Model-Predictive Motion Planning Networks for Quadrotors in Cluttered Environments

Yuxuan Chen Aniket Gujarathi Jiachen Zhou

University of Toronto Institute for Aerospace Studies

April 22, 2022

1 Introduction

Motion planning algorithms decide how the robot will achieve its goal of moving from initial starting point to the destination in an efficient way. The "efficient way" may suggest finding the shortest path, finding the quickest path to reach the goal or finding the path which satisfies the kinematic and dynamic constrains on the robot. These constraints are considered in the planning algorithm and an output is a path which the robot can traverse obeying the required conditions. A subfield of motion planning is kinodynamic motion planning (KMP) that deals with finding a path for the robot while also satisfying the kinematic and dynamic constraints on the robot. This project focuses on solving the kinodynamic motion planning problem for underactuated robots (specifically quadrotors) using a learning based approach called MPC-MPNet [1].

As explained in [2], the classical way to approach kinodynamic motion planning is to decouple the planning problem into simpler subproblems: (1) planning a geometric path that respects the kinematic constraints and (2) planning the derivatives of the configuration variables along the path with respect to physical constraints. These decoupled approaches can be very efficient tools, but suffer from two main drawbacks. First, obtaining good quality paths is challenging since the considered cost-function may not even be defined in the configuration space. Second, although a kinodynamic motion planning problem can have a solution, a decoupled approach may fail to find them.

To avoid the aforementioned problems, sampling-based planning approaches like [3], [4] were suggested that search for solutions directly in the control or the state space rather than in the configuration space. However, these methods still take long computation times and the performance further deteriorates in high dimensional spaces. Recently, learning based methods attempt to solve this problem efficiently using reinforcement learning or imitation learning based methods.

For this project, we will explore an imitation learning based approach, Model Predictive Motion Planning Networks (MPC-MPNet) [1]. It has a deep neural generator and discriminator, which once trained using expert data outputs feasible paths that satisfy the given kinodynamic constraints. This method is an extension to the MPNet [5] that leverages collision-aware Model Predictive Control (MPC) methods in the planning loop for parallelizable steering. As opposed to MPNet, MPC-MPNet performs only feedforward path expansion and avoids re-planning by building a set of informed possible paths. MPC-MPNet is specifically designed to solve the KMP problem for underactuated robots. For the scope of this project, we will focus on planning specifically for quadrotors in cluttered environments.

Kinodynamic Motion Planning has wide range of applications for quadrotors and UAVs ranging from drone racing, performing acrobotic motions to quadrotor flight in cluttered environments [6]. The motion planning problem becomes even more complex for underactuated robots. Furthermore, complexity of motion planning for quadrotors increases due to its significant dynamics imposing severe constraints on the allowable velocities at each configuration. Through this project, we try to solve this problem using MPC-MPNet to generate kinodynamically feasible and optimal paths.

We organize the remaining report as follows. Section 2 explains the background relevance of the kinodynamic motion planning problem and the approaches used to solve it. Section 3 explains the related literature

to solve the KMP problem. Section 4 delves into the details of the methodology of MPC-MPNet. Section 5 describes the implementation details for the quadrotor experiments using MPC-MPNet. Section 6 explains the quantitative and qualitative results of the proposed approach with other benchmarks. Finally, section 7 concludes the report with some discussion and comments for future research.

2 Background and Relevance

2.1 Kinodynamic Motion Planning

In robotics and motion planning, kinodynamic motion planning (KMP) is a class of problems for which control bounds i.e force/torquue, velocity/acceleration bounds must be satisfied along with kinematic constraints such as avoiding obstacles. For many robots such as ground vehicles at high speeds, UAVs, fixed-wing aerial vehicles, etc. it is difficult to adapt a collision-free path into a feasible one given the underlying dynamics. To solve this problem, Donald et.al [7] proposed the idea of kinodynamic motion planning. The kinodynamic planning problem is to synthesize a robot motion subject to simultaneous kinematic constraints, such as avoiding obstacles, and dynamics constraints, such as modulus bounds on velocity, acceleration, and force. This is a harder problem than kinematic motion planning, as it involves searching a higher-dimensional space while simultaneously managing the constraints introduced by robot dynamics. Existing methods solve KMP using sampling-based planners such as [3], [4] etc., or lattice based planners like [8].

2.2 Lattice Based Kinodynamic Motion Planning

Like other motion planning problems, the core task of a KMP is combinatorial search. Some of the earliest works in KMP included the formulation of kinodynamic planning as a discrete search problem involving construction of lattice of states connected locally by a fixed set of control trajectories. These lattice state structures allowed the problem to be formulated as a dynamic programming-based search over a state transition graph. Differential constraints are incorporated into the state space through some pre-computed motion primitives. This generated graph represents a discrete search space which can be explored using efficient graph search algorithms. Although, these lattice based planners perform well, especially for ground robots, they do not generalize well to arbitrary state-space dynamics. Hence, alternatives to lattice based planners have been sought.

2.3 Sampling Based Kinodynamic Motion Planning

Sampling Based Motion Planners solve KMP by constructing a tree that originates from a start state and exhaustively searches the state-space to find the goal state. To satisfy kinodynamic constraints, the edges to join any two intermediate states are formed using a local steering function. In case of a dynamical system, the steering function corresponds to the solution of a two-point boundary value problem (BVP). However, it is not easy to produce optimal BVP solutions. This BVP is known to be NP-Hard [9] and fails when the boundary constraints are not satisfied. Moreover, these methods take long computation times as the complexity of environment increases due to uniform sampling in state and control spaces.

2.4 Learning Based Kinodynamic Motion Planning

Recently there has been a rise in planners that learn controllers for steering through reinforcement learning (RL) [10], [11] and imitation learning [12]. RL based methods learn optimal policies by interacting with the world exhaustively. However, this exhaustive searching requires lot of data and computational resources.

