Motion Policy Networks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract: Collision-free motion generation in unknown environments is a core building block for robot manipulation. Generating such motions is challenging due to multiple objectives; not only should the solutions be optimal, the motion generator itself must be fast enough for real-time performance and reliable enough for practical deployment. A wide variety of methods have been proposed ranging from local controllers to global planners, often being combined to offset their shortcomings. We present an end-to-end neural model called Motion Policy Networks (MπNets) to generate collision-free, smooth motion from just a single depth camera observation. MπNets are trained on over 3 million motion planning problems in $500{,}000$ environments. Our experiments show that MπNets are significantly faster than global planners while exhibiting the reactivity needed to deal with dynamic scenes. They are 46% better than prior neural planners and more robust than local control policies. Despite being only trained in simulation, MπNets transfer well to the real robot with noisy partial point clouds.

Keywords: Motion Control, Imitation Learning, End-to-end Learning

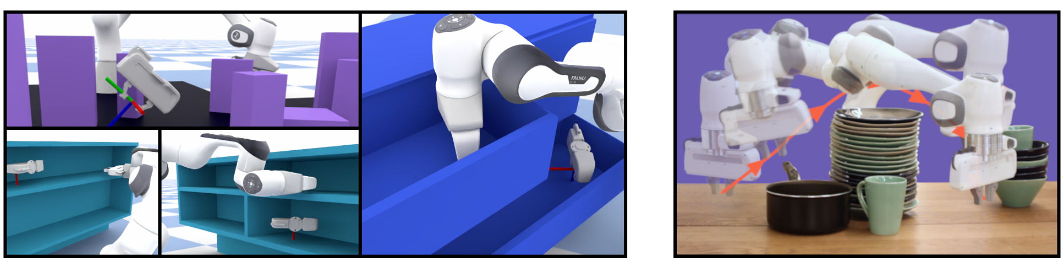

Figure 1: MπNets are trained on a large dataset of synthetic demonstrations (*left*) and can solve complex motion planning problems using raw point cloud observations (*right*).

1 Introduction

Generating fast and legible motions for a robotic manipulator in unknown environments is still an open problem. Decades of research have established many well-studied algorithms, but there are two practical issues that prevent motion planning methods from being widely adopted in industrial applications and home environments that require real-time control. First, it is challenging for any single approach to satisfy multiple planning considerations: speed, completeness, optimality, ease-of-use, legibility (from the perspective of a human operator), determinism, and smoothness. Second, existing approaches enforce strong assumptions about the visual obstacle representations—such as accurate collision checking in configuration space [1] or the availability of a gradient [2, 3, 4]—and hence require intermediate processing to operate in novel scenes directly using raw sensor observations.

Global planners such as RRT [5] are useful to quickly find a feasible path but say nothing about optimality. Other sampling-based approaches iteratively refine their paths to reduce cost and asymptotically approach the optimal solution [6, 7, 8, 9]. Optimization-based approaches [2, 3, 10] embrace locally optimal behavior in exchange for completeness. Recent methods such as Geometric Fabrics [4] and STORM [11] deploy reactive local policies and assume that local decisions will lead to globally acceptable paths. Unfortunately, as we show in our experiments, the performance of these local approaches degrades in more geometrically complex environments as they get stuck in local minima. Motivated by the success of deep learning, neural motion planning approaches such as Motion Planning Networks [12] have been proposed to greatly improve the sampling of an RRT planner with imitation learning. However, they still require a planner and a collision checker with known models at test time.

Planners have traditionally been evaluated with known environment models and perfect state estimation. When deploying them in practice, however, one would have to create one of several scene representations: a static or dynamic mesh, occupancy grids [13, 14], signed distance fields, etc. Reconstruction systems such as SLAM and KinectFusion [15] have a large system start-up time, require a moving camera to aggregate many viewpoints, and ultimately require costly updates in the presence of dynamic objects. Recent implicit deep learning methods like DeepSDF [16] and NERF [17] are slow or do not yet generalize to novel scenes. Methods such as SceneCollisionNet [18] provide fast collision checks but require expensive MPC rollouts at test time. It also draws samples from a straight line path in configuration space which may not generalize to challenging environments beyond a tabletop. Other RL-based methods learn a latent representation from observations but have only been applied to simple 2D [19, 20] or 3D [21] environments in simulation.

We present *Motion Policy Networks (MπNets)*, a novel method for learning an end-to-end policy for motion planning. Our approach circumvents the challenges of traditional motion planning and is flexible enough to be applied in unknown environments. Our contributions are as follows:

- We present a large-scale effort in neural motion planning for manipulation. Specifically, we learn from over 3 million motion planning problems across over $500{,}000$ instances of three types of environments, nearly 300x larger than prior work [12].
- We train a reactive, end-to-end neural policy that operates on point clouds of the environment and moves to task space targets while avoiding obstacles. Our policy is significantly faster than other baseline configuration space planners and succeeds more than local task space controllers.
- On our challenging dataset benchmarks, we show that MπNets is nearly 46% more successful at finding collision-free paths than prior work [12] without even needing the full scene collision model.
- Finally, we demonstrate *sim2real* transfer to real robot partial point cloud observations.

2 Related Work

Global Planning: Robotic motion planning typically splits into three camps: search, sampling, and optimization-based planning. Search-based planning algorithms, such as A* [22, 23, 24], discretize the state space and perform a graph search to find an optimal path. While the graph search can be fast, complete, and guaranteed optimal, the requirement to construct a discrete graph hinders these algorithms in continuous spaces and for novel problems not well covered by the current graph. Sampling-based planners [5] function in a continuous state space by drawing samples and building a tree. When the tree has sufficient coverage of the planning problem, the algorithm traverses the tree to produce the final plan. Sampling based planners are continuous, probabilistically complete, *i.e.* find a solution with probability 1, and some are even *asymptotically optimal* [6, 7, 8], but under practical time limitations, their random nature can produce erratic—though valid—paths.

Both of the aforementioned planner types are designed to optimize for path length in the given state space (*e.g.* configuration space) while avoiding collisions. An optimal path in configuration space is not necessarily optimal for the end effector in cartesian space. Humans motion tends to minimize hand distance traveled [25], so what appears optimal for the algorithm may be unintuitive for a human partner or operator. In the manipulation domain, goals are typically represented in end effector task space [26, 27]. In a closed loop setting with a moving target, the traditional process of using IK to map task to configuration space can produce highly variable configurations, especially around obstacles. Motion Optimization [2, 3, 28] on the other hand, generates paths with non-linear optimization and can consider multiple objectives such as smoothness of the motion, obstacle avoidance and convergence to an end effector pose. These algorithms require careful tuning of the respective cost functions to ensure convergence to a desirable path and are prone to local minima. Furthermore, non-linear optimization is computationally complex and can be slow for difficult planning problems.

Local Control: In contrast to global planners, local controllers have long been applied to create collision-free motions [29, 30, 4, 11]. While they prioritize speed and smoothness, they are highly local and may fail to find a valid path in complex environments. We demonstrate in our experiments that MπNets are more effective at producing convergent motions in these types of environments, including in dynamic and in partially observed settings.

Imitation Learning: Imitation Learning [31] can train a policy from expert demonstrations with limited knowledge of the expert's internal model. For motion planning problems, we can apply imitation learning and leverage a traditional planner as the expert demonstrator—with perfect model of the scene during training—and learn a policy that forgoes the need for an explicit scene model at test time. Popular imitation learning methods include Inverse Reinforcement Learning [32, 33, 34] and Behavior Cloning [35, 36]. The former typically assumes expert optimality and learns a cost function accordingly, whereas the later directly learns the state-action mapping from demonstrations, regardless of the expert's optimality. We thus employ behavior cloning because producing optimal plans for continuous manipulation problems is challenging. Recent work demonstrates behavior cloning's efficacy for fine-grained manipulation tasks, such as chopstick use [37] and pick-and-place [38]. For long-horizon tasks like ours, however, distributional shift and data variance can hinder behavior cloning performance. Distribution shift during execution can lead to states unseen in training data [37]. Complex tasks often have a long tail of possible action states that are under-represented in the data, leading to high data variance [39]. There are many techniques to address these challenges through randomization, noise injection, regret optimization, and expert correction [37, 40, 41, 42, 43]. These techniques, however, have not been demonstrated on a problem of our scale and complexity (see Appendix D for details on the range of data). Our proposal seeks to overcome these issues by specifically designing a learnable expert, increasing the scale and variation of the data, and using a sufficiently expressive policy model.

