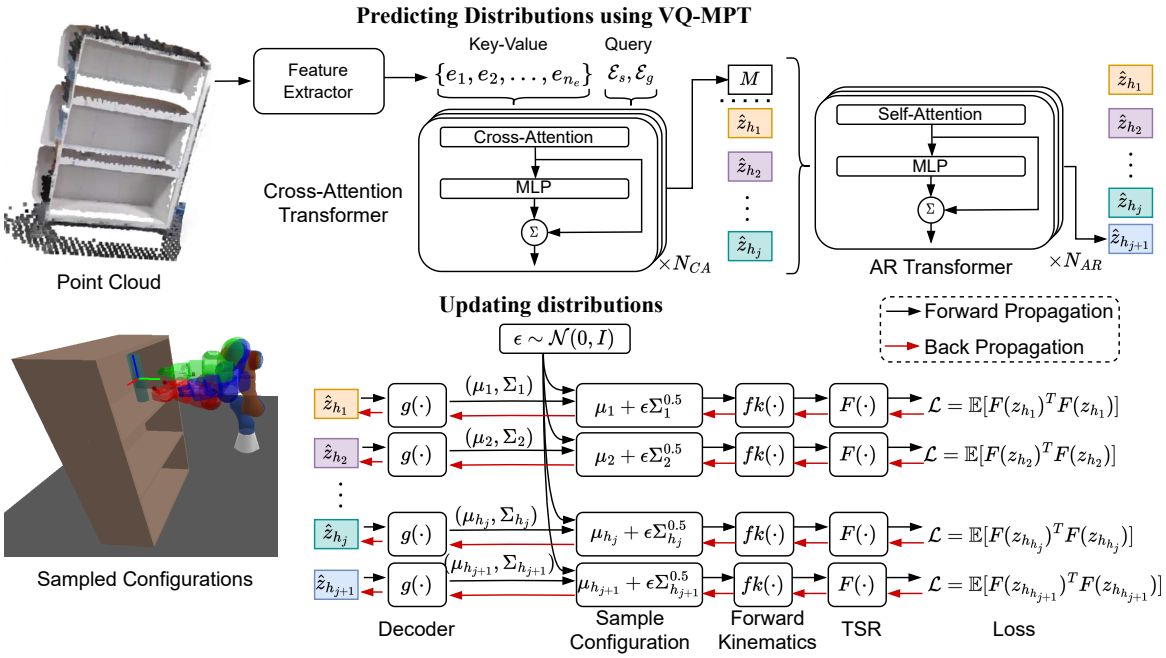

Figure 5.1. An outline of the model architecture of CVQ-MPT. Given a point cloud, and start (q_s) and goal (q_g) configurations, a pre-trained transformer model is used to generate a set of distributions parameterized using $\{\hat{z}_{h_1}, \dots \hat{z}_{j+1}\}$. The predicted distributions are updated using gradient-based optimization, minimizing the loss term $\mathcal{L}$, moving it closer to the constraint function, $F(q)$. This allows for sampling configurations closer to the constraint manifold.

transformation matrix

$$T_w^r = \begin{bmatrix} \mathbf{R}_w^r & \mathbf{p}_w^r \\ \mathbf{0} & 1 \end{bmatrix} \quad \mathbf{R}_w^r \in \mathbb{R}^{3\times 3}, \ \mathbf{p}_w^r \in \mathbb{R}^3 \tag{5.1}$$

where T_w^r represents the pose of frame r relative to frame w. Using the forward kinematics function, $fk(q) : \mathcal{C} \to SE(3)$, the relative pose of the robot end-effector, e, and a target-pose, t, in the robot's base frame, r, is given by:

$$T_e^p = T_r^p T_p^e = T_r^p (fk(q))^{-1} \tag{5.2}$$

This relative transformation can be converted to a vector in $\mathbb{R}^6$ where the first three elements are $\mathbf{p}_e^p$ and the last three are the axis-angle representation, $\boldsymbol{\omega}_e^p$, of the rotation matrix $\mathbf{R}_e^p$. Thus, the constraint function F is defined using these bounds. For example, the constraint function for moving a cup without spilling is given by:

$$F(q) = \begin{bmatrix} \mathbf{P}_e^c & 0 & 0 & \omega_z \end{bmatrix}^T \qquad \forall q \in \sigma \tag{5.3}$$

where the relative position of the cup with respect to the end-effector is fixed, and the only orientation allowed to be changed is the yaw. Additionally, in [9], the authors define TSR-chains for linking multiple TSRs and goal region TSR for defining goal region constraints.

5.4 Constraint VQ-MPT

To improve the efficiency of CMP, we sample from the distributions generated by VQ-MPT and project them onto the constraint manifold. This work uses a pre-trained VQ-MPT model trained on unconstrained shortest trajectory data collected in simulation. The following section details our sampler for constraint planning, optimization for updating predicted distributions, and planner for solving CMP.

5.4.1 Generating samples

Given a start (q_s) and goal (q_g) state, we use the trained Stage 2 model of VQ-MPT to generate a sequence of dictionary indexes $\mathcal{H} = \{h_1, \dots h_{n_h}\}$, where h_{n_h} is the goal index. A beam-search algorithm similar to those used in language model tasks [30] is used to optimize the following:

$$P(h_1, \dots, h_{n_h}|M) = \prod_{i=1}^{n_h} \pi(h_i|h_1, \dots, h_{i-1}, M) \tag{5.4}$$

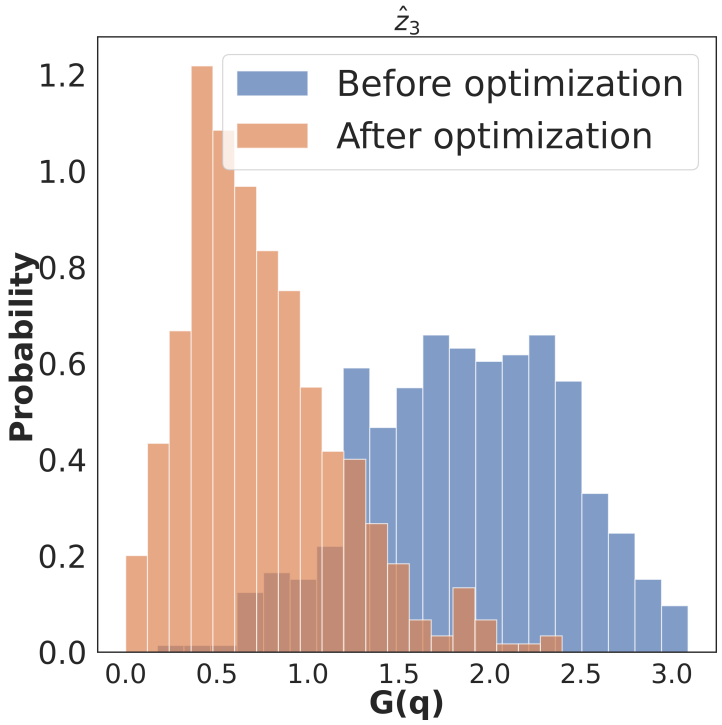

Figure 5.2. The histogram of the objective function, $G(q)$, before and after optimizing for two different dictionary values predicted by VQ-MPT for the place task. We can update the manifolds to generate samples closer to the constraint manifold and not push away distributions that are already closer.

to generate the most probable sequence, $\mathcal{H}$. Here π is the probability from Eqn. 4.7. We use a Gaussian Mixture Model (GMM) with uniform mixing coefficients reconstructed from predicted sequence $\mathcal{H}$ for sampling points. Individual Gaussian distribution is reconstructed from $\mathcal{H}$ using the decoder model from Eqn. 5.5.

$$g(\hat{z}_{h_j}) = \mathcal{N}(\mu_{h_j}, \Sigma_{h_j}) \tag{5.5}$$

$$\mathcal{P}(q) = \sum_{i=1}^{n_h - 1} \frac{1}{n_h - 1} g(\hat{z}_{h_i}) \tag{5.6}$$

To ensure that the sampled points from $\mathcal{P}(q)$ satisfy the given constraints $F(q)$, we project them on the constraint manifold using the first-order gradient projection operator given in Algorithm 4.

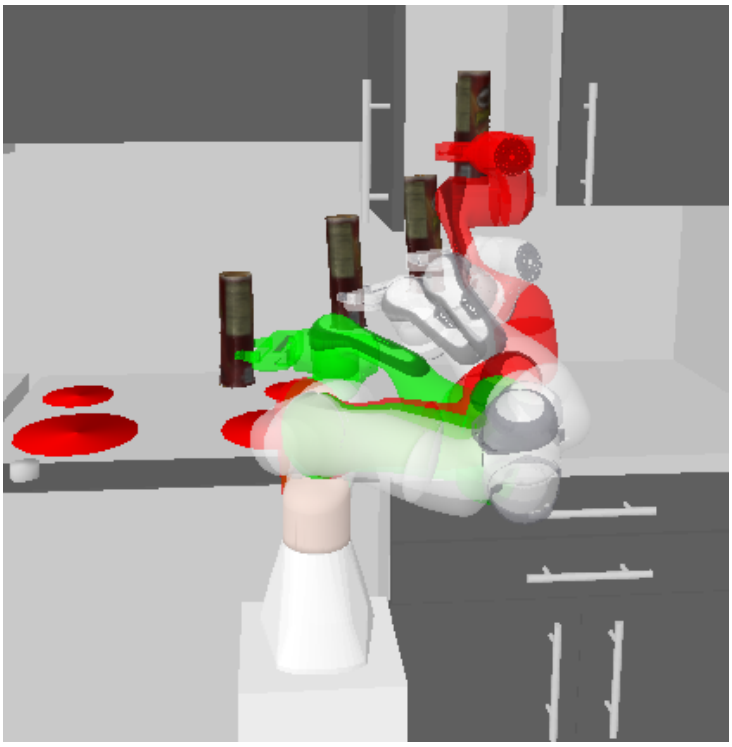

Figure 5.3. An example of the trajectory planned using CVQ-MPT for the place task for a given start (green) and goal (red) configurations. The constraint is to hold the can upright during the motion.