Planners such as [5] generate end-to-end paths by planning in a learned embedding space with a bidirectional planning and replanning strategy. [1] is an exension to MPNet that finds kinodynamically feasible and optimal paths for underactuated robots. These learning based methods outperform the classical planners in terms of computational speed and generalize to unseen environments with a high success rate. We will go deeper into some specific approaches (learning-based and classical) to solve the kinodynamic motion planning problem in Section 3.

3 Related Work

There are two general approaches to handle continuous configuration spaces in solving the motion planning problem, combinatorial planning and sampling-based planning. The former explicitly specifies the connectivity of the configuration space, converting it into a graph of nodes linked by edges and finding solutions with completeness guarantees via a variety of search algorithms, e.g. dijkstra's, A^* [13]. However, these methods quickly become computationally intractable in higher-dimensional spaces with complex obstacles, making them unattractive in read-world scenarios. This led to the development of the latter more efficient approach, sampling-based planning, that comes with weaker guarantees, such as probabilistic completeness. These methods take random samples of points in the configuration space, and connect them, free of collision with obstacles, to form paths from the start to the goal pose. These methods are probabilistically complete because the probability of finding a solution, if one exists, approaches one as the number of sampling points goes to infinity.

3.1 RRT, RRT*

A prominent work in sampling-based planning is Rapidly-exploring Random Trees (RRT) [3]. It works by rapidly sampling the configuration space from a random initial state and generating a new sampled node from its neighbouring region. It then connects the new node to the nearest existing node in the tree. Besides probabilistic completeness, RRT also has anytime resolution [3], which means the representation of the environment becomes increasingly accurate as long as the iteration permits and continues. Although the RRT algorithm is able to quickly find a path solution, if one exists, it lacks any heuristic to guide the sampling process towards the goal and thus fails to find the shortest path. An optimal variant of RRT, called RRT* [14], rewires the tree produced by RRT to ensure i) every new node is connected to a neighbouring node in the existing tree such that the path to it from the start is the shortest, 2) with this new node added, the neighbouring nodes could be disconnected from their previous parent nodes and reconnected to the new node if the path through the new node would lead to an even shorter path. It is found that RRT* asymptotically guarantees to find the shortest path [14]. However, the search for neighbours and the rewiring makes RRT* more computationally inefficient when the dimensionality of the planning problem increases. Furthermore, as shown in [5], RRT* is no better than grid search methods in higher dimensional spaces.

3.2 SST

For many real-world robots, it is not trivial to adapt a collision-free path into a feasible trajectory given the underlying kinodynamic constraints. Kinodynamic planning is more difficult than kinematic path planning, as it involves searching a higher-dimensional space while respecting the contraints that emerge from the dynamics. In the case of a dynamical system, the steering function that returns the optimal path between two states free of obstacles solves a two-point boundary value problem (BVP), which corresponds to solving a different equation. However, it is often difficult or even impractical to generate such a BVP solver for various dynamical models. To achieve sampling-based asymptotic optimality in kinodynamic planning, Li et al. [4] propose Stable Sparse RRT (SST) and SST* which are asymptotically near-optimal and optimal respectively. Moreover, since they only maintain a sparse set of samples, they are shown to be computationally efficient.

3.3 MPNet

A recent research wave has led to the intersection of motion planning and machine learning, focusing on the development of learning based planner, to address computational efficiency with guarantees of completeness and optimality. One of the pioneer work in this field is Motion Planning Networks (MPNet) [5] which shows promising performance in solving complex planning problems by learning general near-optimal heuristics for path planning. Specifically, MPNet [5] consists of two neural networks: an encoder network that encodes the environment into a latent representation, and a planning network that takes the embedding and finds a path. Both networks can be trained together in an end-to-end fashion using the planning network loss function or separately with their individual loss functions. To train the encoder network, contractive autoencoders are used to learn robust features via reconstruction loss between the reconstructed point cloud and the

original observation. The planning network is a feed-forward deep neural network and is trained using a one-step look ahead prediction strategy. The expert demonstration is a list of waypoints from the start to goal configuration, provided by an offline RRT* algorithm [14]. The training objective is to minimize the mean-squared-error (MSE) loss between the predicted next waypoint and the target waypoint in the next time step. The performance of both paths cost and computation time shows that MPNet path planner is not only more than 100× faster than RRT* and does not grow exponentially with the dimensionality of the planning problems, but is also able to find paths that are within a 10% paths cost margin of RRT* [14]. Moreover, [5] formally presents the proof to show MPNet exhibits probabilistic completeness as well as asymptotic optimality.

4 Methodology

In this section we outline the problem definition by describing the details of the configuration space and quadrotor dynamics. To solve the motion planning problem for quadrotors in 3D cluttered environments, we adopt a similar methodology as in [1, 5], which is an end-to-end learning-based KMP algorithm that iteratively generates waypoints, then optimizes local trajectories using MPC to avoid obstacle collisions.

4.1 Problem Definition

4.1.1 Configuration Space

The robot configuration space (C-space) is denoted as $\mathcal{C} \subset \mathbb{R}$, where d is the C-space dimensionality. The obstacle space is denoted as $\mathcal{C}_{obs}$ and the obstacle-free space is denoted as $\mathcal{C}_{free} = \mathcal{C} / \mathcal{C}_{obs}$. The workspace of the robot is denoted as $\mathcal{X} \subset \mathbb{R}^m$, where m is a workspace dimension. Similar to the C-space, we define the obstacle-free workspace as $\mathcal{X}_{free} = \mathcal{X} / \mathcal{X}_{obs}$.

The state $x = (c, \dot{c}) \in \mathcal{X}$, contains a configuration $c \in \mathcal{C}$ and the time derivative $\dot{c}$. The system's dynamics model can be formulated as $\dot{x} = f(x, u)$, where u represents the control input to a system from a feasible set of control $\mathcal{U}$.