Neural Motion Planning: Many deep planning methods [13, 44, 45, 46] seek to learn efficient samplers to speed up traditional planners. Motion Planning Networks (MPNets) [12] learn to directly plan through imitation of a standard sampling based RRT* planner [6] and is used in conjunction with a traditional planner for stronger guarantees. While these works greatly improve the speed of the planning search, they have the same requirements as a standard planning system: targets in configuration space and an explicit collision checker to connect the path. Our work operates based on task-space targets and perceptual observations from a depth sensor without explicit state estimation.

Novel architectures have been proposed, such as differentiable planning modules in Value Iteration Networks [20], transformers by Chaplot et al. [47] and goal-conditioned RL policies [48]. These methods are challenging to generalize to unknown environments or have only been shown in simple 2D [19] or 3D settings [21]. In contrast, we illustrate our approach in the challenging domain of controlling a 7 degrees of freedom (DOF) manipulator in unknown, dynamic environments.

3 Learning from Motion Planning

3.1 Problem Formulation

MπNets expect two inputs, a robot configuration q_t and a segmented, calibrated point cloud z_t. Before passing q_t through the network, we normalize each element to be within $[-1, 1]$ according to the limits for the corresponding joint. We call this $q_t^{\|\cdot\|}$. The point cloud is always assumed to be calibrated in the robot's base frame, and it encodes three segmentation classes: the robot's current geometry, the scene geometry, and the target pose. Targets are inserted into the point cloud via points sampled from the mesh of a floating end effector placed at the target pose.

The network produces a displacement within normalized configuration space $\dot{q}_t^{\|\cdot\|}$. To get the next predicted state $\hat{q}_{t+1}$, we take $q_t^{\|\cdot\|} + \dot{q}_t^{\|\cdot\|}$, clamp between $[-1, 1]$, and unnormalize. During training, we use $\hat{q}_{t+1}$ to compute the loss, and when executing, we use $\hat{q}_{t+1}$ as the next position target for the robot's low-level controller.

3.2 Model Architecture

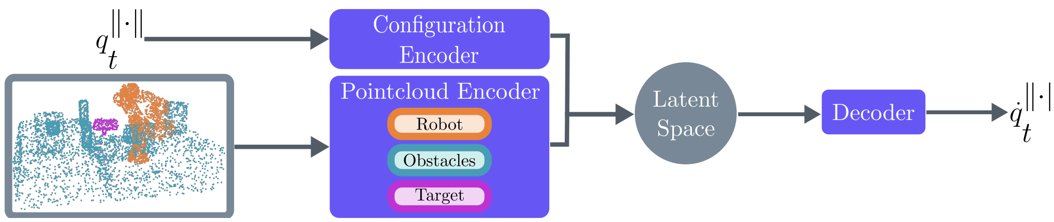

Figure 2: MπNets encodes state as a normalized robot configuration and segmented point cloud with three classes for the robot, the obstacles, and the target. The policy outputs a displacement in normalized joint space, which can then be applied to the input before unnormalizing to get q_{t+1}.

The network consists of two separate encoders, one for the point cloud and one for the robot's current configuration, as well as a decoder, totaling 19M parameters. Our neural policy architecture is visualized in Fig. 2. We use PointNet++ [49] for our point cloud encoder. PointNet++ learns a hierarchical point cloud representation and can encode a point cloud's 3D geometry, even with high variation in sampling density. PointNet++ architectures have been shown to be effective for a variety of point cloud processing tasks, such as segmentation [49], collision checking [18], and robotic grasping [50, 51]. The robot configuration encoder and the displacement decoder are both fully connected multilayer perceptrons. We discuss the architecture in detail in Appendix C. /fishyTalk about where partial observability comes from.

3.3 Loss Function

The network is trained with a compound loss function with two constituent parts: a behavior cloning loss to enforce accurate predictions and a collision loss to safeguard against catastrophic behavior.

Geometric Loss for Behavior Cloning To encourage alignment between the prediction and the expert, we compute a geometric loss across a set of 1,024 fixed points along the surface of the robot.

$$L_{\text{BC}}(\hat{\Delta} q_t) = \sum_i \|\hat{x}^i_{t+1} - x^i_{t+1}\|_2 + \|\hat{x}^i_{t+1} - x^i_{t+1}\|_1, \text{ where } \begin{matrix} \hat{x}^i_{t+1} = \phi^i(q_t + \hat{\Delta} q_t) \\ x^i_{t+1} = \phi^i(q_{t+1}) \end{matrix} \tag{1}$$

$\phi^i(\cdot)$ represents a forward kinematics mapping from the joint angles of the robot to point i defined on the robot's surface. The loss is computed as the sum of the $L1$ and $L2$ distances between corresponding points on the expert and the prediction after applying the predicted displacement. By using both $L1$ and $L2$, we are able to penalize both large and small deviations.

We use a geometric, task-space loss because our goal is to ensure task-space consistency of our policy. Configuration space loss appears in prior work [12], but does capture the accumulated error of the kinematic chain as effectively (see Appendix J).

Collision Loss In order to avoid collisions–a catastrophic failure–we apply an additional hinge-loss inspired by motion optimization [52].

$$L_{\text{collision}} = \sum_i \sum_j \|h_j(\hat{x}^i_{t+1})\|_2, \text{ where } h_j(\hat{x}^i_{t+1}) = \begin{cases} -D_j(\hat{x}^i_{t+1}), & \text{if } D_j(\hat{x}^i_{t+1}) \leq 0 \\ 0, & \text{if } D_j(\hat{x}^i_{t+1}) > 0 \end{cases} \tag{2}$$

The synthetic environments are fully-observable during training, giving us access to the signed-distance functions (SDF), $\{D_j(\cdot)\}_j$, of the obstacles in each scene. For a given closed surface, its SDF maps a point in Euclidean space to the minimum distance from the point to the surface. If the point is inside the surface, the function returns negative.

3.4 Training Implementation Details

MπNets is trained for single-step prediction, but during inference, we use it recursively for closed-loop rollouts. The compounded noise in subsequent inputs equates covariate shift [41, 43]. To promote robustness, we augment our training data with random perturbations in two ways. We apply Gaussian noise to the joint angles of each input configuration, which in turn affects the corresponding points in the point cloud, passed as input to the network [37, 53]. Additionally, for each training example, we generate a unique point cloud during training, *i.e.* during each epoch, the network sees 163.5M unique point clouds. We train our network with a single set of weights across our entire dataset.

4 Procedural Data Generation

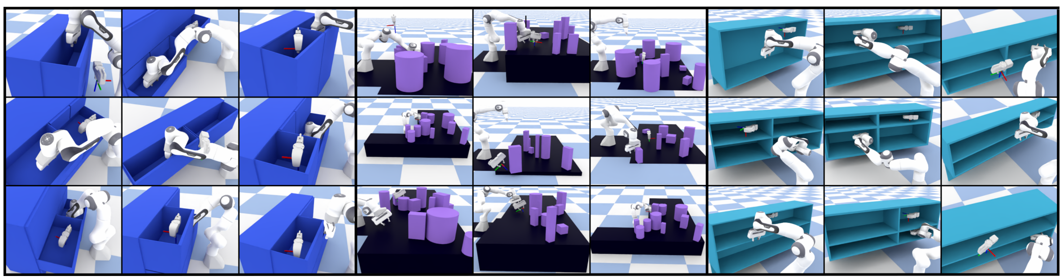

Figure 3: MπNets is trained with a dataset consisting of solutions to 3.27 million unique planning problems across over 575,000 unique, procedurally generated environments.

4.1 Large-scale Motion Planning Problems

Each planning problem is defined by three components: the scene geometry, the start configuration, and the goal pose. Our dataset consists of randomly generated problems across all three components, totaling 3.27 million problems in over $575,000$ environments. We have three classes of problems of increasing difficulty: a cluttered tabletop with randomly placed objects, cubbies and dressers. Representative examples of these environments are shown in Fig. 1. Once we build these environments, we generate a set of potential end-effector targets and corresponding inverse kinematics solutions. We then randomly choose pairs of these configurations and verify if a plan exists between them using our expert pipeline, as detailed further in Sec. 4.2 and in the Appendix D.