5.4.2 Improving sampling efficiency

The samples generated from Eqn. 4.9 are not guaranteed to lie on the constraint manifold and might require multiple optimization iterations to project them onto it. We propose a novel optimization-based update of the latent vectors in $\mathcal{H}$ such that the new distribution lies closer to the constraint, reducing projection times and consequently planning performance (Fig. 5.1). If q^* adheres to the constraint surface, it must also satisfy the following:

$$q^* = \min_q F(q)^T F(q) f \tag{5.7}$$

Thus, we can use the function $G(q) = F(q)^T F(q)$ to evaluate constraint adherence and use the same to update our latent vectors, $\hat{z}_{h_i}$. Since $G : \mathcal{C} \rightarrow \mathbb{R}^+$ is always positive, using Markov Inequality, we can upper-bound the probability of G lying outside a threshold δ

Algorithm 4: Project(q)

$x \leftarrow F(q)$;
while $\|x\| > \epsilon$ **do**
    $J \leftarrow \nabla_q F(q)$;
    $\Delta q \leftarrow -J^T(JJ^T)x$;
    $q \leftarrow q + \Delta q$;
    $x \leftarrow F(q)$;
return q

Algorithm 5: UpdateDistribution(z)

repeat
    $z_{cur} \leftarrow z$;
    $\mu, \Sigma \leftarrow g(z)$;
    $\mathcal{L} \leftarrow 0$;
    for $i \leftarrow 1$ **to** N **do**
        $\epsilon \sim \mathcal{N}(0, I)$;
        $q \leftarrow \mu + \epsilon\Sigma^{0.5}$;
        $\mathcal{L} \leftarrow \mathcal{L} + (F(q)^T F(q)/\mathrm{N})$;
    $\Delta z \leftarrow \nabla_z \mathcal{L}$;
    $z \leftarrow z_{cur} - \eta \Delta z$;
until $\|\Delta z\| < \delta$;
return z

for the distribution $g(\hat{z}_{h_i})$.

$$P(G(q) > \delta) \leq \frac{\mathbb{E}_{g(\hat{z}_{h_i})}[G(q)]}{\delta} \tag{5.8}$$

By minimizing the upper bound in Eqn. 5.8, we can implicitly reduce the samples that are further away from the constraint manifold, improving the sampling efficiency of the planner. We can use Monte Carlo estimates of $G(\cdot)$ by sampling points on the distribution $g(\hat{z}_{h_i})$ to evaluate the upper bound in Eqn. 5.8.

$$\mathcal{L} = \mathbb{E}_{g(\hat{z}_h)}[G(q)] \approx \frac{1}{n}\sum_{k=1}^{n} G(q_k) \quad q_k \sim g(\hat{z}_h) \tag{5.9}$$

To optimize Eqn. 5.9, we want to differentiate the objective function with respect to latent variable $\hat{z}_h$. We use the reparameterization trick [72] to express the random

Algorithm 6: CVQMPTBiPlanner(q_s, q_g, $\mathcal{P}$, K, b)

```
τ_s ← {q_s}, τ_g ← {q_g};
for k ← 0 to K do
    q_rand ← Sample(P);
    q^s_near ← NearestNode(q_rand, T_s) ;
    q^s_reach ← ConstrainedExtent(q_rand, q^s_near, T_s) ;
    q^g_near ← NearestNode(q_rand, T_g) ;
    q^g_reach ← ConstrainedExtent(q_rand, q^g_near, T_g) ;
    if Connect(q^s_reach, q^g_reach) then
        τ ←ExtractPath(τ_s, τ_g);
        return Simplify(τ);
    else
        Swap(τ_s, τ_g);
return Φ
```

variable q_k as a function of μ_h, Σ_h, and a normal distribution ϵ.

$$q_k = \mu_h + LD\epsilon \qquad \epsilon \sim \mathcal{N}(0, I),\ \Sigma_h = LD^2L^T \tag{5.10}$$

where L and D are a lower triangular and diagonal matrix, respectively. To generate points on $g(\hat{z}_h)$, we can sample points on $\mathcal{N}(0, I)$ and transform it using Eqn. 5.10. By substituting Eqn. 5.10 in Eqn. 5.9, the gradient of $\mathcal{L}$ becomes a function of deterministic parameters.

$$\nabla_{\hat{z}_h}\mathcal{L} \approx \frac{1}{n}\sum_{k=1}^{n} \nabla_q F(q_k)\frac{\delta(\mu_{h_j} + LD\epsilon_k)}{\delta\hat{z}_h} \tag{5.11}$$

Using the derivates from Eqn. 5.11, any optimization methods, such as Gradient Descent (GD), can be used to minimize the upper bound. Algorithm 5 provides a general outline for updating the latent variable. Fig. 5.2 shows the histogram of the objective function before and after optimization.

5.4.3 Planning

Any SMP can generate the trajectory using the sampling strategy from Section 5.4.1 and 5.4.2. We provide Algorithm 6, a bidirectional planning algorithm, to generate a path

Figure 5.4. Snapshots of the trajectory planned using CVQ-MPT for the Panda Arm completing the multi-sequence task (from left to right). CVQ-MPT can plan trajectories for various types of planning constraints in complex environments.

using samples from the distribution in Eqn. 4.9. The *CVQMPTBiPlanner* function takes the start and goal state (q_s and q_g), the GMM model ($\mathcal{P}$), the number of samples to generate (K), and a threshold value (b) to sample the goal state and returns a valid trajectory. The function *NearestNode* finds the closest node on the tree to the sampled node, while the function *ConstrainedExtend* extends the tree from the nearest node toward the sampled configuration until a collision or constraint violation occurs. For start and goal regions defined by TSR, which occur during grasping or object-placement tasks, we sample a pose from a TSR and use an Inverse Kinematics (IK) solver [6] to generate a valid collision-free configuration. This configuration is used to identify the sampling region using the VQ-MPT planner. Since the VQ-MPT planner can generate paths rapidly, multiple configurations can be evaluated successively to find a valid path.

In [9, 74], the authors have proved the probabilistic completeness of projection-based planners. They show that any RRT-based algorithm with a non-zero probability of sampling in an n-dimensional ball centered at a node of the existing tree together with the projection operator in Algorithm 4 is probabilistically complete. Since we use a set of Gaussian Distributions to sample, which spans the entire planning space, our planner also satisfies these conditions, making it probabilistically complete as well.

5.5 Experiments

This section covers our experiment setup and compares CVQ-MPT with three other planners - CBiRRT [9], Atlas-RRT [57], and TB-RRT [125] for a 7D Franka Panda arm

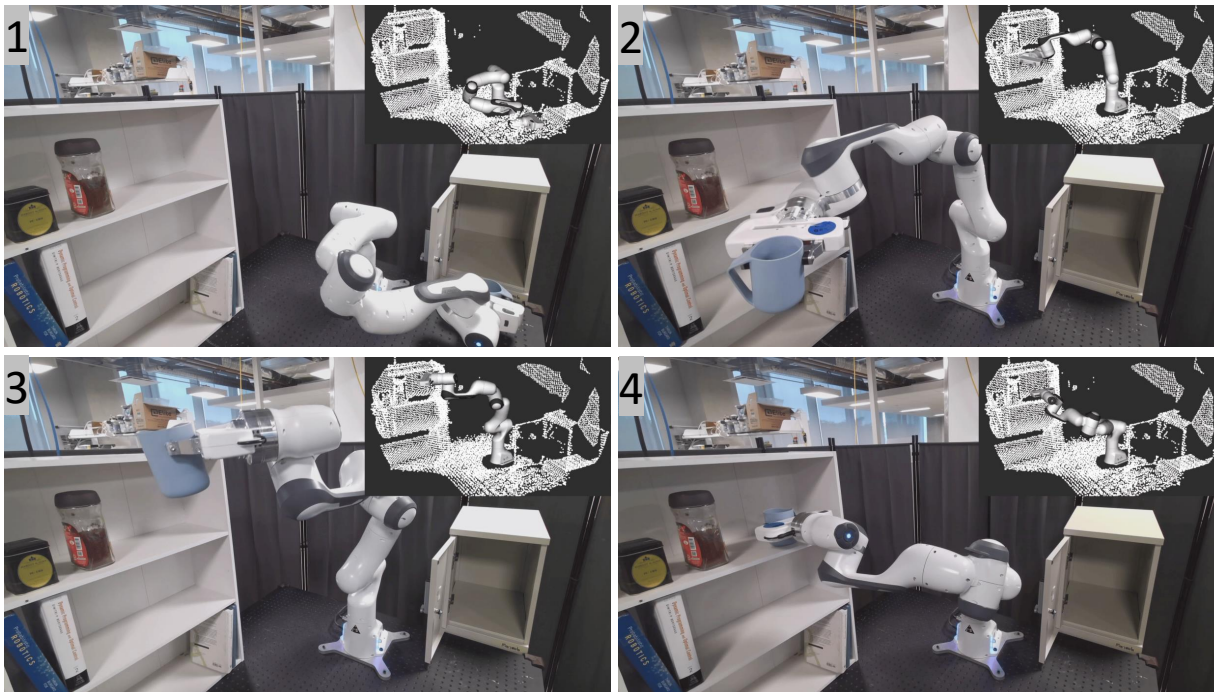

Figure 5.5. Sequences of the trajectory planned using CVQ-MPT for the Panda Arm on the physical robot and the corresponding point cloud used to represent the environment (1→2→3→4). CVQ-MPT can plan trajectories using physical sensor data and achieve the same level of performance observed in simulated environments.

on simulated and physical scenes. We also compared the performance of a pre-trained MPNet [105], which used the projection operator for steering.

5.5.1 Setup

We used pre-trained VQ-MPT and MPNet models from our prior work [63] for a 7D Franka Panda robot arm. The model was trained on simulated data using unconstrained trajectories. No constraint trajectories were used to finetune these models. All planners used in this work were implemented using the Open Motion Planning Library (OMPL) [124] on a system with an AMD Ryzen Threadripper 1950X CPU with an Nvidia RTX 3090 GPU.

Table 5.1. Place Task and Physical Robot Experiments: accuracy, planning time, vertices, & path length (l)

Environments		CBiRRT	Atlas-RRT	TB-RRT	MPNet (w/ proj)	CVQ-MPT	Opt-CVQ-MPT
Place Task	Accuracy	93.7%	100%	89.76%	33.85%	97.63%	98.42%
	Time (s)	22.95	21.80	25.31	20.61	11.25	11.03
	Vertices	40	37	32	35.98	26	24
	l	8.715	5.85	7.28	8.934	5.080	5.507
Physical robot	Accuracy	10/10	10/10	10/10	4/10	10/10	10/10
	Time (s)	8.37	11.01	13.98	7.35	9.89	8.37
	Vertices	38	28	34	5	56	41
	l	12.643	10.613	12.659	18.747	7.530	7.313

Table 5.2. Multi-sequence task: planning, executing times

	CBiRRT	Atlas-RRT	TB-RRT	CVQ-MPT	Opt-CVQ-MPT
Planning (s)	19.71	23.46	19.24	13.10	10.98
Execution (s)	22.11	20.48	22.49	20.86	18.66
Total Time (s)	41.82	43.94	41.73	33.96	29.64

5.5.2 Place Task

First, we compared our planner's performance against traditional and learning-based SMP algorithms in solving a placement task (Fig 5.3). A can is randomly placed on the kitchen counter, and the objective is to plan a path to place it on the shelf without tilting it. To quantify planning performance, we measured four metrics: planning time - the time it takes for the planner to generate a valid trajectory; vertices - the number of collision-free vertices required to find the trajectory, accuracy - the percentage of planning problems solved before a given cutoff time, and path length - the sum of all the Euclidean distance between adjacent joint states. The number of vertices will help us to determine the efficiency of different constraint planning techniques since projecting points on the constraint manifold can be computationally expensive [73]. We tested 120 different planning problems; Table 5.1 summarizes the results.

CVQ-MPT and Opt-CVQ-MPT achieve similar or better accuracy than previous constraint planners while planning shorter trajectories. CVQ-MPT produces shorter paths because the underlying VQ-MPT model was trained to identify sampling regions in

$\mathcal{C}$ where the shortest unconstrained path exists. Thus, our planner projects the unconstrained shortest path on the constraint manifold. We can also observe that by reducing the search space, CVQ-MPT can plan constraint trajectories almost 2× faster than previous planners. The MPNet model on the other hand achieves poor accuracy since the model is not generalizable to unseen environments [63].

5.5.3 Multi-Sequence Task

Our next experiment compared the planning and execution of CVQ-MPT for a sequence of tasks with varying constraints. The task involves opening the kitchen cabinet, grasping a can from the shelf, and placing it on the kitchen counter (Fig. 5.4). We compared 10 different tasks, where the robot's start position and the can's final goal position were random. Due to the poor performance of MPNet on the place task, we did not compare against it for this experiment. The average total planning and executing time results for the tasks are reported in Table 5.2. CVQ-MPT reduces total planning time by 45%, while planning a shorter path can reduce task execution times by 18%. This also shows how our planner can adapt to diverse types of task constraints.