Given the initial state x_{init} and goal region $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$, the objective is to find a feasible collision-free trajectory $\sigma = [(x, \mu, \tau)_t]_{t=0}^{T}$, which comprises a ordered sequence of states $[x_t]_{t=0}^{T} \rightarrow \mathcal{X}$ and controls $[u_t]_{t=0}^{T} \rightarrow \mathcal{U}$ with their corresponding durations $[\tau_t]_{t=0}^{T}$, such that $x(0) = x_{init}$, $x(T) \in \mathcal{X}_{goal}$.

4.1.2 Quadrotor Dynamics

We consider the quadrotor robotic systems as Figure 1 for the cluttered, kinodynamically constrained environments. The state space for quadrotor dynamics is defined as: $[p, q, \dot{p}, w]$, where p and q represent the position and orientation of the quadrotor respectively. The corresponding time derivatives for the position and orientation are represented as $\dot{p}$ and ω. The control space for the quadrotor is 4 dimensional with bound $[-15, -5] \times [-1, 1]^3$, where obstacles are randomly placed in the workspace.

4.2 Model Predictive Motion Planning Network

MPNet comprises of two neural networks: an observation encoder network and a neural generator network. After converting the surrounding environment of the quadrotor into 3D voxel data, the observation encoder takes the voxel data as input, then encode it into a latent space embedding. The neural generator takes the environmental encoding, the current state of the robot, and the goal state as inputs, then outputs sample for either a path or tree expansion.

In addition, we train a neural discriminator that predicts the cost of each sampled node, and select the one with the minimal cost.

After planning trajectories using MPNet, Model Predictive Control is used as a steering function to optimize the local trajectories.

In this section, we also outline two planning algorithms: MPC-MPNetPath and MPC-MPNetTree. These two methods differ in the ways of sampling new nodes to expand the search tree in order to find a path

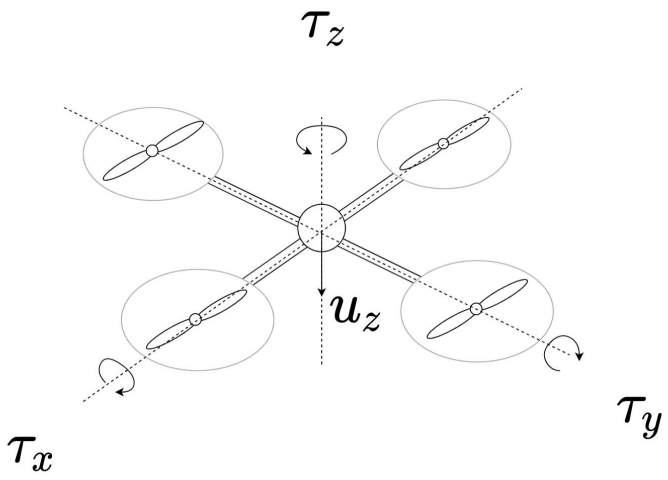


Figure 1: Quadrotor system dynamics

solution. MPC-MPNetPath extends nodes radially with subset selection, whereas MPC-MPNetTree expands towards the goal region with parallelized MPC.

4.2.1 Observation Encoder

The workspace information is represented as volumetric voxel maps v with dimensions $L \times W \times H \times C$, which corresponds to length, width, height, and number of channels. In order to save computational time, we use 2D convolutional neural network instead of commonly used 3D convolutional neural network as observation encoder to process the voxel maps.

To preprocess the voxel maps, the voxel maps are converted into voxel patches with dimensions $L \times W \times \hat{C}$, where $\hat{C} = HC$. The observation encoder is a 2D-CNN that embeds the voxel maps into latent features Z, which encodes critical underlying anchor points for training the neural generator and discriminator.

4.2.2 Neural Generator

We denote the neural generator as G, with parameter θ_G. The model takes the given environment features Z from the output of the observation encoder, along with the quadrotor's current state $x_t \in X_{free}$, and the goal state $x_{goal} \in X_{goal}$ as inputs, then outputs intermediate waypoints $\hat{x}_{t+1}$. This is demonstrated as Equation 1.

The demonstration trajectory is a list of waypoints, $\sigma^* = \{x_0^*, x_1^*, \ldots, x_T^*\}$, connecting the start and goal states in obstacle-free space.

$$\hat{x}_{t+1} \leftarrow G_{\theta_G}(Z, x_t, x_{goal}; \theta_G) \tag{1}$$

The stochastic neural generator adopts dropout in almost every trained network layer to generate random samples. The neural generator is trained end-to-end with the observation encoder that minimizes the mean-squared-error(MSE) loss between the predicted subsequent states $\hat{x}$ and the ground truth states x^* from the demonstration trajectories σ^*. The training objective is shown in Equation 2.

$$L_{G_{\theta_G}} = \frac{1}{N_p} \sum_{i=1}^{N} \frac{1}{T_i} \sum_{j=0}^{T_i - 1} \left\| \hat{x}_{i,j+1} - x_{i,j+1}^* \right\|^2 \tag{2}$$

where $N \in \mathbb{N}$ is the total number of traning trajectories, N_p is the averaging term, and T_i is the length of each trajectory i in the dataset.