4.2 Expert Pipeline

Our expert pipeline is designed to produce high quality demonstrations we want to mimic, *i.e.* trajectories with smooth, consistent motion and short path lengths. Here, *consistency* is meant to describe quality and repeatability of an expert planner—see Appendix B for further discussion. We considered two candidates for the expert - the *Global Planner* which is a typical state-of-the-art configuration space planning pipeline [9] and a *Hybrid Planner* that we engineered specifically to generate consistent motion in task space. For both planners, we reject any trajectories that produce collisions, exceed the joint limits, exhibit erratic behavior (*i.e.* high jerk), or that have divergent motion (*i.e.* final task space pose is more than $5\mathrm{cm}$ from the target).

Global Planner consists of off-the-shelf components of a standard motion planning pipeline–inverse kinematics (IK) [54], configuration-space AIT* [9], and spline-based, collision-aware trajectory smoothing [55]. For a solvable problem, as the planning time approaches infinity, IK will find a valid solution and AIT* will produce an optimal collision-free path, both with probability 1. Likewise, with continuous collision checking, the smoother will produce a smooth, collision-free path. In practice, our dataset size goal—we generated 6.54M trajectories across over 773K environments—dictated our computation budget and tuned the algorithms according to this limit. We attempted IK at most 1,000 times, utilized an AIT* time out of 20s, and employed discrete collision checking when smoothing. Most commonly, the pipeline failed when AIT* timed out or when, close to obstacles, the smoother's discrete checker missed a collision, thereby creating invalid trajectories.

Hybrid Planner is designed to produce consistent motion in task space. The planner consists of task-space AIT* [9] and Geometric Fabrics [4]. AIT* produces an efficient end effector path and Geometric Fabrics produce geometrically consistent motion. The end effector paths acts as a dense sequence of waypoints for a sequence of Geometric Fabrics, but as the robot moves through the waypoints, the speed can vary. To promote smooth configuration space velocity over the final trajectory, we fit a spline to the path and retime it to have steady velocity. As we discuss in Sec. 5.1, Geometric Fabrics often fail to converge to a target, so we redefine the planning problem to have the same target as the final position of the trajectory produced by the expert. Inspired by [56], we call this technique *Hindsight Goal Revision (HGR)* and demonstrate its importance in Sec. 5.4. Using the *Hybrid Planner*, we generated 3.27 million trajectories across 576,532 environments.

5 Experimental Evaluation

We evaluate our method with problems generated from the same distribution as the training set. See Appendix for more detail on the procedural generation and random distribution. Within the test set, each problem has a unique, randomly generated environment, as well as a unique target and starting configuration. None of the test environments, starting configurations, nor goals were seen by the network during training. Our evaluations were performed on three test sets: a set of problems solvable by the *Global Planner*, problems solvable by the *Hybrid Planner*, and problems solvable by both. Each test set has 1,800 problems, with 600 in each of the three types of environments.

Quantitative Metrics: To understand the performance of a policy, we roll it out until it matches one of two termination conditions: 1) the Euclidean distance to the target is within 1cm or 2) the trajectory has been executed for 20 s (based on consultations with the authors of [4] and [11]). We consider the following metrics (see Appendix for details):

- *Success Rate* - A trajectory is considered a success if its final position and orientation target errors are below 1 cm and 15° respectively and there are no physical violations.
- *Time* - We measure the wall time for each *successful* trajectory. We also measure *Cold Start (CS) Time*, the average time to react to a new planning problem.
- *Rollout Target Error* - The L2 position and orientation error (taken from [57]) between the target and final end-effector pose in the trajectory.
- *Collision Rate* - The rate of fatal collisions, both self and scene collisions
- *Smoothness* - We use Spectral Arc Length (SPARC) [58] and consider a path to be smooth if its SPARC values in joint and end-effector space are below -1.6.

5.1 Comparison to Methods With Complete State

Most methods to generate motion in the literature assume access to complete state information in order to perform collision checks. In each of the following experiments, we provide each baseline method with an oracle collision checker. When running MπNets, we use a point cloud sampled uniformly from the surface of the entire scene. Results are shown in Table 1. /fishyTalk about which expert would be better where

	Soln. Time (s)	CS Time (s)	Success Rate (%) Global	Hybrid	Both	Smooth (%)
Global Planner [9]	16.46 ± 0.90	16.46 ± 0.90	100	78.44	100	51.00
Hybrid Planner	7.37 ± 2.23	7.37 ± 2.23	50.22	100	100	99.26
G. Fabrics [4]	0.15 ± 0.09	2.4e-4 $\pm$ 3e-5	38.44	59.33	60.06	85.39
STORM [11]	4.03 ± 1.89	13.4e-3 $\pm$ 2.2e-3	50.22	74.50	76.00	62.26
MPNets [12]						
Hybrid Expert	4.95 ± 23.51	4.95 ± 23.51	41.33	65.28	67.67	99.97
Random	0.31 ± 3.55	0.31 ± 3.55	32.89	55.33	58.17	99.96
MπNets (Ours)						
Global Expert	0.33 ± 0.08	6.8e-3 $\pm$ 7e-5	75.06	80.39	82.78	89.67
Hybrid Expert	0.33 ± 0.08	6.8e-3 $\pm$ 7e-5	75.78	95.33	95.06	93.81

Table 1: Algorithm performance on problems sets solvable by planner types. All prior methods use state-information and a oracle collision checker while MπNets only needs a point cloud

	Training Set	Evaluation Set MPNets-Style	Hybrid Expert Solvable (Ours)
MPNets [12]	MPNets-Style	78.70	49.89
MπNets (Ours)	MPNets-Style	33.70	5.50
MPNets [12]	Hybrid Expert	88.90	65.28
MπNets (Ours)	Hybrid Expert	89.50	95.33

Table 2: Success rates (%) of our method compared to Motion Planning Networks (MPNets) [12] trained and evaluated on different datasets

Global Configuration Space Planner The *Global Planner* is unmatched in its ability to reach a target, but this comes at the cost of average computation time (16.46s) compared to MπNets (0.33s). With a global planner, there is no option to partially solve a problem, meaning the Cold Start Time is equal to the planning time. In a real system, optimizers [2, 3, 10] could be used to quickly replan once an initial plan has been discovered. As discussed in Sec. 4.2, the *Global Planner* is theoretically complete, but fails in practice on some of the *Hybrid Planner*-solvable problems due to system timeouts and discrete collision checking during smoothing.

Hybrid End-Effector Space Planner Our *Hybrid Planner* struggles with a large proportion of problems solvable by the *Global Planner*. Yet, its solutions are both faster and smoother than the *Global Planner*. Surprisingly, MπNets trained with data from the expert *outperformed* the expert on the *Global Planner*-solvable test set. We attribute this to two features: 1) we use strict rejection sampling to reduce erratic and divergent behavior in our expert dataset and train only on the filtered data and 2) our use of Hindsight Goal Revision to turn an imperfect expert into a perfect one.

Neural Motion Planning Motion Planning Networks (MPNets) [12] proposed a similar method for neural motion planning, but there are a few key differences in both problem setup and system architecture. MPNets requires a ground-truth collision checker to connect sparse waypoints, plans in configuration space, and is not reactive to changing conditions. In the architecture, MPNets uses a trained neural sampler within a hierarchical bidirectional planner. The neural sampler is a fully-connected network that accepts the start, goal, and a flattened representation of the obstacle points as inputs and outputs a sample. MPNets guarantees completeness by using a traditional planner as a fallback if the neural sampler fails to produce a valid plan. /fishyPut in some info about why MPiNets score is so low on MPNets data

In addition to our data, we generated a set of tabletop problems, which we call *MPNets-Style*, akin to the Baxter experiments in [12], in order to fairly compare the two methods. The results of this experiment can be seen in Table 2. MπNets requires a large dataset that covers the space of test

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	8.61	0.11	0.44	69.89	75.17	83.44	85.11
STORM [11]	0.93	0.11	0.25	79.81	83.54	81.57	85.41
MπNets (Ours)							
Hybrid Expert	0.94	0.00	0.00	98.94	99.72	98.22	99.00
Global Expert	13.78	0.06	0.00	98.67	99.89	97.56	99.11

Table 3: Failure Modes on problems solvable by both the global and hybrid planners

problems to achieve compelling performance, while MPNets' utilization of a traditional planning system is much more effective with a small dataset or out of distribution problems. However, the MPNets architecture does not scale to more complex scenes, even with more data, as we show in Fig. 4. When trained and evaluated on the Hybrid Planner-solvable dataset, MPNets succeeds in 65.28% of the test set, whereas MπNets succeeds in 95.33%, thus decreasing the failure rate by 7X. Furthermore, as we show in Table 1, using the MPNets neural sampler trained with the *Hybrid Planner* performs similarly to a uniform random sampler when both are embedded within the bidirectional MPNets planner.