5.5.4 Real-world environment

To evaluate the performance of our planner on physical sensor data, we tested it in a real-world environment (Fig. 5.5). The environment was represented using point cloud data from Azure Kinect sensors, and collision checking was done using the Octomap collision checker from Moveit [25]. Camera to robot base transform was estimated using marker-less pose estimation technique [92]. We tested on 10 random start and goal configurations, and the results are summarized in Table 5.1. We observe that CVQ-MPT and Opt-CV-MPT outperform traditional planners, similar to the Place Task experiment. This experiment shows that CVQ-MPT models can also generalize well to physical sensor data without further training or fine-tuning. Such generalization will benefit the larger

robotics community since other researchers can use trained models in diverse settings without collecting new data or fine-tuning the model.

5.6 Discussion

In this chapter, we introduced CVQ-MPT, a zero-shot planning framework for solving constraint motion planning problems. By reducing the search space for sampling-based planners, we improve constraint planning efficiency and speed, simultaneously generating shorter trajectories than previous planners. We further refine our search space by optimizing the predicted distributions to be closer to the constraint manifold, further improving planning performance. As we have shown, CVQ-MPT also improves task execution times and success rate which will enable future robotic systems to handle more complex tasks with intricate planning sequences. Future works can explore the application of CVQ-MPT to such task and motion planning problems.

5.7 Acknowledgement

Chapter 5, in part, is a reprint of material from J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Zero-Shot Constrained Motion Planning Transformers Using Learned Sampling Dictionaries," on arXiv preprint arXiv:2309.15272. The dissertation author is the primary author of this paper.

Chapter 6

Planning under Uncertainty

In our previous chapters, we focused on the motion planning problem for an assortment of tasks such as car navigation around obstacles [117, 61, 82] and constrained robotic manipulation [123, 103]. This work has focused on demonstrating examples where environments are highly structured, and uncertainties in sensing are overlooked. In reality, robots in the real world will face different sources of uncertainties: 1. errors in system model and sensor measurements, 2. ambiguity in the position of obstacles in the space, and 3. varying physical properties of the environment itself. Motion planning algorithms that consider the collision probability, i.e., *chance constraints* [94], perform better than previous methods in such unstructured environments [11, 26].

Previous works on planning under uncertainty using chance constraints have not guaranteed that states along a given trajectory are collision-free but rather verify that discrete states satisfy the chance constraints. Furthermore, they bound the obstacles and the robot to make the optimization tractable, making the probabilistic estimates overly conservative, leading to winding trajectories. Finally, estimating collision probabilities utilizing Monte Carlo methods is computationally expensive since the shortest distance from the robot to the obstacle, i.e., *distance-to-collision*, has to be evaluated for a large number of samples.

Ultimately, our goal is to find optimal trajectories that continuously meet chance

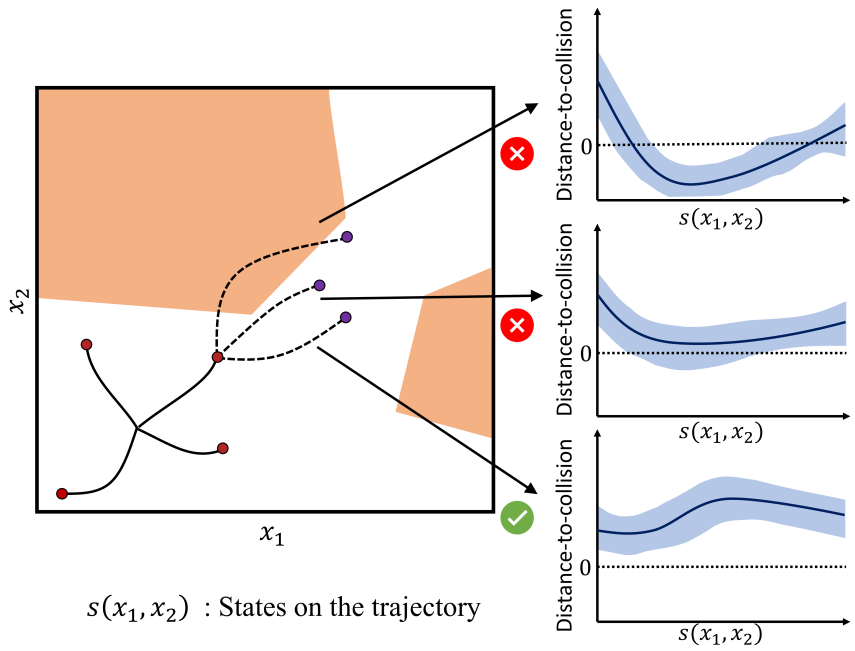

Figure 6.1. CCGP-MP is a motion planning algorithm for robotic systems under motion and sensor uncertainty which uses a Gaussian Process to model the variations in distance-to-collision. The model verifies user-defined chance constraints for trajectory segments in sampling-based planners.

constraints along an entire trajectory. To this end, we propose the Chance Constrained Gaussian Process-Motion Planning algorithm (CCGP-MP) that addresses those issues. We use a Gaussian Process (GP) to model the distribution of distance-to-collision measures for noisy robotic systems. In turn, we integrate this model with traditional sampling-based planners to generate trajectories that satisfy a given chance constraint. Our formulation ensures both the sampled states in a path satisfy the chance-constraint as well as all the points that lie along the edges connecting the sampled states.

6.1 Related Works

Many existing planning methods are based on using collision probability for planning under uncertainty [131, 11, 26, 146, 59, 119], while other solutions rely on Markov Decision Process (MDP) [16] or Partially Observable MDPs (POMDPs) [132]. MDP and POMDP often need discretization of the state space, and solving an MDP can quickly

become computationally intractable for continuous planning domains. In the following section, we will review a few of the works on estimating collision probability for planning and recent techniques used for distance estimation.

In [11], the authors find a path by formulating the planning problem as an optimization problem where the planned states have to satisfy user-defined chance-constraints while minimizing a cost. Further development of this algorithm [26] ensured that inter-node trajectories satisfied the chance-constraint but only for obstacles represented as linear functions. In [146], the authors propose tighter bounds over ellipsoidal obstacles. For these methods, the number of constraints to solve increases linearly with the number of obstacles, and for higher dimensions, the number of constraints grows exponentially. Using a GP model to capture the distance-to-collision function, we avoid the need to convexify the environment and robot, and irrespective of the environment's complexity, a single equation represents the chance constraints.

Another class of methods estimates the probability of collision along a path by obtaining the robot states' distribution and choosing one with the least likelihood of a collision. Linear-Quadratic Gaussian Motion Planning (LQG-MP) [131] method derives a distribution for the states along a path associated with using a Linear-Quadratic Gaussian (LQG) controller to stabilize the robot. Although this method can obtain a trajectory that reduces the probability of collision, as suggested in [15], there is no guarantee that LQG-MP may find a path because of the finite number of paths generated by RRT. In [98], the authors propose linear constraints on the distribution of states to obtain a tighter collision probability. [126] extends the LQG-MP for higher DOF robots, but like LQG-MP, the method does not have a user-defined chance constraint parameter. All these methods guarantee safety for discrete states but are intractable to verify safety for all points on a trajectory.

In [59, 119], the authors use Monte-Carlo simulations to get a more accurate distribution of states and a more precise collision probability estimate. In [59], the

authors use importance sampling to be more data-efficient in their simulation and reduces the variance of estimates using control variate. [119] extends this work to non-linear systems for verification of trajectory in an online planning setting. Although these methods estimate collision probability along a path precisely, they rely on meta planning algorithms to generate an initial path and, as such, cannot incorporate additional optimality criteria into the planning problem. CCGP-MP integrates with optimal planners such as RRT* to solve planning problems with optimality criteria.

Many methods are proposed that use geometric sensors and modeling techniques to estimate the distance-to-collision [19, 145], and for a brief review, readers can refer to [29], but none considers the measurement models in unstructured environments. Das and Yip [29] proposed one of the first techniques that included uncertainties to the distance measure model. The authors added Gaussian noise to the distance measure but did not consider the effect of state estimation uncertainty while modeling the distribution.

6.2 CCGP-Motion Planning

In this section, we define our problem and the assumption we make and introduce the building blocks of CCGP-MP.

6.2.1 Problem Definition

Let the state, control, and observation space be defined as $\mathcal{X} \subseteq \mathbb{R}^{n_x}$,$\mathcal{U} \subseteq \mathbb{R}^{n_u}$, and $\mathcal{Z} \subseteq \mathbb{R}^{n_z}$ respectively. For a given start position ($x_{start} \in \mathcal{X}$) and goal region ($\mathcal{X}_{goal} \subset \mathcal{X}$), the objective is to find a trajectory that satisfies a user-defined collision constraint. A trajectory Π, defined as a sequence of states $\{x_0, x_1, \ldots, x_N\}$, is considered a solution if $x_0 = x_{start}$, $x_n \in \mathcal{X}_{goal}$, and all the points that connect state x_i and x_{i+1} also satisfy the given collision chance constraint. We assume that for planning the states are sampled in a subspace $\mathbb{X} \subset \mathcal{X}$, where the system's velocities are zero. The problem can be further expanded by enforcing optimality criteria to the sequence of states, such as reducing path

length. In the subsequent sections, we detail our solution for this problem.

6.2.2 Motion and Observation Model

Throughout this paper, we will suppose that we have a nonlinear dynamics and observation model give by:

$$x_{t+1} = f(x_t, u_t) + v(m_t) \quad m_t \sim \mathcal{N}(0, M) \tag{6.1a}$$

$$z_{t+1} = h(x_t) + n_t \qquad\quad n_t \sim \mathcal{N}(0, N) \tag{6.1b}$$

where $x_t, x_{t+1} \in \mathcal{X}$, $u_t \in \mathcal{U}$, $z_t \in \mathcal{Z}$, m_t and n_t are the process noise sampled from a Gaussian distribution with variance $M \in \mathbb{R}^{n_x \times n_x}$ and $N \in \mathbb{R}^{n_z \times n_z}$ respectively, and v_t additive noise to the motion model at time t. Standard filter techniques are used to keep track of the state estimate $\hat{x}_t$ of the true state x_t. In [1], the authors show that under certain conditions, for a system given by (6.1), an LQG controller can drive the system to any point $x \in \mathbb{X}$ starting from any Gaussian distribution. The authors also show that the estimated distribution of states converges to a unique deterministic stationary covariance. We use such a controller for trajectory tracking.

6.2.3 Gaussian Process Distance Model

The shortest distance to a collision is modeled probabilistically over the entire state-space using a GP. Given a model of the environment, a GP is constructed from sampled data by randomly moving the robot model around in the environment model and searching for the distance-to-collision using a geometric method for each estimated state. In a similar fashion to [29], the distance-to-collision is evaluated using the Gilbert–Johnson–Keerthi (GJK) method, though any available geometric method can be used in practice.