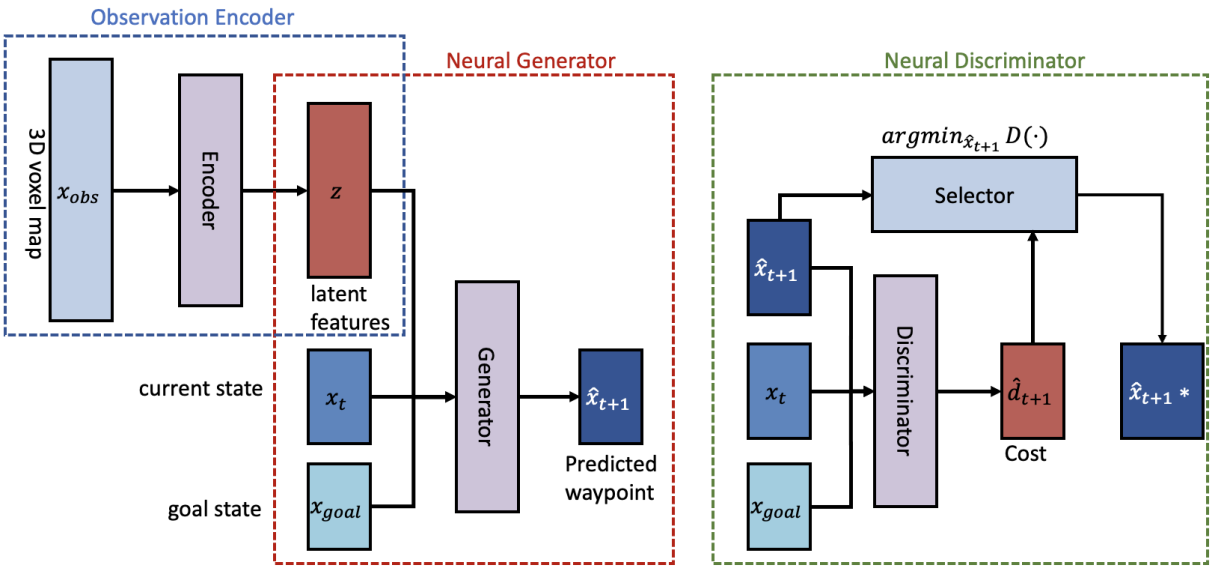


Figure 2: **Overview of the Motion Planning Network.** MPNet consists of an encoder network and a planning network. The encoder network is a 2D convolutional neural network that processes voxel patches x_{obs} then embeds them into latent feature Z. The planning network takes the encoding Z, current state of the robot x_t, and the goal state x_{goal} as inputs, then generates collision-free intermediate waypoints. Both networks can be trained end-to-end. Neural discriminator is trained to predict the cost of the predicted waypoints to prune out inefficient waypoints,

4.2.3 Neural Discriminator

The predicted batch of the next waypoints from the neural generator are usually scattered towards the goal region due to the stochastic nature of the network. The neural discriminator network filters out the best waypoint from the batch by predicting a given waypoint's cost-to-go to reach the goal state. Here we denote the neural discriminator as D, and the parameters of the network as θ_D. The inputs of the networks are: environment features Z, the quadrotor's current state x_t and goal state x_{goal}. Given these inputs, the network would then output a prediction of the cost for each waypoint, which is shown in Equation 3.

$$\hat{d}_t \leftarrow D_{\theta D}(Z, x_t, x_{goal}; \theta_D) \tag{3}$$

In order to train the neural discriminator network, we obtain the ground truth costs from the demonstration trajectory, and define a loss function as the mean-squared-error between the predicted costs and the ground truth costs, which is shown in Equation 4:

$$\mathcal{L}_{D_{\theta_D}} = \frac{1}{N_p} \sum_{i=1}^{N} \frac{1}{T_i} \sum_{j=0}^{T_i - 1} \left\| \hat{d}_{i,j+1} - \left(\sum_{k=j+1}^{T_i} d_{i,k*} \right) \right\|^2 \tag{4}$$

where $d^*_{i,k}$ is the cost to the goal state from the waypoint k in the demonstration trajectory i.

In addition, the training samples are augmented by assigning large penalties to unreachable waypoints in order to balance the positive and negative training samples in the generated datasets. With such augmentation, the trained discriminator network predicts high costs for invalid states to filter out anomalous waypoints.

4.2.4 Model Predictive Control

In order to satisfy the kinodynamic constraints of the planned trajectories, MPC is used as a steering function, and the overall algorithm outline is shown in Algorithm 1. MPC takes the current state x_t and the

target state $\hat{x}_{t+1}$, then generates an optimized local trajectory $\sigma_t = [(x, \mu, \tau)_t]$ for the duration of τ, which minimizes the cost between the propagated terminal state and $\hat{x}_{t+1}$.

Here we define a parametrized distribution $(\mu, \tau)_i \sim \mathcal{D}(u, \tau; \theta_{mpc})$, where MPC samples controls and durations from. We make an assumption that the distribution is Gaussian. Given the starting state x_t, the algorithm propagates and generates the steering trajectory σ_i. Here we define a cost function as Equation 5.

$$d_s = d(\sigma_i, \hat{x}_{t+1}) + d_c(\sigma_i) \tag{5}$$

where $d(\cdot)$ represents the distance metrics between the given states, and $d_c(\cdot)$ is a collision penalty function that discourages in-collision trajectory expansions.

The cost function is used to rank all propagated trajectories, and we select the propagated states with the lowest cost. These states are then used to update the parameters of MPC θ_{mpc} by minimizing the cross-entropy loss between the distributions of steered terminal states and the target states.

Algorithm 1 Model Predictive Control

```
Initilize parameters θ_mpc
for iter ← 1 to N_iter do
    for μ_i, τ_i ~ D(μ, τ; θ_mpc) do
        Propagate the current state x_t with u_i, τ_i to generate σ_i
        Compute scores for each propagation as d_s = d(σ_i, x̂_{t+1}) + d_c(σ_i)
        Rank all propagations and select samples with the lowest cost
        Update θ_mpc with elite samples
        Update optimal trajectory with the best σ_i^*
    end for
end for
```

In addition, for the MPC-MPNetTree planning algorithm, we use GPU to process multiple waypoints in parallel. To represent the parallelized processes, we denote all vectorized inputs in batch form as:

$$B_t = \begin{bmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^{N_B} \end{bmatrix}, B_{goal} \begin{bmatrix} x_{goal} \\ x_{goal} \\ \vdots \\ x_{goal} \end{bmatrix}, B_Z \begin{bmatrix} Z \\ Z \\ \vdots \\ Z \end{bmatrix} \tag{6}$$

where MPC processes $N_B \in \mathbb{N}$ samples in parallel. B_t, B_{goal}, and B_Z correspond to the batch of current states, desired states, and observation encodings.