Local Task Space Controllers Unlike planners, which succeed or fail in binary fashion, local policies will produce individual actions that, when rolled out, may fail for various reasons. We break down the various failure modes across the set of problems solvable by both experts in Table 3.

STORM [11] and Geometric Fabrics [4] make local decisions that can lead them to diverge from the target in complex scenarios, such as cluttered environments or those with pockets. For example, both STORM and Geometric Fabrics struggle to retract from a drawer and then reach into another drawer in a single motion without intermediate waypoints. While STORM, Geometric Fabrics, and MπNets are all local policies, STORM and Geometric Fabrics rely on human tuning to achieve strong performance. Prior environment knowledge alongside expert tuning can lead to phenomenal results, but these parameter values do not generalize. We used a single set of parameters across all test environments just as we used a single set of weights for MπNets. MπNets encodes long-term planning information across a wide variety of environments, which makes it less prone to local minima, especially in unseen environments.

On problems solvable by the *Hybrid Planner*, MπNets ties or outperforms these other methods across nearly all metrics (see Table 4). On the set of problems solvable by the *Global Planner*, MπNets target convergence rate is consistently higher, while its collision rate (11%) is worse than either STORM (1.94%) or Geometric Fabrics (7.83%) (see Table 5). Deteriorating performance on out-of-distribution problems is a typical downside of a supervised learning approach such as MπNets. However, this could be improved with a more robust expert, *e.g.* one with the consistency of our *Hybrid Planner* but the success rate of the *Global Planner*, with finetuning, or with DAgger [40].

5.2 Importance of the Expert Pipeline

We observed that the choice of the expert pipeline affects the performance of MπNets. We trained three policies: MπNets-G with 6.54M demonstrations from the *Global Planner*, MπNets-H with 3.27M demonstrations from the *Hybrid Planner*, and MπNets-C with 3.27M demonstrations from each. MπNets-C did not exhibit improved performance over either MπNets-H or MπNets-G (see Appendix J for discussion). When evaluated on a test set of problems solvable by the *Global Planner*, MπNets-G shows far better target convergence (97.94% vs. 87.72%) compared to MπNets-H but worse obstacle avoidance (21.94% collision rate vs. 11%). Nonetheless, MπNets-H is significantly better across all metrics when evaluated on problems solved by both experts as shown in Table 3. We hypothesize that an expert combining the properties of these two–the consistency of the *Hybrid Planner* and the generality of the *Global Planner*, would further improve MπNets's performance. We refer to MπNets-H as MπNets throughout the rest of the paper.

5.3 Comparison to Methods With Partial Observations

In addition to demonstrating MπNets' performance on a real robot system, we also compared MπNets to the *Global Planner* (AIT* [9]) in a single-view depth camera setting in simulation. We evaluated on the test set of problems solvable by both the *Global* and *Hybrid Planners*. MπNets only has a minor drop in success rate when using a partial point cloud vs. a full point cloud– from 95.06% to 93.22% though the collision rate increases from 0.94% to 3.06% due to occlusions. For this experiment, we compared to the AIT* component of our *Global Planner* alone to minimize false-positive solutions caused by the smoother's discrete collision checker (see discussion in Section 4.2). We used a voxel-based reconstruction akin to the standard perception pipeline packaged with MoveIt [59]. In our implementation, a voxel is filled only if a 3D point is registered within it. On the same test set using the voxel representation, AIT* produces plans with collisions on 16.41% of problems. In this setting, MπNets's collision rate is over $5X$ smaller than that of the *Global Planner*.

5.4 Ablations

We perform several ablations to justify our design decisions. All ablations were trained using the *Hybrid Planner* dataset and evaluated on the *Hybrid Planner*-solvable test set. More ablations and details can be found in Appendix J.

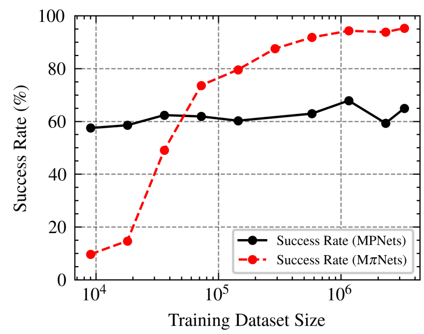

Figure 4: MπNets performance continues to increase with more training data, while MPNets performance stays relatively constant

MπNets Performance Scales with More Data As shown in Fig. 4, the performance of MπNets continues to improve with more data, although it saturates at 1.1M. Meanwhile, MPNets [12] has constant performance, demonstrating that our architecture is better able to scale with the data.

Robot Point Representation Improves Performance Instead of representing the robot by its configuration vector, we insert the robot point cloud at the specific configuration. Without this representation, the success rate decreases from 95.33% to 65.06%.

Hindsight Goal Revision Improves Convergence When trained without *HGR*, *i.e.* with the planner's original target given to the network, we see 58.11% success rate vs. 95.33% when trained with *HGR*. In particular, only 60.28% of trajectories get within 1cm of the target during evaluation.

Noise Injection Improves Robustness When we train MπNets without injecting noise into the input q_t, the policy performance decreases by 10.72%.

5.5 Dynamic Environments

MπNets is an instantaneous policy that assumes a static world at the time of inference. If the scene changes between inference steps, the policy will react accordingly. If the environment is continually changing–as is often the case in dynamic settings–MπNets implicitly approximates the dynamic movement as a sequence of static motions. When the scene changes are slow, this assumption works well. When the changes are fast, it does not. To demonstrate this, we evaluated MπNets in a static tabletop environment with a single, moving block placed on the table. We generated 1,000 planning problems across the table with the block placed at different locations. We specifically chose problems where MπNets succeeds when the block is stationary. When moving, the block follows a periodic curve in x and y, but the two curves have indivisible periods, preventing repetitive movement. We then moved the block at three different speeds: slow, medium, and fast and measured the success rate. At these speeds, MπNets succeeds 88.1%, 57.4%, and 28.3% respectively.

5.6 Real Robot Evaluation

We deployed MπNets on a 7-DOF Franka Emika Panda robot with an extrinsically calibrated Intel Realsense L515 RGB-D camera mounted next to it. Depth measurements belonging to the robot are removed and re-inserted using a 3D model of the robot before inference with MπNets. We created qualitative open-loop demonstrations in static environments and closed-loop demonstrations in dynamic ones. Rollouts are between 2 and 80 time steps long depending on the control loop frequency. See Appendix K for system details. Results can be viewed at https://mpinets.github.io and the attached video. As can be seen, MπNets can achieve *sim2real* transfer on noisy real-world point clouds in unknown and changing scenes.

6 Limitations

While MπNets can handle a large class of problems, they are ultimately limited by the quality of the expert supervisor and its need for a large, diverse dataset of training examples. Both generating the data and training MπNets is computationally intensive, requiring access to equipment that is both economically and environmentally expensive. It will also struggle to generalize to out-of-distribution settings typical of any supervised learning approach. When used on a real robot, performance will degrade as the robot's physical environment drifts from the training distribution. Likewise, performance will degrade with increasing point cloud noise. In future work we aim to improve MπNets with DAgger [40] or domain adaptation. In order to further enable safe operation in real robot systems, MπNets could also be combined with a ground-truth or learnt collision checker such as SceneCollisionNet [18]. In future work, we intend to investigate how to incorporate a learned safety component to detect out-of-distribution input data and prevent unsafe operation.

7 Conclusion

MπNets is a class of end-to-end neural policy policies that learn to navigate to pose targets in task space while avoiding obstacles. MπNets show robust, reactive performance on a real robot system using data from a single, static depth camera. We train MπNets with what is, as far as we are aware, the largest existing dataset of end-to-end motion for a robotic manipulator. Our experiments show that when applied to appropriate problems, MπNets are significantly faster than a global motion planner and more capable than prior neural planners and manually designed local control policies. We will release our code and data upon publication.

References

[1] S. M. LaValle. Planning algorithms. 2006.

[2] N. D. Ratliff, M. Zucker, J. A. Bagnell, and S. S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[3] J. Schulman, Y. Duan, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33:1251 – 1270, 2014.