Given a prior number of samples $\mathcal{N}$, the set of states $X = \{x_1, x_2, \ldots, x_{\mathcal{N}}\}$, and the

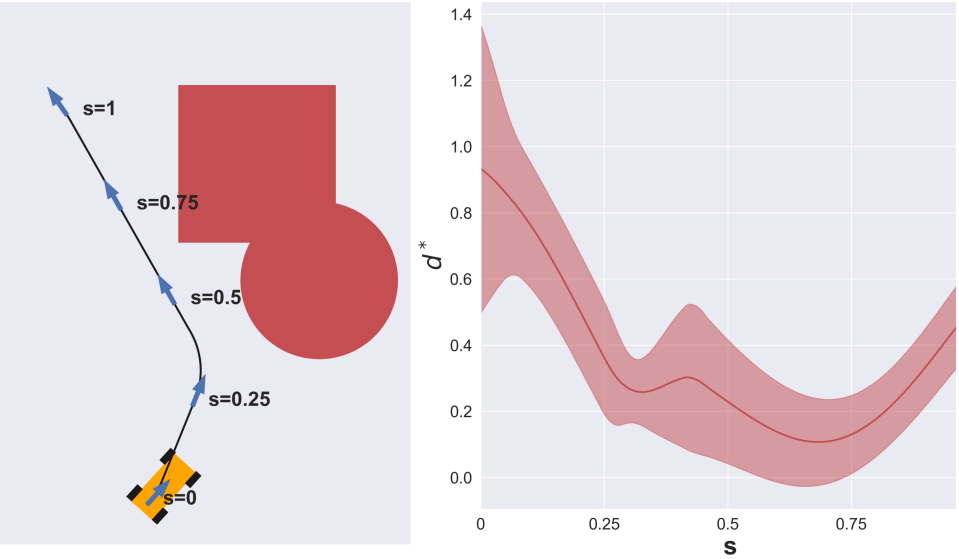

Figure 6.2. Left: An example of a trajectory in $\mathbb{X}$. It is parameterized using s where $s = 0$ and $s = 1$ represent the start and end position. Right: The corresponding mean and standard deviation of distance-to-collision (d^*), given by (6.2), for points along the trajectory (s).

corresponding distance-to-collision from the estimated states, $\mathbf{d} = \{d_1, d_2, \ldots, d_{\mathcal{N}}\}$, are used to model the GP. Note that the subscript does not represent a sequence in time, rather a random mix of samples from various trajectories. This data captures the variation in distance-to-collision measure due to uncertainty in the motion and observation model (See Fig. 6.2 for an example of a trajectory in $\mathbb{X}$ and corresponding distance-to-collision distribution). Given data points X and associated distance measure $\mathbf{d}$, for a state $\mathbf{x}^*$ the distribution of distance-to-collision, d^*, is given by:

$$d^* \mid X, \mathbf{d}, \mathbf{x}^* \sim \mathcal{N}(\mathbb{E}[d^*]), \mathbb{V}[d^*]) \tag{6.2}$$

$$\mathbb{E}[d^*] \triangleq \mathcal{K}(\mathbf{x}^*, X)[\mathcal{K}(X, X) + \sigma^2 I]^{-1}\mathbf{d} \tag{6.3}$$

$$\mathbb{V}[d^*] = \mathcal{K}(\mathbf{x}^*, \mathbf{x}^*) \tag{6.4}$$
$$-\mathcal{K}(\mathbf{x}^*, X)[\mathcal{K}(X, X) + \sigma^2 I]^{-1}\mathcal{K}(X, \mathbf{x}^*)$$

where $\mathcal{K}$ is a stationary kernel function and σ^2 represents the variance of the observed

distance measure. For the sake of brevity, we represent the vector $\mathcal{K}(\mathbf{x}^*, X)$ as $k(\mathbf{x}^*)$, the matrix $\mathcal{K}(X, X)$ as K, and the scalar $\mathcal{K}(\mathbf{x}^*, \mathbf{x}^*)$ as k^*. The kernel function $\mathcal{K}$ belongs to the class of covariance functions [111, Chapter 4] and is chosen based on the application. A Radial Basis Function (RBF) kernel is popular in most applications, though [28] demonstrates that a forward kinematics kernel provides sparser and more accurate models for robot manipulators.

6.2.4 Chance Constraints

The collision probability constraint for the state, $\mathbf{x}^* \in \mathcal{X}$, of the robot can be represented using the distance measure d^*, where

$$\mathcal{P}(\mathbf{x}^* \text{ is in collision}) \leq \delta \implies \mathcal{P}(d^* < 0) \leq \delta \tag{6.5}$$

This probabilistic constraint can be converted to a deterministic constraint as given in [11], where

$$\mathcal{P}(d^* < 0) \leq \delta \iff \frac{\mathbb{E}[d^*]}{\sqrt{2\mathbb{V}[d^*]}} \geq c \tag{6.6}$$

$$c = \text{erf}^{-1}(1 - 2\delta) \tag{6.7}$$

and where $\text{erf}(z)$ is the Gaussian error function. The ratio of mean to standard deviation in (6.6) is defined as function g given by:

$$g(\mathbf{x}^*) \triangleq \frac{k(\mathbf{x}^*)^T[K + \sigma^2 I]^{-1}\mathbf{d}}{(2(k^* - k(\mathbf{x}^*)^T[K + \sigma^2 I]^{-1}k(\mathbf{x}^*)))^{\frac{1}{2}}} \tag{6.8}$$

6.2.5 CONNECT Function

In sampling-based planners, which include popular approaches such as the many variations of Rapidly Exploring Random Trees (RRTs) and Probabilistic Roadmaps

(PRMs), there exists a function that verifies if an edge can connect two nodes by checking if the points on the edge satisfy a set of constraints. In our work, we are calling these functions `CONNECT` functions. In traditional planners, the `CONNECT` function checks for collision by subsampling the edge and evaluating if each point is collision-free. For probabilistic planners, the `CONNECT` function needs to ensure that all the points on the edge satisfy the chance constraints. To verify if an edge from state $\mathbf{x_1}$ to state $\mathbf{x_2}$ meets the given constraints, the condition defined in (6.5) must hold for all points on the edge. We can verify this by checking if the global minimum of (6.8) satisfies the constraint from (6.6) for the path segment:

$$\hat{c} \geq c, \qquad \hat{c} = \inf_{\mathbf{x}^* \in s(\mathbf{x_1}, \mathbf{x_2})} g(\mathbf{x}^*) \tag{6.9}$$

and $s(\mathbf{x_1}, \mathbf{x_2})$ represents the trajectory between $\mathbf{x_1}$ and $\mathbf{x_2}$.

6.2.6 Simplicial Homology Global Optimization

To find the global minima of the function $g(\mathbf{x}^*)$ we use the Simplicial Homology Global Optimization (SHGO) algorithm as proposed in [39]. SHGO is a global optimization technique that exploits the objective function's topography to identify sub-domains where the global minimum may lie.

The method samples the objective function by a predetermined number of samples and constructs a simplicial complex $\mathcal{H}$. The simplicial complex $\mathcal{H}$ can be conceived as a directed graph, where the vertices represent the value of the objective function at the sampled points and the directed edges point towards the vertex with a higher objective value. In [28], a vertex v_i is defined as a local minimizer if all the edges connected to v_i are directed away. A minimizer set, $\mathcal{M}$, is formed with all such local minimizers. $\text{st}(v_i)$ defines a new space called the star of a vertex v_i as the set of points Q such that every simplex containing Q contains v_i. The global minimum is found by searching through

each sub-domain, st(v_i), for all $v_i \in \mathcal{M}$. The authors prove that the cardinality of $\mathcal{M}$ remains unchanged with increasing samples, i.e., the number of regions to search for the global minimum does not change with increasing samples. The following theorem guarantees a stationary point in each of these sub-domains:

Theorem 1. *Given a minimizer $v_i \in \mathcal{M} \subseteq \mathcal{H}$ on the surface of a continuous, Lipschitz smooth objective function f with a compact bounded domain in $\mathbb{R}^n$ and range $\mathcal{R}$, there exist at least one stationary point of f within the domain defined by st(v_i) [39].*

Thus if the objective function is Lipschitz smooth and an adequate number of samples is given, SHGO is able to converge to the global minimum.

In our situation, to use SHGO to identify the global minimum, we show that (6.8) is Lipschitz smooth. (6.8) can be re-written as a composition of two functions, $g(\mathbf{x}^*) = q \circ k(\mathbf{x}^*)$. For simplicity, we assume k^* to be 1. First, we show that the function q is Lipschitz continuous.

Lemma 1. *For $\mathbf{k} \in [0,1]^n$, $K \succeq 0$ and $\sigma > 0$, the function $q(\mathbf{k})$ given by*

$$q(\mathbf{k}) = \frac{\mathbf{k}^T(\sigma^2 I + K)^{-1}\mathbf{d}}{(2(1 - \mathbf{k}^T(\sigma^2 I + K)^{-1}\mathbf{k}))^{1/2}} \tag{6.10}$$

satisfies the Lipschitz condition:

$$\|q(\mathbf{k_1}) - q(\mathbf{k_2})\| \leq L_q \|\mathbf{k_1} - \mathbf{k_2}\| \tag{6.11}$$

$$L_q = \frac{\|(\sigma^2 I + K)^{-1}\mathbf{d}\|}{\sqrt{2}} \left(\frac{1}{1 - \lambda_{max} n}\right)^{3/2} \tag{6.12}$$

where λ_{max} is the largest eigenvalue of $(I\sigma^2 + K)^{-1}$, and $\mathbf{k_1}, \mathbf{k_2} \in [0,1]^n$.

Proof. For $\mathbf{k_1}, \mathbf{k_2} \in [0,1]^n$, we can write $\|q(\mathbf{k_1}) - q(\mathbf{k_2})\|$ as:

$$\|q(\mathbf{k_1}) - q(\mathbf{k_2})\| = \left\| \frac{\mathbf{k_1}^T(\sigma^2 I + K)^{-1}\mathbf{d}}{(2(1 - \mathbf{k_1}^T(\sigma^2 I + K)^{-1}\mathbf{k_1}))^{1/2}} - \frac{\mathbf{k_2}^T(\sigma^2 I + K)^{-1}\mathbf{d}}{(2(1 - \mathbf{k_2}^T(\sigma^2 I + K)^{-1}\mathbf{k_2}))^{1/2}} \right\| \tag{6.13}$$

Let $M = (\sigma^2 I + K)^{-1}$. Since K is a Gram matrix of a covariance function, the matrix M is positive definite and symmetric [111, Chapter 4]. Hence we can simplify (6.13) as:

$$(6.13) \leq \left\| \frac{\mathbf{k_1}}{(1 - \mathbf{k_1}^T M \mathbf{k_1})^{1/2}} - \frac{\mathbf{k_2}}{(1 - \mathbf{k_2}^T M \mathbf{k_2})^{1/2}} \right\| \frac{\|M\mathbf{d}\|}{\sqrt{2}}$$

(from Cauchy-Schwarz inequality)

$$\leq \left\| \mathbf{k_1} \left(1 + \sum_{m=1}^{\infty} \frac{(\mathbf{k_1}^T M \mathbf{k_1})^m}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i \right) \right) - \mathbf{k_2} \left(1 + \sum_{m=1}^{\infty} \frac{(\mathbf{k_2}^T M \mathbf{k_2})^m}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i \right) \right) \right\| \frac{\|M\mathbf{d}\|}{\sqrt{2}}$$

(from (A.1) in Appendix)

$$\leq \left\| \mathbf{k_1} - \mathbf{k_2} + \sum_{m=1}^{\infty} \frac{(\mathbf{k_1}^T M \mathbf{k_1})^m \mathbf{k_1} - (\mathbf{k_2}^T M \mathbf{k_2})^m \mathbf{k_2}}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i \right) \right\| \frac{\|M\mathbf{d}\|}{\sqrt{2}}$$

$$\leq \left(\|\mathbf{k_1} - \mathbf{k_2}\| + \sum_{m=1}^{\infty} \frac{\| \|\mathbf{k_1}\|_M^{2m} \mathbf{k_1} - \|\mathbf{k_2}\|_M^{2m} \mathbf{k_2} \|}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i \right) \right) \frac{\|M\mathbf{d}\|}{\sqrt{2}}$$

(from triangle inequality)

$$\leq \|\mathbf{k_1} - \mathbf{k_2}\| \frac{\|M\mathbf{d}\|}{\sqrt{2}} \left(1 + \sum_{m=1}^{\infty} \frac{(1+2m)(\lambda_{max} n)^m}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i \right) \right)$$

(from Lemma 4 in Appendix)

$$
\begin{aligned}
&\leq \|\mathbf{k_1} - \mathbf{k_2}\| \frac{\|M\mathbf{d}\|}{\sqrt{2}} \\
&\quad \left(1 + \sum_{m=1}^{\infty} \frac{(\lambda_{max} n)^m}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i\right) \right. \\
&\qquad \left. +2 \sum_{m=1}^{\infty} \frac{m(\lambda_{max} n)^m}{m!} \prod_{i=0}^{m-1} \left(\frac{1}{2} + i\right)\right) \\
&\leq \|\mathbf{k_1} - \mathbf{k_2}\| \frac{\|M\mathbf{d}\|}{\sqrt{2}} \left(\frac{1}{(1 - \lambda_{max} n)^{1/2}} + \right. \\
&\qquad \left. 2 \frac{\lambda_{max} n}{2(1 - \lambda_{max} n)^{3/2}}\right)
\end{aligned}
$$

(from (A.2) and (A.3) in Appendix)

$$
\leq \|\mathbf{k_1} - \mathbf{k_2}\| \frac{\|M\mathbf{d}\|}{\sqrt{2}} \left(\frac{1}{1 - \lambda_{max} n}\right)^{3/2}
$$

□

Using Lemma 1, we can show that (6.8) is Lipschitz continuous for a Lipschitz continuous kernel function k.