During execution, the neural generator outputs a batch of next states B_{t+1} with variety as Equation 7, and the neural discriminator predicts their associated costs $B_{d_{t+1}}$ as Equation 8.

$$\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g) \tag{7}$$

$$B_{d_{t+1}} \leftarrow D(B_Z, \hat{B}_{t+1}, B_{goal}; \theta_d) \tag{8}$$

Given the start states B_t and target states $\hat{B}t + 1$, the parallelized MPC generates the corresponding local kinodynamic trajectories as in Equation 9.

$$B_{\sigma_t} \leftarrow MPC(B_t, \hat{B}_{t+1}) \tag{9}$$

where $B_{\sigma_t} = [(x_i, u_i, \tau_i)_t]_{i=1}^{N_B}$ is a batch of local trajectories at time t.

4.2.5 MPC-MPNetPath

The MPC-MPNetPath algorithm is outlined in Figure 3 and Algorithm 2. The neural generator G generates a batch of new samples B_t. The discriminator D predicts the cost of each sample, then select a sample $\hat{x}_{t+1}^*$ from the batch with a minimum cost $\hat{d}$ to reach the given target x_{goal}. The MPC module takes the current

Algorithm 2 MPC-MPNetPath

$T \leftarrow x_{init}, B_t \leftarrow x_{init}$
$B_Z \leftarrow Z, B_{goal} \leftarrow x_{goal}$
for $i \leftarrow 1$ to n **do**
    $\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g)$
    $\hat{x}_{t+1}* \leftarrow argmin_{\hat{x}_{t+1}} D(B_Z, \hat{B}_{t+1}, B_{goal}; \theta_d)$
    $\sigma_t = MPC(x_t, \hat{x}_{t+1}*)$
    if Invalid(σ_t) **then**
        $B_t \leftarrow randomNode(T)$
    else
        $addToTree(\sigma_t, T)$
        set batch B_t with a terminal state of σ_t
    end if
    if $Reached(T, x_{goal})$ **then**
        Return $ExtractedTotalPath(T)$
    end if
end for
Return None

state x_t and the selected next state $\hat{x}^*_{t+1}$ to perform the kinodynamic steering, leading to a local trajectory σ_t. The terminal state of σ_t is the resulting valid state x_{t+1}, after performing the control input u for a duration of τ at state x_t. The resulting state is as close as possible to the selected desired state $\hat{x}_{t+1}$ while satisfying the kinodynamic constraints. If there is no collision detected, the local trajectory is considered as valid, and is added to the tree. However, if collision is detected, a random node from the tree is selected and is treated as the new current state x_t for the next planning iteration. Once the algorithm has reached to a proximal region around the goal, we can extract an end-to-end path that links the start and goal states under kinodynamical constraints.

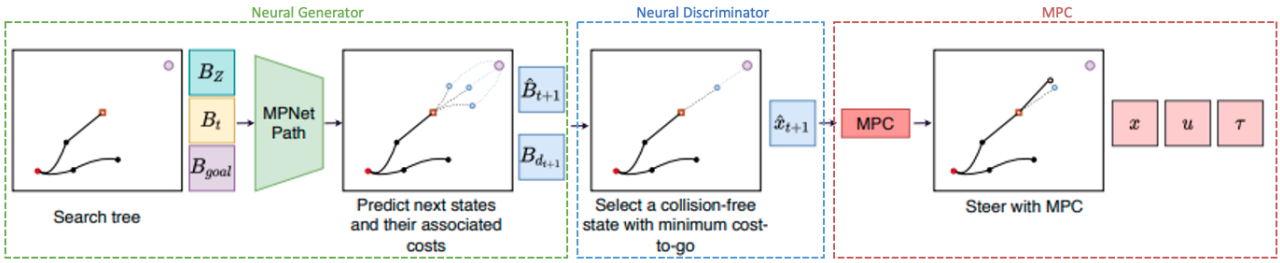


Figure 3: **MPC-MPNetTree Planning Algorithm [1].** In each iteration, the neural generator predicts a batch of next states from a given current and select a collision-free state with a minimum estimated cost for the tree expansion using MPC.

4.2.6 MPC-MPNetTree

This method leverages the parallelized MPC framework to expand multiple nodes of the search tree at the same time, hence does not require the neural descriminator to prune out inefficient intermediate waypoints. This algorithm is outlined in Figure 4 and Algorithm 3. In each iteration, MPC-MPNetTree samples a set of random states B_{rand} and finds their corresponding nearest neighbors in the tree. These nearest nodes are treated as the current set of states B_t for the subsequent expansion procedures. The neural generator G outputs the next batch of samples and MPC computes their local trajectories B_{σ_t} for each sample. The collision-free trajectories generated by MPC are then added to the search tree, and the final path is extracted once the tree reaches to the given goal region B_{goal}.

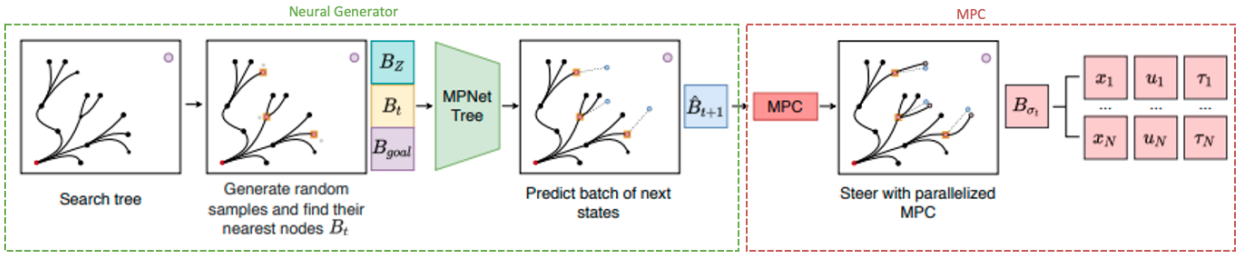


Figure 4: **MPC-MPNetTree Planning Algorithm [1].** The neural generator predicts a batch of next states from nearest neighbors of random states inside the search tree. Our parallelized MPC finds the local controllers between randomly selected start and neurally generated next states for extending multiple branches of tree simultaneously towards the given goal state.