[4] K. V. Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. N. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, B. Boots, and N. D. Ratliff. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics and Automation Letters*, 7:3202–3209, 2022.

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

[6] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30:846 – 894, 2011.

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074, 2015.

[8] M. P. Strub and J. D. Gammell. Advanced bit* (abit*): Sampling-based planning with advanced graph-search techniques. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 130–136, 2020.

[9] M. P. Strub and J. D. Gammell. Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3191–3198, 2020.

[10] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots. Continuous-time Gaussian process motion planning via probabilistic inference. volume 37, pages 1319–1340, 2018.

[11] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots. STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation. 2021.

[12] A. H. Qureshi, M. J. Bency, and M. C. Yip. Motion planning networks. *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124, 2019.

[13] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018.

[14] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust. Neural collision clearance estimator for batched motion planning. *arXiv preprint arXiv:1910.05917*, 2019.

[15] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[16] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[17] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[18] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox. Object rearrangement using learned implicit collision functions. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017, 2021.

[19] T. Jurgenson and A. Tamar. Harnessing Reinforcement Learning for Neural Motion Planning. In *Robotics Science and Systems*, 2019.

[20] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. In *Neural Information Processing Systems*, 2016.

[21] R. A. M. Strudel, R. G. Pinel, J. Carpentier, J.-P. Laumond, I. Laptev, and C. Schmid. Learning obstacle representations for neural motion planning. In *CoRL*, 2020.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968.

[23] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *NIPS*, 2003.

[24] M. Likhachev, D. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, 2005.

[25] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, 61:89–101, 1989.

[26] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox. Nerp: Neural rearrangement planning for unknown objects. *ArXiv*, abs/2106.01352, 2021.

[27] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444, 2021.

[28] N. D. Ratliff, M. Toussaint, and S. Schaal. Understanding the geometry of workspace obstacles in motion optimization. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4202–4209, 2015.

[29] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[30] N. D. Ratliff, J. Issac, and D. Kappler. Riemannian motion policies. *ArXiv*, abs/1801.02854, 2018.

[31] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Found. Trends Robotics*, 7:1–179, 2018.

[32] S. J. Russell. Learning agents for uncertain environments (extended abstract). In *COLT' 98*, 1998.

[33] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607072.

[34] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[35] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1988.

[36] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[37] L. Ke, J. Wang, T. Bhattacharjee, B. Boots, and S. S. Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6185–6191, 2021.

[38] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

[39] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9328–9337, 2019.

[40] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[41] S. Ross and A. Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, page 661–668, 2010.

[42] M. Laskey, J. N. Lee, R. Fox, A. D. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *CoRL*, 2017.

[43] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *arXiv preprint arXiv:1011.0686*, 2010.

[44] R. Kumar, A. Mandalika, S. Choudhury, and S. S. Srinivasa. Lego: Leveraging experience in roadmap generation for sampling-based planning. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1488–1495, 2019.

[45] C. Zhang, J. Huh, and D. D. Lee. Learning implicit sampling distributions for motion planning. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661, 2018.

[46] C. Chamzas, Z. K. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki. Learning sampling distributions using local 3d workspace decompositions for motion planning in high dimensions. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1283–1289, 2021.

[47] D. S. Chaplot, D. Pathak, and J. Malik. Differentiable spatial planning using transformers. In *Internal Conference on Machine Learning*, 2021.

[48] B. Eysenbach, R. Salakhutdinov, and S. Levine. C-learning: Learning to achieve goals via recursive. In *International Conference on Learning Representations*, 2021.

[49] C. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[50] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *International Conference on Computer Vision (ICCV)*, 2019.

[51] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox. 6-dof grasping for target-driven object manipulation in clutter. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[52] A. Fishman, C. Paxton, W. Yang, D. Fox, B. Boots, and N. D. Ratliff. Collaborative interaction models for optimized human-robot teamwork. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11221–11228, 2020.

[53] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. *Conference on Robot Learning*, pages 143–156, 2017.

[54] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[55] K. K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. *2010 IEEE International Conference on Robotics and Automation*, pages 2493–2498, 2010.

[56] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *NIPS*, 2017.

[57] P. Wunsch, S. Winkler, and G. Hirzinger. Real-time pose estimation of 3d objects from camera images using neural networks. *Proceedings of International Conference on Robotics and Automation*, 4:3232–3237 vol.4, 1997.

[58] S. Balasubramanian, A. Melendez-Calderon, A. Roby-Brami, and E. Burdet. On the analysis of movement smoothness. *Journal of NeuroEngineering and Rehabilitation*, 12, 2015.

[59] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[60] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi:10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[61] C. R. Garrett. Pybullet planning. https://pypi.org/project/pybullet-planning/, 2018.

[62] K. K. Hauser. Fast interpolation and time-optimization on implicit contact submanifolds. In *Robotics: Science and Systems*, 2013.

[63] I. Misra, C. L. Zitnick, M. Mitchell, and R. B. Girshick. Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2939, 2016.

[64] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. In *ECCV*, 2016.

[65] C. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[66] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[67] B. Çalli, A. Singh, A. Walsman, S. S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[68] T. Kunz and M. Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. *Robotics: Science and Systems VIII*, pages 1–8, 2012.

[69] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *arXiv:2103.00020*, 2021.

Appendix

A Failure Modes Across All Test Sets

In the main paper, we presented the breakdown of the failure modes on the set of problems solvable by both the global and hybrid planners. In this section we present the failure modes separately across the two test sets. The *Global Planner*-solvable test set is consistently the hardest for all methods, having the highest collision rates and target error. While STORM and Fabrics both see significant increases in target error, the change in collision rate is minor. When trained with the *Global Expert*, MπNets has the highest collision rate across all test sets, yet it also has the most consistent rollout accuracy. We attribute the collision rate to inconsistency in the *Global Planner's* motion and the rollout accuracy to the high coverage of the problem space. When evaluated on the *Global Planner*-solvable test set, MπNets trained with the *Hybrid Expert* also has its highest collision rate. We attribute this to distribution shift in the problem space.

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	8.17	0.00	0.39	68.56	73.33	82.06	84.00
STORM [11]	0.39	0.11	0.28	83.11	85.33	90.00	91.61
MπNets (Ours)							
Hybrid Expert	0.89	0.00	0.00	98.83	99.61	98.83	99.28
Global Expert	15.94	0.00	0.00	99.00	99.83	97.06	99.28

Table 4: Failure Modes on problems solvable by the hybrid planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	7.83	0.50	0.33	45.67	57.33	74.39	78.22
STORM [11]	1.94	0.11	0.28	71.33	78.22	64.44	72.67
MπNets (Ours)							
Hybrid Expert	11.00	0.78	0.00	87.72	93.17	84.56	88.56
Global Expert	21.94	0.00	0.00	97.94	99.50	96.56	99.22

Table 5: Failure Modes on problems solvable by the global planner

B Expert Pipelines

We present more details of our planning pipeline in this section.

Global Planner is composed of widely used off-the-shelf components. We first use inverse kinematics to convert our task space goals to configuration space, followed by AIT* [9] in configuration space, and finally, spline-based, collision-aware trajectory smoothing [55]. We use IKFast [54] for inverse kinematics, OMPL [60] for AIT*, and Pybullet Planning for the smoothing implementation [61]. To manage the compute load when generating a large dataset of trajectories, we employed a time-out with AIT* of 20 seconds.

Hybrid Expert is designed to produce consistent motion in task space. We start by using AIT* [9] with a 2 second timeout to plan for a floating end effector, *i.e.* one not attached to a robot arm, and then use Geometric Fabrics [4] to follow the path. Geometric Fabrics are deterministic and geometrically consistent [4] local controllers, but they struggle to solve the problems in our dataset without assistance from a global planner. Geometric Fabrics are highly local, and even with dense waypoints given by a global planner, they can run into local minima, which in turn generate trajectories with highly variable velocity. We use a combination of spline-based smoothing and downsampling [62] to create a consistent configuration space velocity profile across our dataset.

Consistency We use the term *consistency* to describe a qualitative characteristic of a planner and its learnability. Specifically, we use it to describe two quantities: 1) expert quality and 2) repeatability of the planner. Mandlekar et al. [38] demonstrate how Imitation Learning performance varies depending on expert quality. Among the metrics they use to describe expert quality, they demonstrate the importance of expert trajectory length. MπNets employs task-space goals, and the *Hybrid Planner* produces shorter task space paths. Across our test dataset of global and hybrid solvable problems, the *Hybrid Planner*'s end effector paths average 57cm ± 31cm and the total orientation distance traveled in 95° ± 52°. Meanwhile, the *Global Planner*'s paths average 61cm ± 39cm and 113° ± 55°, respectively.