Theorem 2. *For a Lipschitz continuous kernel k, (6.8) satisfies the Lipschitz condition.*

$$
\|q \circ k(\mathbf{x_1}) - q \circ k(\mathbf{x_2})\| \leq L_q L_k \|\mathbf{x_1} - \mathbf{x_2}\| \tag{6.14}
$$

where L_k is the Lipschitz constant for the kernel function.

Thus for planning, the `CONNECT` function within sampling-based planners finds the global minimum of (6.8) using SHGO and verifies that the user-defined chance constraints are satisfied for the given segment.

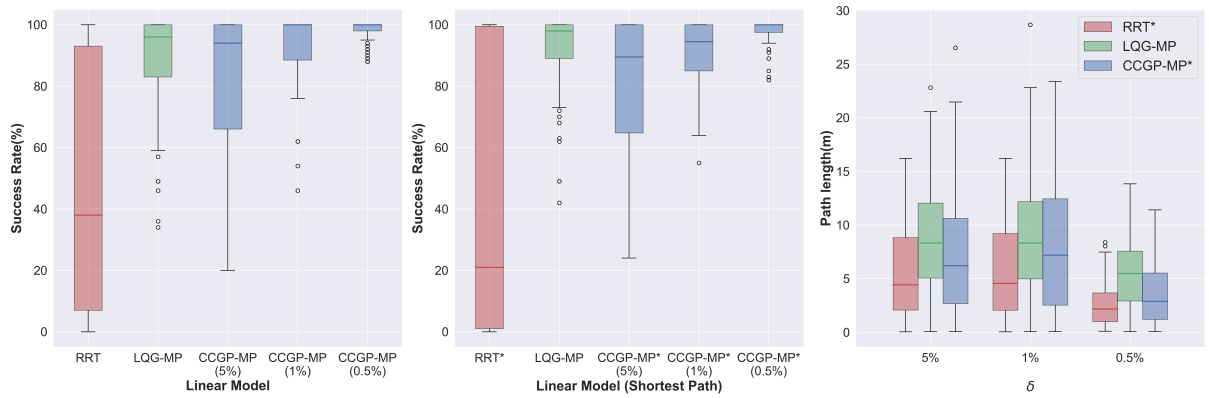

Figure 6.3. We compare the success rate and path length of the planned paths for the Linear model. Left: The quartile plot compares the success rate for a planning problem without any optimality criteria. Center: The quartile plot compares the success rate for a planning problem with added optimality criteria for reducing path length. Right: The quartile plot compares the length for the same set of start and goal pairs for different δ values. The plans generated by CCGP-MP and CCGP-MP* are robust to motion and sensor noise, and CCGP-MP* generates shorter paths than LQG-MP.

6.3 Experiments

To evaluate the CCGP-MP technique's performance, we tested it on two noisy robot models - a Linear and a Dubins Car model. We explored the planner's performance for 200 random start and goal pairs for different δ values on randomly generated environments of blocks and circles. Next, to investigate the effects of the increased number of obstacles in the environment on the planning time and accuracy, we evaluated CCGP-MP for the Linear system on 6 randomly generated environments for 10 random start and goal pairs. In addition to these simulated test environments, we assessed the Dubins Car model in a realistic indoor environment taken from the Gibson Environment suite [139].

In our experiments, we report the δ values used as percentages, since from our definition in (6.5), it represents the upper bound of the probability measure of a state in a collision. For each robot, 2000 state-distance pairs sampled randomly in the environment, and a RBF kernel are used to define the GP model. To measure the distance to a collision for sampled states, GJK was used over the map. Each planned path's performance was

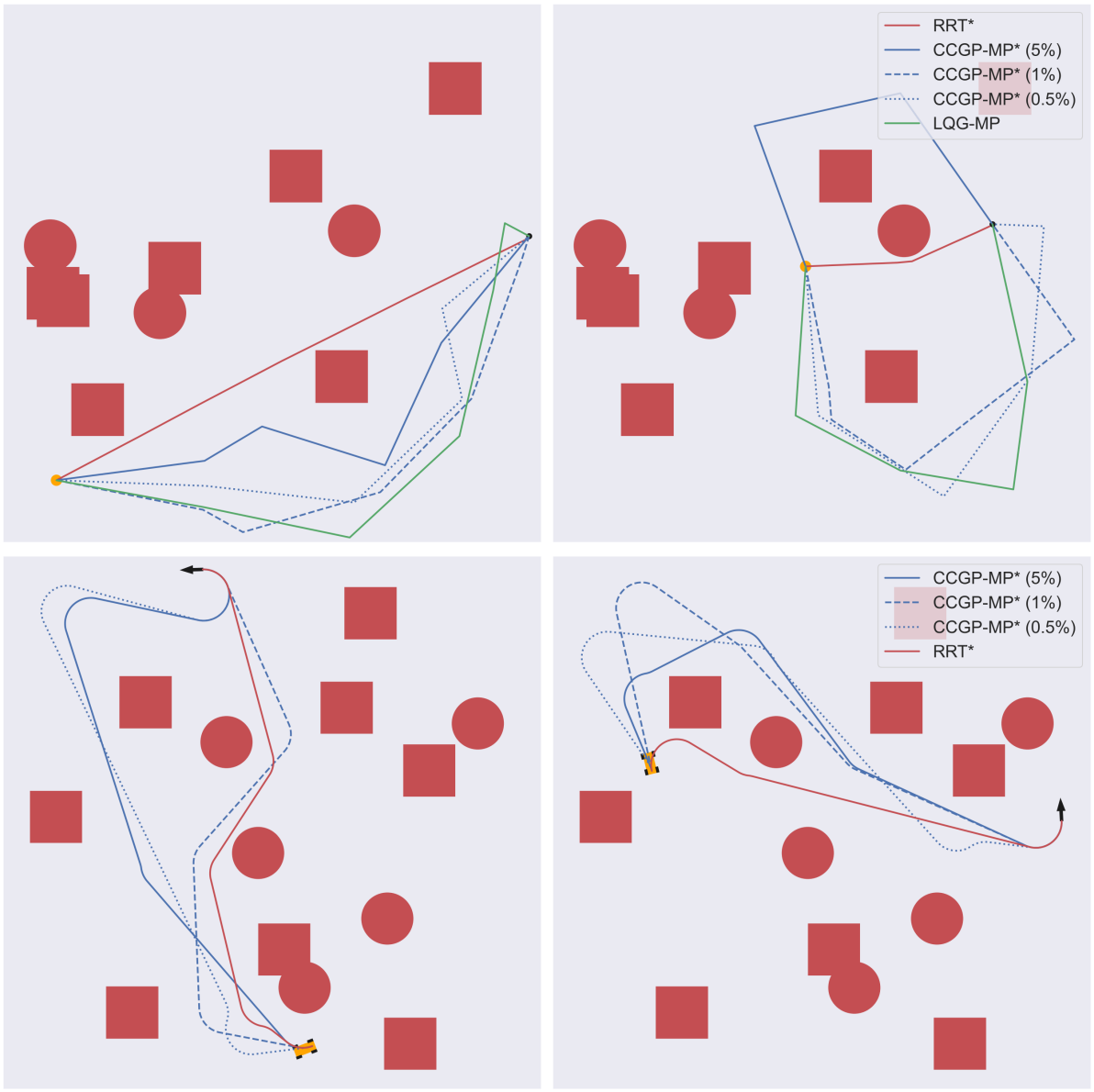

Figure 6.4. The top row shows the plans generated for different start and goal pairs for the Linear model, while the bottom row does the same for the Dubins Car model. The goals are marked using a black circle for the Linear model and a black arrow for the Dubins Car model. CCGP-MP* deviates from the RRT* planner to satisfy chance constraints.

evaluated for 100 trials using a Linear Quadratic Regulator (LQR) based trajectory following controller. A trial was concluded to be successful if the robot could reach the goal region without colliding with any obstacles. We used the Open Motion Planning Library (OMPL) [124] to implement the RRT and RRT* planners. We integrated the

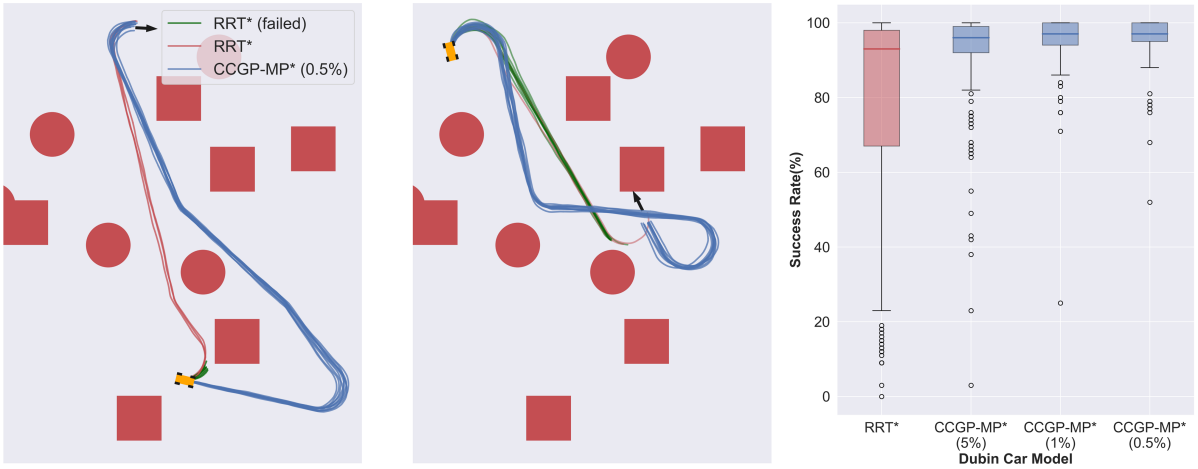

Figure 6.5. Left and Center: Two examples of path roll-outs for RRT* and CCGP-MP* for a noisy Dubins Car model. The robot collides with the obstacle while executing the path from RRT* (green), while the trajectory generated by CCGP-MP* operates without collision. Right: The quartile plot comparing the success rate of planned paths for random start and goal pairs for a noisy Dubins Car model.

`CONNECT` function for both RRT and RRT* planners and called the resulting planners CCGP-MP and CCGP-MP*, respectively. All experiments were written in the Python Programming Language and executed on an AMD Ryzen 2950x CPU with 32GB of RAM. In this section, we provide details of our experiment setup and report our results.