Algorithm 3 MPC-MPNetTree

$T \leftarrow x_{init}$
$B_Z \leftarrow Z, B_{goal} \leftarrow x_{goal}$
for $i \leftarrow 1$ to n **do**
  $B_{rand} \leftarrow RandomSample()$
  $B_t \leftarrow NearestNeighbor(B_{rand}, T)$
  $\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g)$
  $B_{\sigma_t} \leftarrow MPC(B_t, \hat{B}_{t+1})$
  $addToTree(B_{\sigma_t}, T)$
  if $Reached(T, x_{goal})$ **then**
    Return $ExtractedTotalPath(T)$
  end if
end for
Return None

5 Implementation Details

We generate our own datasets by randomly placing 10 obstacles in the workspace. We create 9 environments with different placements of the obstacles. For each environment, we generate 32 different trajectories by randomly sampling the start and goal states. We reserve 20% of the data for testing. In addition, we create 2 unseen environments to test the generalizability of the learned model by ensuring the environments are different from the training and testing data. We sample 200 valid start and goal pairs for evaluation on the unseen environments. Hence, 11 unique environments are created during the data generation process, and the point-cloud of the environments are obtained by sampling the obstacle space and are processed into voxel v of size $32 \times 32 \times 32$. The demonstration trajectories for the training data are obtained using the SST algorithm [4].

The neural network modules are implemented using Pytorch. After end-to-end training, the models are exported to C++ with Torchscript for testing. All experiments are performed using GeForce GTX 3080 GPU and Intel Core i7 processor.

The observation encoder is a 2-layered 2D CNN and MaxPool with two linear layer attached at the end. The input size of the network is set to 32, and the output size of the network is set to 64. After processing each patch of the voxel map, we concatenate the output features into one vector Z. The neural generator consists of 6 linear layers with Dropouts, which has an output size of 64. For MPC-MPNetTree algorithm, we train the MPNet separately on the training set for 1000 epochs, with a batch size of 128 and a learning rate of $3e-4$. As for the MPC-MPNetPath planning algorithm, we train the MPNet and neural discriminator end-to-end on the same settings.

6 Results

We conduct a set of experiments for quadrotor motion planning in various simulated cluttered environments, and compare the computation time (measured as time-to-reach), path quality of MPC-MPNetPath, MPC-MPNetTree, SST [4], and RRT [3] planning algorithms. We reserve 2 unseen environments that are not used during training for evaluating the generalizability of the learned model.

Table 1: The *mean*($\pm$*std.dev.*) of computational time and path costs and number of successful trajectories achieved. Preferable results are **bolded**.

Model	seen	Runtime	Path Cost	Num. of Successful Traj.
MPC-MPNet Path	✓	**1.46**($\pm$1.89)	3.34($\pm$6.55)	285
MPC-MPNet Tree	✓	1.56($\pm$2.22)	**0.9**($\pm$2.26)	**289**
SST	✓	8.18($\pm$8.18)	2.02($\pm$3.34)	270
RRT	✓	6.31($\pm$7.35)	1.78($\pm$2.97)	186
MPC-MPNet Path	✗	**2.20**($\pm$3.71)	5.55($\pm$6.53)	108
MPC-MPNet Tree	✗	2.23($\pm$3.68)	**1.74**($\pm$3.12)	**109**
SST	✗	9.07($\pm$8.71)	4.08($\pm$4.59)	34
RRT	✗	8.49($\pm$8.05)	3.18($\pm$3.85)	48