Repeatable input-output datasets are important for deep learning systems. Prior works have shown the deep learning systems deteriorate or require more data when using noisy labels [63, 64]. Both the *Global Planner* and *Hybrid Planner* are sampling-based planners and do not produce repeatable paths by their very nature. Yet, the *Hybrid Planner* uses sampling to plan in a lower-dimensional state space—6D pose space—while the *Global Planner* samples in 7D configuration space. We use a naive sampler, so the lower dimensionality of the *Hybrid Planner's* sampler implies that it's typical convergence rate will be faster. After planning, the *Hybrid Planner* employs Geometric Fabrics [4] to follow the task-space trajectory. Geometric Fabrics are deterministic, which further promotes repeatability in the final, configuration space trajectories. Meanwhile, the *Global Planner* uses a randomized smoothing algorithm that is not deterministic. Taking these individual components together, we expect the *Hybrid Planner's* solutions on similar problem to be typically more alike than the *Global Planner's* solutions to the same problems.

C Network Architecture

Our PointNet++ architecture has three set abstraction groups followed by three fully connected layers. The first set abstraction layer performs iterative furthest point sampling to construct a set of 512 points, then it does a grouping query within 5cm of at most 128 points. Finally, there is a local PointNet [65] made up of layers of size 4, 64, 64, 64 respectively. The second set abstraction is lower resolution, sampling 128 furthest points and then grouping at most 128 points within a 30cm radius. The corresponding PointNet is made up of layers of size 64, 128, 128, and 256 respectively. Our third set abstraction layer skips the furthest point sampling, groups all points together, and uses a final PointNet with layers of size 256, 512, 512, $1{,}024$ respectively. Finally, after the set abstraction layers, we have three fully connected layers with $4{,}096$, $4{,}096$, and $2{,}048$ dimensions respectively. In between these layers, we use group norm and Leaky ReLU.

The output of our point cloud encoder is a $2{,}048$ dimensional embedding. The robot configuration encoder and the displacement decoder are both fully connected multilayer perceptrons with Leaky ReLU activation functions [66]. The robot configuration encoder maps our 7 dimensional input to a 64 dimensional output and has four hidden layers with 32, 64, 128, and 128 dimensions respectively. The displacement decoder maps the combined embeddings from the point cloud and robot configuration encoders, which together have $2{,}112$ dimensions, to the 7 dimensional normalized displacement space. The decoder has three hidden layers with 512, 256, and 128 dimensions respectively. Our entire architecture together has 19 million parameters.

D Data Generation Pipeline

We used the same procedural data generation pipeline to generate data for training as well as inference test problems. We will be releasing the code to generate the data alongside our generated data sets upon publication.

Tabletop The dimensions of the table, including height, are randomized, as well as whether the table has an L-bend around the robot. The table itself is always axis-oriented. Table height ranges from 0

to 40cm. Table edges are chosen independently, *e.g.* the maximum x value for a table is chosen from a uniform distribution, and the center of the tables is not fixed. The front table can range between 90 and 110cm deep and between 205 and 240cm wide. When there is an L-bend, the side table ranges from 90 to 247.5cm deep and 42.5 to 72.5cm wide. After generating the table, a random assortment of boxes and cylinders are placed on the table facing upward, *i.e.* cylinders are on their flat edge. There are between 3 and 15 objects in each scene. These objects are between 5 and 35cm tall. The side dimensions of the boxes, as well as the radius of the cylinders, are between 5 and 15cm.

Cubby The dimensions of the cubby, the wall-thickness, the number of cubbies, and orientation of the entire fixture are randomized. We start by constructing a two-by-two cubby and then modify it to randomize the number of cubby holes. The wall thickness is chosen to be between 1 and 2cm. Similar to the tabletop, cubby edges are chosen independently, which implicitly set the center. The overall fixture is ranges from 120 to 160cm wide, 20 to 35cm deep, and between 30 and 60cm tall. The horizontal and vertical center dividers are then placed randomly within a 20cm range. Finally, we apply a random yaw rotation of up to $40°$around the fixture's central axis. For roughly half of the cubby environments, we modify the cubby to reduce the number of cubby holes. To do this, we select two random, collision-free robot configurations in two separate cubby holes and then merge the cubby holes necessary to create a collision-free path between them.

Dresser The dimensions of the dresser, the placement of the drawers, and the orientation of the entire fixture are randomized. The dresser side walls, drawer side walls, and drawer faces are always 1, 1.9, and 0.4cm thick respectively. Unlike the other two environments, dimensions for the dresser are chosen randomly, as is the center point for the fixture. The dresser dimensions range from 80 to 120cm wide, 20 to 40cm deep, and 55 to 85cm tall. The dresser is always placed on the ground randomly in reachable space of the robot, with a random orientation around its central yaw axis. We next construct the drawers. We randomly choose a direction in which to split the dresser and then split it into two drawers. We perform this recursively within each drawer, stopping according to a decaying probability function. Finally, we open two drawers within reachable space.

Initial Configurations and Target Poses After generating a random fixture, we search for valid start and goal configurations. We first look for target poses with reasonable orientations–in a grasping pose for the tabletop, pointing approximately inward for a cubby, or pointing approximately downward in a drawer. We choose pairs of these targets, solve for a collision-free inverse kinematics solution for each target, and consider these configuration space solutions to be candidates for the start or end of a trajectory. We also add a set of collision-free neutral configurations to the mix. These neutral configurations are generated by adding uniform randomness to a seed neutral configuration. From this set of task-space targets and corresponding collision-free configuration space solutions, we select pairs to represent a single planning problem. For each pair selected, we use the *Global Planner* to verify that a smooth, collision-free planning solution exists.

E Training MπNets

We implemented MπNets in PyTorch and used the Adam optimizer with a learning rate of 0.0004. We trained it across 8 NVIDIA Tesla V100 GPUs for a week.

F Inference with MπNets

We used separate inference hardware for our simulated experiments and the hardware demonstrations For our simulated experiments, we use a desktop with CPU Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz, GPU NVIDIA A6000, and 64GB of RAM. For our hardware demonstrations, we used a desktop with CPU Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz, GPU NVIDIA Titan RTX, and 32GB of RAM.

G Quantitative Metrics

Success Rate A trajectory is considered a success if the rollout position and orientation target errors are below $1\ \mathrm{cm}$ and $15°$ respectively and there are no physical violations. To avoid erroneously passing a trajectory that ends on the wrong side of a narrow structure, we also ensure that the end effector is within the correct final volume and likewise avoids incorrect volumes. For the cubby and dresser environments, these volumes are individual cubbies or drawers.

Time After setting up each planning problem, we measured the wall time for each *successful* trajectory. We also measure *Cold Start (CS) Time*, the average time to react to a new planning problem. While both expert pipelines have to compute the entire path, the local controllers only need time to compute a single action. We only consider the cold-start time here, but if the new planning problem is sufficiently similar to a previous one–such as a minor change in the environment or target–a global planning system could employ an optimizer that can replan quickly [10].

Rollout Target Error We calculate both position and orientation errors from the target for the final end effector pose in the trajectory. We measure position error with Euclidean distance and orientation error with the metric described by Wunsch et al. [57].

Collisions A trajectory can have two types of fatal collisions–when the robot collides with itself or when the robot collides with the scene. When checking for collisions, we use an ensemble of collision checkers to ensure fairness. Collision checking varies across algorithmic implementations, *e.g.* our AIT* implementation uses meshes to check scene collisions, while STORM [11] and Geometric Fabrics [4] use a sphere-based approximation of the robot's geometry. A trajectory is only considered to be in collision if the entire ensemble agrees.

Smoothness We use Spectral Arc Length (SPARC) [58] to measure smoothness. Balasubramanian et al. [58] use a SPARC threshold of -1.6 as sufficiently smooth for reaching tasks. This measurement qualitatively describes the behavior of our benchmark algorithms well, so we used the same threshold for sufficiency. We therefore consider a path to be smooth if both its joint-space trajectory and end effector trajectory have SPARC values below -1.6.

H Local Policy Implementations

Both STORM [11] and Geometric Fabrics [4] require expert tuning to achieve compelling performance, and we worked closely with the authors of these papers to tune them as best as possible for our evaluation. We train a single network on all three environment types, so similarly use a single set of tuning parameters for each algorithm over the entire evaluation set.