6.3.1 Linear Model

The first system we tested was a simple 2D model (see Fig. 6.4 top row). As the robot is symmetric about its base, we plan in the $\mathbb{R}^2$ space. The discrete-time dynamics model of the system is given by:

$$f(\mathbf{x}, \mathbf{u}) = \mathbf{x} + \mathbf{u} + m, \qquad m \sim N(0, 0.1I) \tag{6.15a}$$

$$\mathbf{z} = f(\mathbf{x}, \mathbf{u}) + n \qquad n \sim N(0, 0.01I) \tag{6.15b}$$

The LQG controller used a Kalman Filter for state estimation and an infinite horizon LQR for generating the control signal.

6.3.2 Dubins Model

We also tested CCGP-MP on a Dubins Car model whose dynamics is described by:

$$f(\mathbf{x}, u) = \mathbf{x} + \begin{bmatrix} \frac{v}{u}(-\sin(\theta) + \sin(\theta + \tau u))\tau \\ \frac{v}{u}(\cos(\theta) - \cos(\theta + \tau u))\tau \\ u\tau \end{bmatrix} \tag{6.16a}$$

where τ is the time step, and v is the linear velocity of the car. The state $\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}$ and control $u = \omega$ where (x, y) represents the position, θ the orientation, and ω the robot's angular velocity. The noisy motion model is implemented as described in [129] with linear and angular velocity noise sampled from from $\mathcal{N}(0, 0.1)$, and rotation noise sampled from $\mathcal{N}(0, 5^\circ)$. The boundary value problem of connecting the sampled state was solved using Dubins curves. An Extended Kalman Filter was used to estimate the robot state, and similar to [1], and a time-varying LQG controller was used to track the trajectory.

6.3.3 Experiment Results

We compared CCGP-MP against the RRT planner and the Linear Quadratic Gaussian-Motion Planning (LQG-MP) algorithm for the Linear system in a simulated environment. Fig. 6.3 (center) compares the planner's performance with an added optimality criteria of finding the path with the shortest length, and Fig. 6.3 (right) reports the corresponding path length. From Fig. 6.3 (left) and (center), it is evident that considering uncertainties while planning has a significant impact on the success rate of the trajectories. The variance of the success rate for CCGP-MP reduces with decreased δ. The CCGP-MP and CCGP-MP* planner have an equivalent or lower standard deviation of success-rate than LQG-MP for lower thresholds. The better performance for the CCGP methods is because they ensure all intermediate points on a path satisfy the chance-constraint while LQG-MP does not have such guarantees.

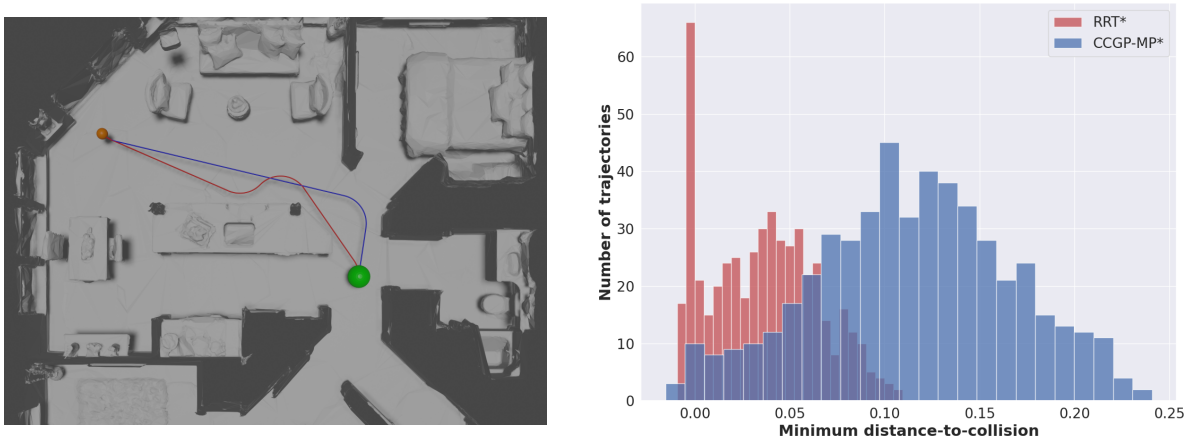

Figure 6.6. Left: The trajectories generated by RRT* (red) and CCGP-MP* (5%) (blue) for a start (orange sphere) and goal (green sphere) pair in a real-world environment. Right: The histogram comparing the minimum distance-to-collision distribution for 500 trajectories for the adjacent path. As formulated, less than 5% of the paths collide with an obstacle for CCGP-MP*.

Fig. 6.3 (right) reports the length of paths generated by the different planners for the same set of start and goal pairs. The paths generated by CCGP-MP* are shorter compared to the paths generated by LQG-MP. CCGP-MP* is able to generate optimal paths because the underlying planner of CCGP-MP* uses the RRT* algorithm, which is an asymptotically optimal planner [67] that makes no assumptions on the `CONNECT` function. Fig. 6.4, plots the different plans generated by RRT*, LQG-MP, and CCGP-MP* for a random start and goal point. From the image, we may infer that CCGP-MP* deviates from the RRT* plan where the chance constraints are not met, which results in better accuracy for these paths. In comparison, the paths from RRT*, although shorter, would result in multiple failures during execution because of the noisy robot motion. The paths from LQG-MP, on the other hand, although safer, are not optimal.

For the Dubins Car model, we compared CCGP-MP* with RRT*. We did not consider the LQG-MP algorithm for this experiment since it does not explicitly solve the shortest path problem. As expected, CCGP-MP* has a much lower standard deviation of

the success rate than RRT* (See Fig. 6.5 (right)). Fig. 6.5 reveals the reason for this improvement. The path planned by CCGP-MP* avoids the obstacles, even with the noisy robot model, while for the RRT* plan, the robot hits the obstacle. In Fig. 6.4, we compare the paths generated by CCGP-MP* and RRT*, and like the Linear model, the CCGP-MP* deviates from the RRT* plan when chance-constraints are not met.

The results of the study on environment density on planning time and accuracy are summarized in Table 6.1. The start and goal pairs were sampled from independent normal distributions with fixed means and 0.5 standard deviations. This distribution was fixed for all the environments. Apart from planning time, we also recorded the time taken by SHGO to find the global minimum for 50 random path segments. The GP model used for each environment had an equal number of support points, resulting in almost similar edge evaluation times. The overall planning time increases with the number of obstacles in space since the planner has to search more to find a feasible solution. We also observed that path accuracy was inversely proportional to the number of objects. The drop in accuracy could be attributed to the fact that more path segments are closer to the set threshold with a denser environment, thus decreasing the overall accuracy of the path.

Table 6.1. Study of number of obstacles on planning performance

Number of Obstacles	Edge Evaluation Time (sec)	Planning Time Time (min)	Median Accuracy (%)
10	0.397 ± 0.058	2.38 ± 1.27	81.0
14	0.394 ± 0.043	1.87 ± 1.13	74.5
18	0.393 ± 0.061	2.39 ± 1.18	72.0
22	0.385 ± 0.058	3.28 ± 2.07	43.0
26	0.401 ± 0.061	4.70 ± 2.34	59.5
30	0.406 ± 0.052	4.36 ± 2.59	67.0

In addition to the simulated environment, we tested our planner on an indoor environment from the Gibson suite [139]. Fig. 6.6 (left) shows the trajectory generated by the RRT* and CCGP-MP* (5%) for a single start and goal pair. Fig. 6.6 (right) shows

the minimum distance to collision for each trajectory evaluated across 500 runs. For RRT, 16.4% of the trajectories have the distance-to-collision less than zero, while for CCGP-MP, only 2.4% of the trajectories have distance-to-collision less than zero. The value for CCGP-MP* also satisfies the delta threshold set for the planner, which is 0.05.

6.4 Discussion

In this chapter, we introduced the Chance Constrained Gaussian Process Motion Planning, a chance-constrained motion planning approach that uses modeled distance-to-collision functions to plan in unstructured environments. Through the modeled distribution function, the planner guarantees that all states along the trajectory satisfy the given chance constraints. Simulation results on two robot systems showed that CCGP-MP and CCGP-MP* were able to generate paths that improved the planned path's success rate.

One of the few limitations of our work lies in approximating distance-to-collision distribution as a Gaussian distribution. This simplification may not apply to some robotic systems. Another limitation centers around the scaling up of planning space. For larger maps, we require more support points to model our GP. For a large number of points(>10000), the kernel matrix requires a considerable amount of memory [111, Chapter 8], and the linear equations that need solving for inference becomes computationally intensive. One way to overcome these limitations is to use sparse GP models. One could even construct local GP models for larger maps similar to [136].

6.5 Acknowledgement

Chapter 6, in part, is a reprint of material from J. J. Johnson and M. C. Yip, "Chance-Constrained Motion Planning using Modeled Distance- to-Collision Functions," 2021 IEEE 17th International Conference on Automation Science and Engineering

(CASE) pp. 1582-1589, doi:10.1109/CASE49439.2021.9551655. The dissertation author is the primary author of this paper.

Chapter 7

Conclusion

In this thesis, we have demonstrated the remarkable potential of transformers in revolutionizing motion planning. Unlike traditional methods that often require customizing models for specific environments, our novel approach, as presented in this thesis, offers unrivaled adaptability. We have shown that our methods can seamlessly extend to a wide array of diverse environments without the need for time-consuming fine-tuning or retraining. This exceptional generalizability simplifies the implementation process and opens up new horizons in tackling intricate challenges, such as constraint planning. By leveraging the power of transformers, we have significantly enhanced the versatility and efficiency of motion planning, paving the way for more effective and adaptable solutions to complex real-world problems.

Using VQ-MPT, we demonstrated the adaptability of the transformer model, showcasing its capabilities to plan for robots with both 7D and 14D configurations effectively. However, the extension to even larger planning spaces, such as a 36D humanoid robot, presents new challenges, particularly in determining the optimal number of dictionary values for effective quantization. Furthermore, we encountered limitations in cluttered scenes where the VQ-MPT model struggles to represent the environment succinctly. To address these issues, future research endeavors could delve into a more theoretical approach for defining the number of dictionary vectors and explore the

application of sparse transformers [116, 8] to encode complex and cluttered environments, thus enhancing the versatility and robustness of our planning system.

Our planners are designed with the assumption of perfect environmental information. However, due to sensor and environment uncertainties, a planned path may still encounter collisions due to discrepancies between the anticipated obstacle positions and the robot's actual path. Although CCGP-MP deals with such uncertainties for $SE(2)$ robots, for robots with higher planning spaces, the model requires a more significant number of support points. With support points exceeding 10,000, the kernel matrix consumes a significant amount of memory, as highlighted in [111, Chapter 8], and solving the linear equations for inference becomes computationally expensive. To mitigate these challenges, one promising approach is the utilization of sparse GP models. It's even possible to construct local GP models for larger maps, akin to the methodology proposed in [136].

In motion planning, our approach can be applied in several exciting avenues for future exploration and innovation. One promising direction involves extending the transformer models to motion planning of systems with dynamics, offering the potential to enhance the efficiency and adaptability of such systems. Using the quantized model, we can reduce the search space by learning reachable sets and identifying self-collision-free states. This development holds the promise of significantly expediting the planning process. The transformer-based approach could also be harnessed for more intricate systems, such as multi-robot arm configurations like the ABB Yumi and Intuitive's da Vinci®Surgical System, offering a versatile approach to solving complex, real-world challenges in robotics and automation.