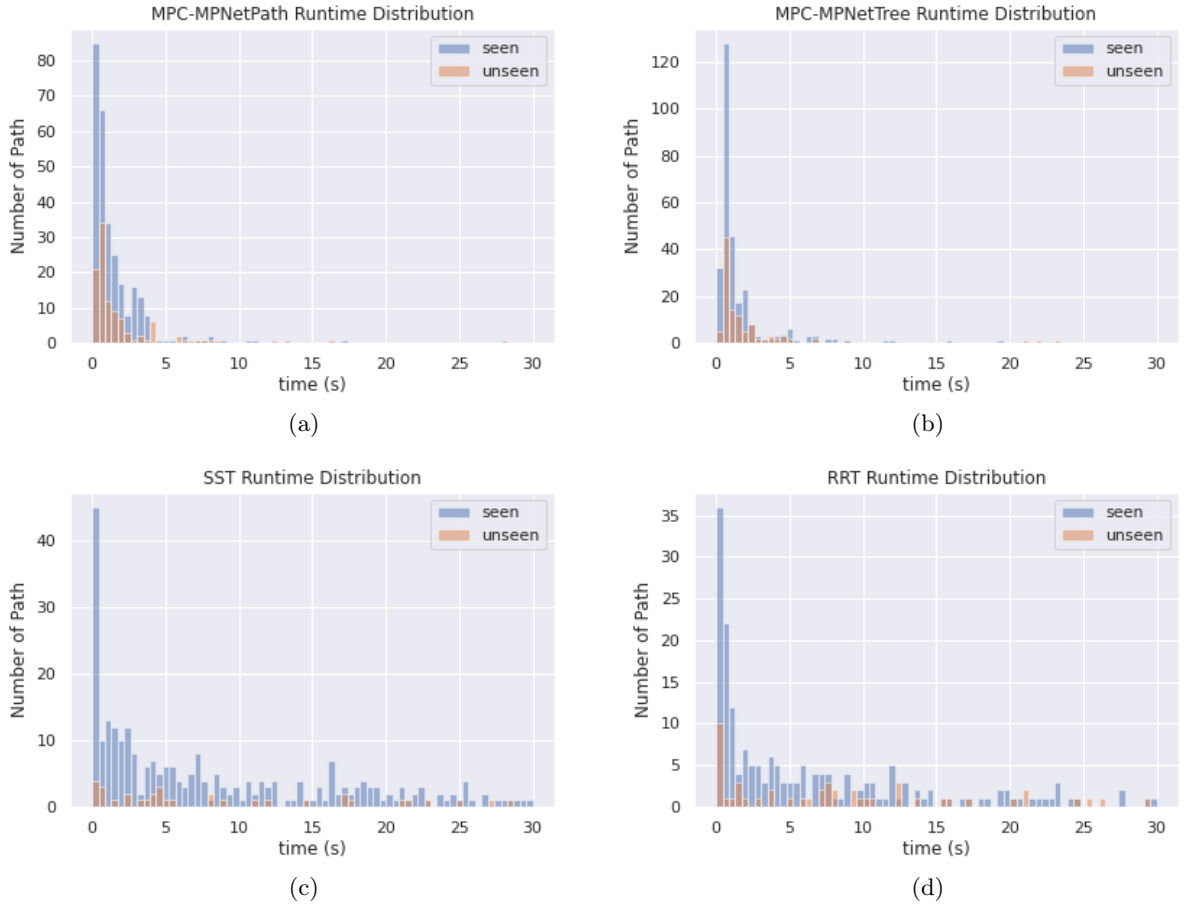
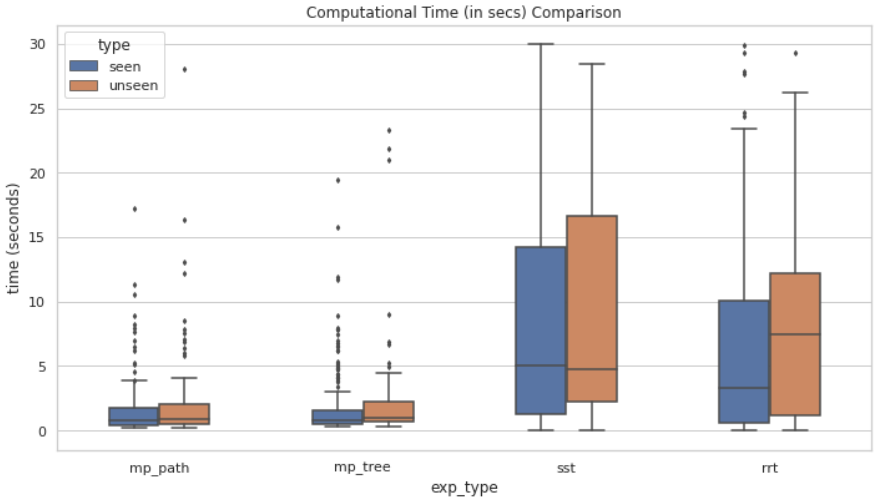


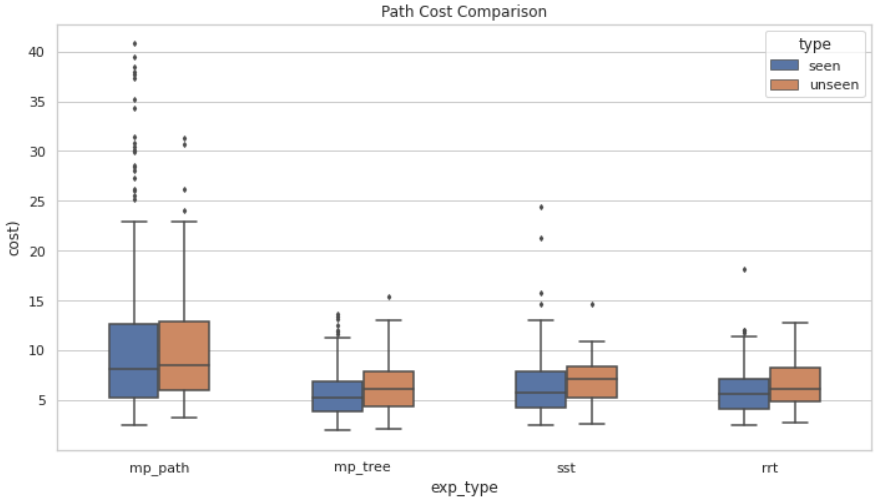
Figure 5: The runtime distribution of MPC-MPNetPath, MPC-MPNetTree, SST, RRT for Quadrotor in seen and unseen environments.

We present the mean computational time with standard deviation and mean path costs with standard deviation in Table 1 for both seen and unseen environments. In both types of environments, the number of obstacles is kept the same while the start and goal states are randomly sampled every run. From the table, it is clear that no matter if it is a seen or an unseen environment, both the MPC-MPNet methods construct more successful trajectories while only using a fraction (less than 1/4) of runtime compared to SST [4] and RRT [3]. MPC-MPNetTree also outperforms SST [4] and RRT [3] by a large margin in terms of path cost. However, the trajectories generated by the MPC-MPNetPath, despite having more successful ones, are more costly.

We further visualize the histogram distribution of the runtime for all trajectories across models in Figure 5. Both MPC-MPNet methods exhibit a similar heavily right skewed distribution with most runtime values below 2 seconds and very few data points beyond 4 seconds. On the contrary, SST [4] and RRT [3] models show significantly more data with longer runtimes up to 30 seconds.