I MPNets Implementation and Data

In the original paper, Qureshi et al. [12] trained MPNets for execution on the Baxter robot using a dataset of 10 different tabletop environments, each with 900 plans. Then, it was evaluated in the same environments using 100 unseen start and goal configurations in each. In total, their real-robot dataset was $10{,}000$ problems.

To compare fairly to MPNets, we generated an analogous set of $10{,}000$ problems within 10 tabletop environments, which we call the *MPNets-Style* dataset. We re-implemented the MPNets-algorithm based on their open source implementation at https://github.com/anthonysimeonov/baxter_mpnet_experiments.

After we trained our implementation of their model on the MPNets-Style data, it achieved a similar success rate as the one quoted in their paper for the Baxter experiments (78% vs. 85%). We attribute the performance difference to the increased complexity of our environments, which, unlike the original dataset, have varying table geometry in addition to object placement. In the original paper, they quote planning as taking 1 second on average. Our re-implementation took 2.47 seconds on average with a median of 0.02 seconds. Again, we attribute this difference to the increased complexity, given that the median time is so far below the mean. Just as they do in the open source implementation, we employ hierarchical re-planning, but we do not fall back to a traditional planner. If given access to a collision checker, both MπNets and MPNets can use a similar fallback to re-plan, thus achieving theoretically complete performance.

We used the same training setup described in Appendix E to train MPNets. When trained on the MπNets data set, *i.e.* 3.27M demonstrations from the*Hybrid Planner*, MPNets converged within 15 hours.

J Additional Experiments

Training with Mean Squared Error Loss Increases Collisions When trained with a loss of mean-squared-error in configuration space, MπNets has a similar success rate–94.56% vs. 95.33%–but scene collision rate is significantly higher at 2.39% vs 0.89%.

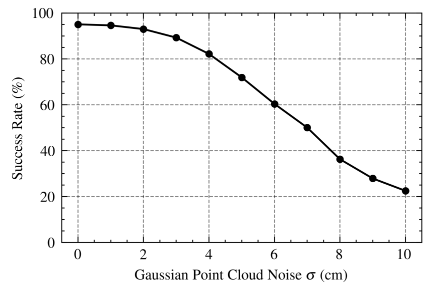

Figure 5: After injecting Gaussian noise into the point clouds, MπNets performance stays fairly constant up until $\sigma = 3$cm when success rate is 89.28%.

Representing the Target in Point Cloud Improves Performance When trained with the target fed explicitly through a separate MLP encoder as a position and quaternion, MπNets succeeds less–88.83% vs. 95.33% when the target is specified within the point cloud. In particular, only 91.61% of trajectories get within 1cm of the target vs. 98.83% with the point cloud-based target.

Training with Collision Loss Improves Collision Rate When trained without the collision loss, MπNets collides more often–2.11% vs 0.89% when trained with the collision loss.

Training with the Configuration Encoder Improves Success Rate When trained with no robot configuration encoder, *i.e.* with only the point cloud encoder, MπNets has success rate of 94.17% vs 95.33% when trained with both encoders.

MπNets is Robust to Point Cloud Noise Up to 3.2cm Figure 5 shows MπNets success rate on the set of problems solvable by both planners when random Gaussian noise is added to the point cloud. Model performance stays above 90% until noise reaches 3cm at which point success drops to 89.28%.

MπNets is Robust to Varying Point Cloud Shapes To evaluate performance in out-of-distribution geometries, we replaced all tabletop objects in test set of problems solvable by the *Hybrid Planner* with randomly meshes from the YCB dataset [67]. For each tabletop primitive, we sampled a mesh from the dataset and transformed it so that the bounding boxes of the primitive and mesh were aligned and of identical size. Note that in these modified scenes, the primitives-based *Hybrid Planner* solution is still valid. MπNets succeeded in 88.33% in this YCB-tabletop test set, whereas with the original primitives, it succeeds in 94.67%. Note that the network was not trained with these geometries—we would expect even higher performance if these meshes were included in the training set.

MπNets is Not Suitable for Unsolvable Problems To evaluate performance on unsolvable problems, we generated a set of 800 planning problems in randomized tabletops where the target is in collision with the table or an object on the table. When used for these problems, MπNets showed a 64.25% collision rate.

MπNets is Not Improved by Combining Experts We trained MπNets-C on a combination of 3.27M demonstrations each from the *Hybrid Planner* and *Global Planner*. Environments may have overlapped in these data sets, but entire problems, *i.e.* environment, start, and goal, did not. In problems solvable by the global planner, MπNets-C—like MπNets-G—outperformed MπNets-H in terms of target convergence (97.17% vs 87.72%). While its collision rate is lower than MπNets-G, (18.56% vs 21.94%) MπNets-C's collision rate is still significantly higher than MπNets-H (11%). The behavior of MπNets-C is essentially an average of MπNets-G and MπNets-H, which we attribute to the lack of easily learnable obstacle avoidance behavior by the *Global Planner*. These demonstrations equate to additional noise in the training data, which creates less successful obstacle avoidance behavior. In future work, we intend to explore how to robustly combine experts for improved performance.

K Real-World Experiments

We demonstrated MπNets in a variety of table top problems using a Franka Emika Panda 7-DOF manipulator. A calibrated Intel Realsense L515 RGB-D camera is placed in front of the robot's workspace, viewing the table and potential obstacles on top of it. Point cloud measurements are filtered to remove all points belonging to the robot geometry. The remaining cloud is downsampled to 4096 points and treated as the obstacle. The filtering process runs at 9 Hz. We investigated two different control methods:

Open-Loop Motion: Using a fixed, user-defined goal location and the current depth observation, MπNets is rolled out over 80 timesteps or until goal convergence. The resulting path is used to compute a time-parametrized trajectory [68] which is then tracked by a position controller. The videos listed under "Open Loop Demonstrations" at https://mpinets.github.io show a mix of sequential motions toward pre-defined goals. In some of the examples, the objects are static throughout the video and in others, we re-arrange the objects throughout the video. Despite the changing scene, these are still open-loop demos. While the motions adapt to changing obstacles in the scene, the policy only considers scene changes that happen before execution of a trajectory. This is because the point cloud observations are only updated once the robot reaches its previous target.

Closed-Loop Motion: To account for dynamic obstacles MπNets is rolled out for a single timestep at the same frequency as the point cloud filter operates (9 Hz). A time-parametrized trajectory is generated by linearly interpolating $\approx 70\%$ of the rolled out path. As in the open-loop case the resulting trajectory is tracked by a PD controller controller at 1 kHz. The videos listed under "Closed Loop, Dynamic Scene Demonstrations" at https://mpinets.github.io show examples of boxes thrown into the robot's path while it is moving towards a user-defined target. The evasive maneuver shows MπNets' ability to react to dynamic obstacles.

L Limitations

Training Distribution The limitations of the *Hybrid Planner* translate to limitations in the trained policy network. Certain target poses and starting configurations can create unanticipated behavior. When target poses are narrowly out of distribution, the rollout fails to converge to the target, but as a target poses drifts further from the training distribution, behavior becomes erratic. Likewise, random, initial configurations–such as from rejection-sampling based inverse kinematics–can create unexpected behavior, but we did not observe this in our real robot trials running the policy continuously to a sequence of points. With an improved expert, *e.g.* one with the consistency of our *Hybrid Expert* and guaranteed convergence of the *Global Planner*, we anticipate that the occurrence of failure cases will diminish. We also do not expect the network to generalize to wholly unseen geometries without more training data. But, in future work, we aim to improve the generalization of this method, much in the way that Large Language Models [69] continue to improve generalization through data.