An intriguing avenue for applying this approach is in the domain of task and motion planning (TAMP). Within TAMP, a critical element in any task is the construction of individual trajectories. If we consider robot configurations as "words" and trajectories as "sentences," we can liken solving TAMP problems to generating coherent paragraphs (Fig. 7.1. This thesis lays the foundation for learning such models. Consequently, there is

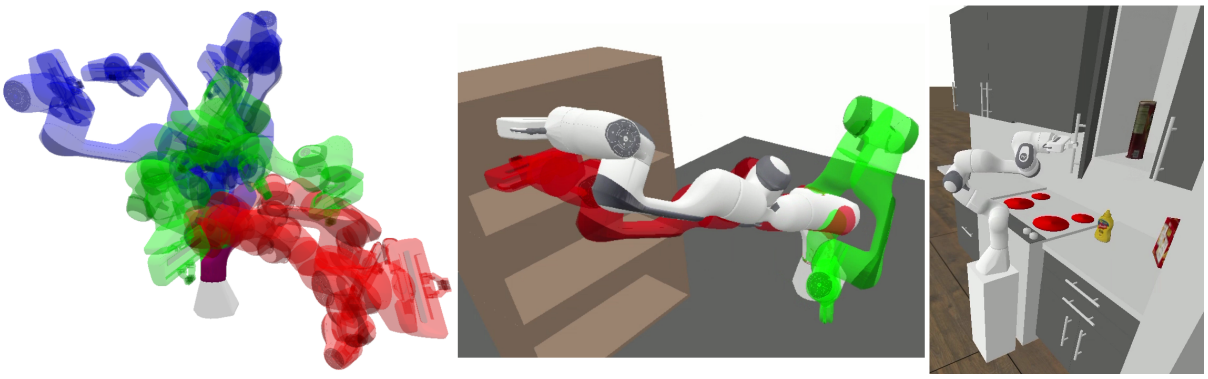

Figure 7.1. In this thesis, we have defined a set of words using vector quantization (Left) and constructed trajectories using this sequence of words (Center). Future works could look at how these sentences can be stitched together to complete complex TAMP problems (Right).

substantial room for further exploration and development to investigate how this approach can be effectively harnessed for TAMP problem-solving. Moreover, with the ongoing advancements in semantic models that exhibit a superior understanding of complex scenes [75, 77], integrating such models with our planner holds the potential to enhance the capabilities of TAMP systems further.

Appendix A

Additional Proofs for Chance Constraint Planning

For $x \in [0,1]$, the Taylor series expansion of $\frac{1}{(1-ax)^{\frac{1}{2}}}$ about 0 is given by:

$$\frac{1}{(1-ax)^{\frac{1}{2}}} = 1 + \sum_{m=1}^{\infty} \frac{(ax)^m}{m!} \prod_{j=0}^{m-1} (\frac{1}{2} + j) \tag{A.1}$$

Using simple calculus, we can show that:

$$1 + \sum_{m=1}^{\infty} \frac{a^m}{m!} \prod_{j=0}^{m-1} (\frac{1}{2} + j) = \frac{1}{(1-a)^{\frac{1}{2}}} \tag{A.2}$$

$$\sum_{m=1}^{\infty} \frac{ma^m}{m!} \prod_{j=0}^{m-1} (\frac{1}{2} + j) = \frac{a}{2(1-a)^{\frac{3}{2}}} \tag{A.3}$$

Lemma 2. *For $\mathbf{x_1}, \mathbf{x_2} \in [0,1]^n$ and a positive definite symmetric matrix M we have ,*

$$|\mathbf{x_1}^T M \mathbf{x_1} - \mathbf{x_2}^T M \mathbf{x_2}| \leq 2\lambda_{max}\sqrt{n}\|\mathbf{x_1} - \mathbf{x_2}\| \tag{A.4}$$

where λ_{max} is the maximum eigenvalue of M.

Proof. Since M is symmetric, we can expand (A.4) as

$$\begin{aligned}|\mathbf{x_1}^T M\mathbf{x_1} - \mathbf{x_2}^T M\mathbf{x_2}| &= |(\mathbf{x_1} - \mathbf{x_2})^T M(\mathbf{x_1} + \mathbf{x_2})| \\ &\leq \lambda_{max}\|\mathbf{x_1} - \mathbf{x_2}\|\|\mathbf{x_1} + \mathbf{x_2}\| \\ &\leq 2\lambda_{max}\|\mathbf{x_1} - \mathbf{x_2}\|\sqrt{n}\end{aligned}$$

□

Lemma 3. *For $\mathbf{x_1}, \mathbf{x_2} \in [0,1]^n$, a symmetric positive definite matrix M and $m \in \mathbb{N}$ we have,*

$$|(\mathbf{x_1}^T M\mathbf{x_1})^m - (\mathbf{x_2}^T M\mathbf{x_2})^m| \leq \frac{2}{\sqrt{n}} m(n\lambda_{max})^m\|\mathbf{x_1} - \mathbf{x_2}\| \tag{A.5}$$

where λ_{max} is the maximum eigen value of M.

Proof. The polynomial equation $x^m - y^m$ can be expressed as follows:

$$x^m - y^m = (x - y)(x^{m-1} + x^{m-2}y + \ldots + y^{m-1}) \tag{A.6}$$

Let $\mathbf{x_i}^T M\mathbf{x_i} = \|\mathbf{x_i}\|_M^2$ for $i \in \{0, 1\}$ we can express (A.5) in the same fashion as above,

$$\begin{aligned}|\|\mathbf{x_1}\|_M^{2^m} - \|\mathbf{x_2}\|_M^{2^m}| = |(\|\mathbf{x_1}\|_M^2 - \|\mathbf{x_2}\|_M^2)(\|\mathbf{x_1}\|_M^{2^{(m-1)}} \\ +\|\mathbf{x_1}\|_M^{2^{(m-2)}}\|\mathbf{x_2}\|_M^2 + \ldots + \|\mathbf{x_2}\|_M^{2^{(m-1)}})|\end{aligned} \tag{A.7}$$

Since $\mathbf{x_i} \in [0,1]^n$ we can write $\|\mathbf{x_i}\|_M^2 \leq \lambda_{max} n$ for $i \in \{0, 1\}$, where λ_{max} is the largest eigen value of M. Using this bound we can simplify (A.7) as follows:

$$\begin{aligned}(A.7) &\leq |\|\mathbf{x_1}\|_M^2 - \|\mathbf{x_2}\|_M^2| m(\lambda_{max} n)^{m-1} \\ &\leq 2\sqrt{n}\lambda_{max}\|\mathbf{x_1} - \mathbf{x_2}\| m(\lambda_{max} n)^{m-1}\end{aligned}$$

(from Lemma 2)

$$\leq \frac{2}{\sqrt{n}} m(\lambda_{max} n)^m \|\mathbf{x_1} - \mathbf{x_2}\|$$

□

Lemma 4. *For $\mathbf{x_1}, \mathbf{x_2} \in [0,1]^n$, a symmetric positive definite matrix M and $m \in \mathbb{N}$ we have:*

$$\|(\mathbf{x_1}^T M \mathbf{x_1})^m \mathbf{x_1} - (\mathbf{x_2}^T M \mathbf{x_2})^m \mathbf{x_2}\| \leq \tag{A.8}$$
$$(1+2m)(\lambda_{max} n)^m \|\mathbf{x_1} - \mathbf{x_2}\|$$

where λ_{max} is the largest eigenvalue of M.

Proof. We can simplify (A.8) using Lemma 3.

$$\begin{aligned}
&\| \, \|\mathbf{x_1}\|_M^{2m} \mathbf{x_1} - \|\mathbf{x_2}\|_M^{2m} \mathbf{x_2} \, \| = \\
&\quad \| \, \|\mathbf{x_1}\|_M^{2m} (\mathbf{x_1} - \mathbf{x_2}) + (\|\mathbf{x_1}\|_M^{2m} - \|\mathbf{x_2}\|_M^{2m}) \mathbf{x_2} \, \| \\
&\leq \| \, \|\mathbf{x_1}\|_M^{2m} (\mathbf{x_1} - \mathbf{x_2}) \, \| + \| \, (\|\mathbf{x_1}\|_M^{2m} - \|\mathbf{x_2}\|_M^{2m}) \mathbf{x_2} \, \| \\
&\leq (\lambda_{max} n)^m \|\mathbf{x_1} - \mathbf{x_2}\| + \sqrt{n} \frac{2}{\sqrt{n}} m (\lambda_{max} n)^m \|\mathbf{x_1} - \mathbf{x_2}\| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(from Lemma 3)} \\
&\leq (1+2m)(\lambda_{max} n)^m \|\mathbf{x_1} - \mathbf{x_2}\|
\end{aligned}$$

□

Bibliography

[1] Ali Agha-mohammadi, Suman Chakravorty, and Nancy M Amato. Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *Int. J. of Robot. Res.*, 33(2):268–304, 2014.

[2] Oktay Arslan, Karl Berntorp, and Panagiotis Tsiotras. Sampling-based algorithms for optimal motion planning using closed-loop prediction. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4991–4996, 2017.

[3] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, and Dharshan Kumaran. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557, 05 2018.

[4] Holger Banzhaf, Paul Sanzenbacher, Ulrich Baumann, and J. Marius Zöllner. Learning to predict ego-vehicle poses for sampling-based nonholonomic motion planning. *CoRR*, abs/1812.01127, 2018.

[5] Ulrich Baumann, Claudius Guiser, Serge Herman, and Johann Marius Zöllner. Predicting ego-vehicle paths from environmental observations with a deep neural network. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9, 2018.

[6] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935, 2015.

[7] Bradley M. Bell and James V. Burke. Algorithmic differentiation of implicit functions and optimal values. In Christian H. Bischof, H. Martin Bücker, Paul Hovland, Uwe Naumann, and Jean Utke, editors, *Advances in Automatic Differentiation*, pages 67–77, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[9] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[10] John T. Betts, Samuel K. Eldersveld, Paul D. Frank, and John G. Lewis. An interior-point algorithm for large scale optimization. In Lorenz T. Biegler, Matthias Heinkenschloss, Omar Ghattas, and Bart van Bloemen Waanders, editors, *Large-Scale PDE-Constrained Optimization*, pages 184–198, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[11] L. Blackmore, M. Ono, and B. C. Williams. Chance-constrained optimal path planning with obstacles. *IEEE Trans. on Robot.*, 27(6):1080–1094, 2011.

[12] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, Thomas Lew, and Marco Pavone. Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019.

[13] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Trans. on Robotics and Auto.*, 1991.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[15] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE Int. Conf. Robot. and Automation*, pages 723–730, 2011.

[16] W. Burgard, O. Brock, and C. Stachniss. *The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty*, pages 233–240. The MIT Press, 2008.

[17] Constantinos Chamzas, Zachary Kingston, Carlos Quintero-Peña, Anshumali Shrivastava, and Lydia E. Kavraki. Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions. In *IEEE Int. Conf. on Robot. and Autom.*, 2021.

[18] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of

Proceedings of Machine Learning Research, pages 1484–1495. PMLR, 18–24 Jul 2021.

[19] J. Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust. Neural collision clearance estimator for batched motion planning. In *Algorithmic Found. of Robot. XIV*, pages 73–89, 2021.

[20] Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *Int. Conf. on Learning Representations, ICLR*, 2020.

[21] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

[22] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016.

[23] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end to end. *CoRR, abs/1809.10124*, 2018.

[24] Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14):1632–1672, 2018.

[25] D.M. Coleman, Ioan Alexandru Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *ArXiv*, abs/1404.3785, 2014.