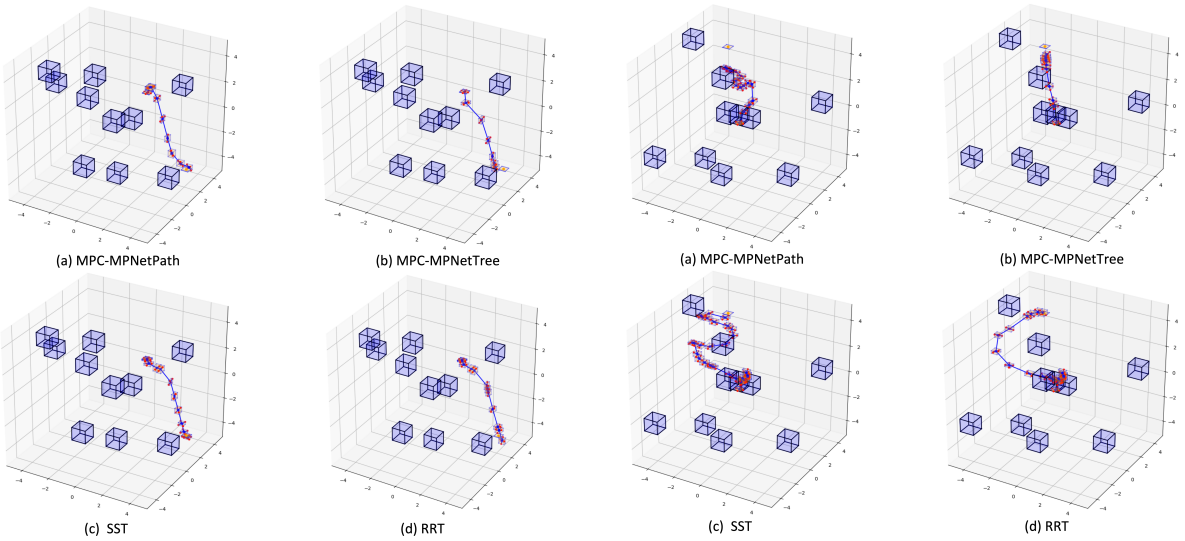
(a) Computational time (in seconds)



(b) Path cost

Figure 6: Comparison of computational time and path cost for Quadrotor in seen and unseen environments.

Moreover, in Figure 6, we compare the distribution of both runtime and path cost across different planning algorithms using boxplots. Figure 6a clearly shows that not only do SST and RRT have higher mean and median values, they also have wide interquartile ranges resulting in a dispersed distribution as previously shown in Figure 5. In Figure 6b, we also present the path cost comparison. All four methods show a slight increase in path cost, switching from the seen environments to the unseen ones. However, since SST [4] and RRT [3] are not learning-based methods, they should be mostly indifferent to the change in different environments, given the same amount of obstacles. We attribute this increase in their path cost to the 2 unseen environments being inherently more challenging for motion planning due to the placement of obstacles and the start and goal states. This could also explain the similar increase in path cost as we can see for MPC-MPNetPath and MPC-MPNetTree, though we do not rule out generalization issue for these two learned methods. Nevertheless, MPC-MPNetTree shows consistently strong performance in terms of both computational time and path cost, outperforming SST [4] and RRT [3] to a great extent.



(a) Quadrotor environment 1 (seen)

(b) Quadrotor environment 10 (unseen)

Figure 7: Quadrotor environment: kinodynamic motion planning of a 12 DoF quadrotor.

Lastly, we visualize some of the successfully planned trajectories in 3D cluttered environments in Figure 7. Figure 7a shows the resulting trajectories in environment #1 that the models have seen in the training set. In this instance, all four methods are able to achieve desired trajectories. Figure 7b, on the other side, depicts an unseen challenging environment #10. This time, both MPC-MPNetPath and MPC-MPNetTree successfully plan trajectories that are significantly shorter than those found by SST [4] and RRT [3]. This suggests that the models trained on the given 9 environments supervised by the expert demonstration from SST [4] are capable of outperforming its demonstrator in an unseen environment.

7 Conclusion

In this project, we demonstrate the performance of MPC-MPNet to solve the kinodynamic motion planning problem for quadrotors. Our experiments show that MPC-MPNet outperforms the classical sampling-based planning approaches in generating collision-free near-optimal paths in complex scenarios. Further, the method also generalizes to unseen environments with high success rate. The scope of this project focuses specifically on quadrotor planning, but MPC-MPNet could be further extended to various robotic systems. One of the main drawbacks of MPC-MPNet is the inability to handle dynamic obstacles. Due to the distributed approach to encode the environment and perform planning in an encoded subspace of a given configuration space, the method is unable to handle dynamic obstacles. The future work could tackle the environment encoding in the planning pipeline to produce an end-to-end pipeline.

References

[1] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496–4503, 2021.

[2] A. Boeuf, "Kinodynamic motion planning for quadrotor-like aerial robots," Ph.D. dissertation, 07 2017.

[3] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[4] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016. [Online]. Available: https://doi.org/10.1177/0278364915614386

[5] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, 2020.

[6] R. E. Allen and M. Pavone, "A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance," *Robotics and Autonomous Systems*, vol. 115, pp. 174–193, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017308692

[7] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. ACM*, vol. 40, no. 5, p. 1048–1066, nov 1993. [Online]. Available: https://doi.org/10.1145/174147.174150

[8] M. Pivtoraiko, R. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices. journal of field robotics (jfr), 26(3), 308-333," *J. Field Robotics*, vol. 26, pp. 308–333, 03 2009.

[9] B. Kacewicz, "Complexity of nonlinear two-point boundary-value problems," *Journal of Complexity*, vol. 18, no. 3, pp. 702–738, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0885064X02906436

[10] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 07 2019.

[11] K. Ota, Y. Sasaki, D. Jha, Y. Yoshiyasu, and A. Kanezaki, "Efficient exploration in constrained environments with goal-oriented reference path," 10 2020, pp. 6061–6068.

[12] W. Wolfslag, M. Bharatheesha, T. Moerland, and M. Wisse, "Rrt-colearn: Towards kinodynamic planning without numerical trajectory optimization," *IEEE Robotics and Automation Letters*, vol. PP, 10 2017.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

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