Real Robot System In order to ensure safe operation in a real-robot system, MπNets could be combined with a collision checker–either one with ground-truth or a learned, such as Scene Collision

	Soln. Time (s)	Success Rate (%)			Smooth (%)
		Global	Hybrid	Both	
Global Planner [9]	16.56 ± 0.88	100	73.50	100	56.86
Hybrid Planner	6.82 ± 1.50	44.33	100	100	99.22
G. Fabrics [4]	0.11 ± 0.06	37.83	66.67	65.83	88.61
STORM [11]	3.65 ± 1.64	53.50	76.67	77.33	59.72
MPNets [12]					
Hybrid Expert	2.68 ± 17.39	44.67	59.00	66.17	17.26
Random	0.06 ± 0.06	32.17	50.17	53.67	100.00
MπNets (Ours)					
Hybrid Expert	0.33 ± 0.08	67.00	94.33	93.17	93.06
Global Expert	0.34 ± 0.07	74.83	81.50	80.00	93.44

Table 6: Algorithm performance on cubby problems sets solvable by planner types. All prior methods use state-information and a oracle collision checker while MπNets only needs a point cloud

Net [18]. The collision checker could be used to a) stop the robot before hitting collisions b) make small perturbations to nudge the policy back into distribution or c) enable a traditional planner to plan to the goal. In a physical system, not all problems will have feasible solutions. As discussed in Appendix J, MπNets will often collide in these scenarios, underscoring the need for some additional safety mechanisms to prevent catastrophic behavior. Additionally, MπNets has no concept of history and can collide with the scene if, for example, the robot arm blocks the camera mid-trajectory. To mitigate this, the perception system could employ a historical buffer or filter to maintain some memory of the scene.

Emergent Behavior In some of our test problems, we observed that MπNets produces a rollout where the final gripper orientation is $180°$ off from the target about the gripper's central axis (*i.e.* the central axis parallel to the fingers). In the test set of problems solvable by the *Global Planner*, this occurs in 2.44% of rollouts. We suspect this behavior is due to the near-symmetry in the gripper's mesh about this axis. The minor differences between the two sides of the gripper may not provide enough information for the Pointnet++ encoder to distinguish between these two orientations. While the rollout does not match the requested problem, this behavior can be desirable in some circumstances. For example, because grasps are symmetric with the Franka Panda gripper, a $180°$ rotation is preferable if it reduces the likelihood of a collision. For applications where this behavior is unacceptable, we could replace the target representation in the pointcloud with points sampled from a mesh with no symmetry.

M Experimental Results per Environment

In this section, we present the evaluation metrics broken down by environment type. However, we omit Cold Start Time because for global methods, it is the same as the total time and for local methods, the type of environment does not affect startup or reaction time.

The Tabletop environment is the least challenging with the highest success rates for all methods. In general, the dresser environment is the most challenging due to its complex geometry, as evidenced by the high collision rates. When trained with the *Hybrid Expert*, MπNets has the highest rollout target error in the cubby problems solvable by *Global Planner*. Since MπNets trained with the *Global Expert* does not have this issue, we attribute it to a lack of adequate coverage in the training dataset.

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within 1cm	5cm	15°	30°
G. Fabrics [4]	5.00	0.17	0.67	40.17	57.83	84.67	89.17
STORM [11]	0.50	0.00	0.50	79.33	85.33	69.17	80.33
MπNets (Ours)							
Hybrid Expert	10.67	0.17	0.00	75.83	84.50	75.83	81.67
Global Expert	23.17	0.00	0.00	99.17	100.00	99.33	100.00

Table 7: Failure Modes on cubby problems solvable by the global planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within 1cm	5cm	15°	30°
G. Fabrics [4]	4.83	0.00	1.00	72.50	83.00	95.83	96.33
STORM [11]	0.17	0.17	0.33	87.33	89.33	89.17	91.67
MπNets (Ours)							
Hybrid Expert	0.50	0.00	0.00	99.83	99.83	100.00	100.00
Global Expert	16.67	0.00	0.00	99.50	100.00	99.83	100.00

Table 8: Failure Modes on cubby problems solvable by the hybrid planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within 1cm	5cm	15°	30°
G. Fabrics [4]	5.00	0.00	1.17	72.33	84.33	96.33	97.33
STORM [11]	0.00	0.00	0.00	88.33	89.00	89.33	91.67
MπNets (Ours)							
Hybrid Expert	0.50	0.00	0.00	99.83	100.00	99.83	100.00
Global Expert	18.17	0.00	0.00	99.00	100.00	100.00	100.00

Table 9: Failure Modes on cubby problems solvable by both the global and hybrid planners

	Soln. Time (s)	Success Rate (%) Global	Hybrid	Both	Smooth (%)
Global Planner [9]	16.97 ± 0.81	100	66.83	100	75.63
Hybrid Planner	9.19 ± 2.81	37.33	100	100	99.82
G. Fabrics [4]	0.26 ± 0.12	15.00	25.83	28.50	78.94
STORM [11]	5.54 ± 1.84	24.17	58.50	62.00	83.22
MPNets [12]					
Hybrid Expert	15.55 ± 46.31	12.83	41.83	41.67	26.68
Random	1.61 ± 7.38	8.33	27.50	31.17	100.00
MπNets (Ours)					
Hybrid Expert	0.34 ± 0.06	78.67	97.00	96.33	91.56
Global Expert	0.33 ± 0.05	72.33	77.33	82.17	94.89

Table 10: Algorithm performance on dresser problems sets solvable by planner types. All prior methods use state-information and a oracle collision checker while MπNets only needs a point cloud

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	17.17	0.83	0.17	19.83	26.33	57.83	62.33
STORM [11]	4.83	0.17	0.33	42.67	51.67	45.17	53.83
MπNets (Ours)							
Hybrid Expert	17.00	0.83	0.00	98.00	98.67	93.50	94.33
Global Expert	26.67	0.00	0.00	100.00	100.00	99.00	99.83

Table 11: Failure Modes on dresser problems solvable by the global planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	18.33	0.00	0.17	36.00	39.00	61.00	66.00
STORM [11]	0.83	0.00	0.33	65.33	67.67	90.17	91.00
MπNets (Ours)							
Hybrid Expert	1.50	0.00	0.00	99.50	99.50	98.83	99.00
Global Expert	19.67	0.00	0.00	100.00	100.00	97.33	99.50

Table 12: Failure Modes on dresser problems solvable by the hybrid planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	19.50	0.33	0.17	40.00	42.67	64.50	68.17
STORM [11]	1.17	0.17	0.50	69.50	72.83	91.00	92.33
MπNets (Ours)							
Hybrid Expert	1.83	0.00	0.00	99.67	99.67	98.50	98.67
Global Expert	14.50	0.00	0.00	100.00	100.00	96.83	99.17

Table 13: Failure Modes on dresser-problems solvable by both the global and hybrid planners

	Soln. Time (s)	Success Rate (%)			Smooth (%)
		Global	Hybrid	Both	
Global Planner [9]	16.01 ± 0.74	100	95.00	100	28.27
Hybrid Planner	6.43 ± 1.18	69.00	96.33	100	100
G. Fabrics [4]	0.14 ± 0.07	62.50	85.50	85.83	88.61
STORM [11]	3.49 ± 1.65	73.00	88.33	88.67	43.83
MPNets [12]					
Hybrid Expert	1.36 ± 7.98	65.67	94.00	94.50	8.23
Random	0.05 ± 0.05	58.17	85.83	89.67	99.94
MπNets (Ours)					
Hybrid Expert	0.33 ± 0.10	81.67	94.67	95.67	96.83
Global Expert	0.33 ± 0.11	78.00	82.33	86.17	80.67

Table 14: Algorithm performance on tabletop problems sets solvable by planner types. All prior methods use state-information and a oracle collision checker while MπNets only needs a point cloud

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	1.33	0.50	0.17	77.00	87.83	80.67	83.17
STORM [11]	0.50	0.17	0.00	92.00	97.67	79.00	83.83
MπNets (Ours)							
Hybrid Expert	5.33	1.33	0.00	89.33	96.33	84.33	89.67
Global Expert	16.00	0.00	0.00	94.67	98.50	91.33	97.83

Table 15: Failure Modes on tabletop problems solvable by the global planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	1.33	0.00	0.00	97.17	98.00	89.33	89.67
STORM [11]	0.17	0.17	0.17	96.67	99.00	90.67	92.17
MπNets (Ours)							
Hybrid Expert	0.67	0.00	0.00	97.17	99.50	96.17	98.83
Global Expert	11.50	0.00	0.00	97.50	99.50	94.00	98.33

Table 16: Failure Modes on tabletop problems solvable by the hybrid planner

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [4]	1.33	0.00	0.00	97.33	98.50	89.50	89.83
STORM [11]	0.17	0.17	0.17	97.17	99.33	90.50	91.83
MπNets (Ours)							
Hybrid Expert	0.50	0.00	0.00	97.33	99.50	96.33	98.33
Global Expert	8.67	0.17	0.00	97.00	99.67	95.83	98.17

Table 17: Failure Modes on tabletop problems solvable by both the global and hybrid planners