[26] M. d. S. Arantes, C. F. M. Toledo, B. C. Williams, and M. Ono. Collision-free encoding for chance-constrained nonconvex path planning. *IEEE Trans. on Robot.*, 35(2):433–448, 2019.

[27] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Trans. on Robotics*, 2020.

[28] Nikhil Das and Michael C. Yip. Forward kinematics kernel for improved proxy collision checking. *IEEE Robot. and Automat. Lett.*, 5(2):2349–2356, 2020.

[29] Nikhil Das and Michael C. Yip. Stochastic modeling of distance to collision for robot manipulators. *IEEE Robot. and Automat. Lett.*, 6(1):207–214, 2021.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[31] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[32] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. on Learning Representations*, 2021.

[34] Anca D. Dragan, Nathan D. Ratliff, and Siddhartha S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 4582–4588, 2011.

[35] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 1957.

[36] Lester E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497, 1957.

[37] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in Neural Information Processing Systems*, 2000.

[38] Scott Emmons, Ajay Jain, Misha Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5251–5262. Curran Associates, Inc., 2020.

[39] Stefan C. Endres, Carl Sandrock, and Walter W. Focke. A simplicial homology algorithm for lipschitz optimisation. *J. of Global Optim.*, 72(2):181–217, Oct 2018.

[40] Peter Englert, Isabel Rayas Ferná ndez, Ragesh Ramachandran, and Gaurav Sukhatme. Sampling-based motion planning on sequenced manifolds. In *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, jul 2021.

[41] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by

combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[42] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *6th Annual Conference on Robot Learning*, 2022.

[43] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[44] 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 *Int. Conf. on Intelligent Robots and Systems*, 2014.

[45] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE Int. Conf. Robot. Autom.*, 2015.

[46] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1243–1252. JMLR.org, 2017.

[47] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[48] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control*, pages 43–66. Springer London, London, 2011.

[49] Peter E. Hart, Nils J. 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.

[50] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019.

[51] Humphrey Hu and George Kantor. Parametric covariance prediction for heteroscedastic noise. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[52] Jiaming Hu, Akanimoh Adeleye, and Henrik I. Christensen. Place-and-pick-based re-grasping using unstable placement. In *Robotics Research*, 2023.

[53] Chia-Man Hung, Shaohong Zhong, Walter Goodwin, Oiwi Parker Jones, Martin Engelcke, Ioannis Havoutis, and Ingmar Posner. Reaching through latent space: From joint statistics to path planning in manipulation. *IEEE Robotics and Automation Letters*, 7(2):5334–5341, 2022.

[54] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[55] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Auto. Letters*, 2019.

[56] L. Jaillet and J.M. Porta. Efficient asymptotically-optimal path planning on manifolds. *Robotics and Autonomous Systems*, 2013.

[57] Léonard Jaillet and Josep M. Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Transactions on Robotics*, 29(1):105–117, 2013.

[58] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[59] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. In *Robot. Res.: Volume 2*, pages 343–361, 2018.

[60] Jacob J. Johnson, Uday S. Kalra, Ankit Bhatia, Linjun Li, Ahmed H. Qureshi, and Michael C. Yip. Motion planning transformers: A motion planning framework for mobile robots, 2021.

[61] Jacob J. Johnson, Linjun Li, Fei Liu, Ahmed H. Qureshi, and Michael C. Yip. Dynamically constrained motion planning networks for non-holonomic robots. In *2020 IEEE/RSJ Int. Conf. Intell. Robots and Systems*, pages 6937–6943, 2020.

[62] Jacob J. Johnson, Linjun Li, Fei Liu, Ahmed H. Qureshi, and Michael C. Yip. Dynamically constrained motion planning networks for non-holonomic robots. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[63] Jacob J Johnson, Ahmed H Qureshi, and Michael Yip. Learning sampling dictionaries for efficient and generalizable robot motion planning with transformers, 2023.

[64] Jacob J. Johnson and Michael C. Yip. Chance-constrained motion planning using modeled distance- to-collision functions. In *Int. Conf. on Auto. Science and Engineering (CASE)*, 2021.

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

[66] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[67] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotic Res.*, 30:846–894, 06 2011.

[68] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Auto.*, 1996.

[69] Piotr Kicki, Puze Liu, Davide Tateo, Haitham Bou-Ammar, Krzysztof Walas, Piotr Skrzypczyński, and Jan Peters. Fast kinodynamic planning on the constraint manifold with deep neural networks, 2023.

[70] Beobkyoon Kim, Terry Taewoong Um, Chansu Suh, and F. C. Park. Tangent bundle rrt: A randomized algorithm for constrained motion planning. *Robotica*, 34(1):202–225, 2016.

[71] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. on Learning Representations*, 2015.

[72] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[73] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):159–185, 2018.

[74] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, 38(10-11):1151–1178, 2019.

[75] Lingdong Kong, You-Chen Liu, Runnan Chen, Yuexin Ma, Xinge Zhu, Yikang Li, Yuenan Hou, Y. Qiao, and Ziwei Liu. Rethinking range view representation for lidar segmentation. *ArXiv*, abs/2303.05367, 2023.

[76] Rahul Kumar, Aditya Mandalika, Sanjiban Choudhury, and Siddhartha Srinivasa. Lego: Leveraging experience in roadmap generation for sampling-based planning. In *Int. Conf. on Intelligent Robots and Systems*, 2019.

[77] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical transformer for lidar-based 3d recognition, 2023.

[78] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[79] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[80] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 2001.

[81] Peter Lehner and Alin Albu-Schäffer. The repetition roadmap for repetitive constrained motion planning. *IEEE Robot. and Autom. Letters*, 2018.

[82] Linjun Li, Yinglong Miao, Ahmed H. Qureshi, and Michael C. Yip. MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints, 2021. *arXiv:2101.06798*.

[83] Linjun Li, Yinglong Miao, Ahmed H Qureshi, and Michael C Yip. Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. *IEEE Robotics and Automation Letters*, 6(3):4496–4503, 2021.

[84] Y. Li, F. Richter, J. Lu, E. K. Funk, R. K. Orosco, J. Zhu, and M. C. Yip. SuPer: A surgical perception framework for endoscopic tissue manipulation with surgical robotics. *IEEE Robot. and Automat. Lett.*, 5(2):2294–2301, 2020.

[85] Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[86] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[87] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *Int. Conf. on Learning Representations*, 2017.

[88] Katherine Liu, Kyel Ok, William Vega-Brown, and Nicholas Roy. Deep inference for covariance estimation: Learning gaussian noise models for state estimation. In *2018 IEEE Int. Conf. on Robotics and Auto. (ICRA)*, pages 1436–1443, 2018.

[89] Puze Liu, Davide Tateo, Haitham Bou Ammar, and Jan Peters. Robot reinforcement learning on the constraint manifold. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1357–1366. PMLR, 08–11 Nov 2022.

[90] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[91] Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[92] Jingpei Lu, Florian Richter, and Michael C. Yip. Markerless camera-to-robot pose estimation via self-supervised sim-to-real transfer, 2023.

[93] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2020.

[94] Masahiro Ono and B. C. Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *2008 47th IEEE Conf. Decis. and Control*, pages 3427–3432, 2008.

[95] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. *arXiv preprint arXiv:1805.11199*, 2018.

[96] Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020.

[97] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[98] S. Patil, J. van den Berg, and R. Alterovitz. Estimating probability of collision for safe motion planning under gaussian motion and sensing uncertainty. In *2012 IEEE Int. Conf. Robot. and Automation*, pages 3238–3244, 2012.

[99] Elizabeth Peiros, Zih-Yun Chiu, Yuheng Zhi, Nikhil Shinde, and Michael C. Yip. Finding biomechanically safe trajectories for robot manipulation of the human body in a search and rescue scenario. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[100] Adam Pettinger, Farshid Alambeigi, and Mitch Pryor. A versatile affordance modeling framework using screw primitives to increase autonomy during manipulation contact tasks. *IEEE Robotics and Automation Letters*, 7(3):7224–7231, 2022.

[101] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 2017.

[102] A. H. Qureshi, J. Dong, A. Choe, and M. C. Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[103] Ahmed H. Qureshi, Jiangeng Dong, Asfiya Baig, and Michael C. Yip. Constrained motion planning networks X, 2020. *arXiv:2010.08707*.

[104] Ahmed H. Qureshi, Jiangeng Dong, Austin Choe, and Michael C. Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[105] Ahmed H Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Trans. on Robotics*, 2020.

[106] 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, 2019.

[107] Ahmed H Qureshi and Michael C Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[108] Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[109] Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 2016.

[110] Ahmed Hussain Qureshi, Jiangeng Dong, Asfiya Baig, and Michael C. Yip. Constrained motion planning networks x. *IEEE Trans. on Robotics*, 2022.

[111] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[112] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[113] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[114] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[115] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[116] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 02 2021.

[117] C. Rösmann, F. Hoffmann, and T. Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ Int. Conf. Intell. Robots and Systems*, pages 5681–5686, 2017.

[118] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686, 2017.

[119] Edward Schmerling and Marco Pavone. Evaluating trajectory collision probability through adaptive importance sampling for safe motion planning, 2017. *arXiv:1609.05399*.

[120] 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. *Int. J. Rob. Res.*, 33(9):1251–1270, aug 2014.

[121] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *Int. Conference on Machine Learning*, 2018.

[122] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[123] M. Stilman. Task constrained motion planning in robot joint space. In *2007 IEEE/RSJ Int. Conf. Intell. Robots and Systems*, pages 3074–3081, 2007.

[124] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Auto. Magazine*, 2012.

[125] Chansu Suh, Terry Taewoong Um, Beobkyoon Kim, Hakjong Noh, Munsang Kim, and Frank C. Park. Tangent space rrt: A randomized planning algorithm on constraint manifolds. In *2011 IEEE International Conference on Robotics and Automation*, pages 4968–4973, 2011.

[126] Wen Sun, Luis G. Torres, Jur van den Berg, and Ron Alterovitz. Safe motion planning for imprecise robotic manipulators by minimizing probability of collision. In *Robot. Res.*, volume 114 of *STAR.*, pages 685–701, 2013.

[127] Zaid Tahir, Ahmed H Qureshi, Yasar Ayaz, and Raheel Nawaz. Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments. *Robotics and Autonomous Systems*, 2018.

[128] 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.

[129] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robot. (Intell. Robot. and Autonomous Agents)*. The MIT Press, 2005.

[130] Ludovic Trottier, Philippe Giguère, and Brahim Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks. *CoRR*, abs/1605.09332, 2016.

[131] Jur van den Berg, Pieter Abbeel, and Ken Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *Int. J. of Robot. Res.*, 30(7):895–913, 2011.

[132] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The Int. J. of Robot. Res.*, 31(11):1263–1278, 2012.

[133] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, 2017.

[134] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[135] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[136] B. Wilcox and M. C. Yip. SOLAR-GP: Sparse online locally adaptive regression using gaussian processes for bayesian robot model learning and control. *IEEE Robot. and Automat. Lett.*, 5(2):2832–2839, 2020.

[137] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[138] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, March 2006.

[139] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *IEEE Conf. Comput. Vision and Patt. Recognit. (CVPR), 2018*. IEEE, 2018.

[140] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Int. Conf. on Machine Learning*, 2020.

[141] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. In *Int. Conf. on Learning Representations*, 2022.

[142] Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

[143] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. In *Int. Conf. on Learning Representations*, 2022.

[144] 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.

[145] Yuheng Zhi, Nikhil Das, and Michael Yip. Diffco: Auto-differentiable proxy collision detection with multi-class labels for safety-aware trajectory optimization. *arXiv:2102.07413*, 2021.

[146] H. Zhu and J. Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robot. and Automat. Lett.*, 4(2):776–783, 2019.