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

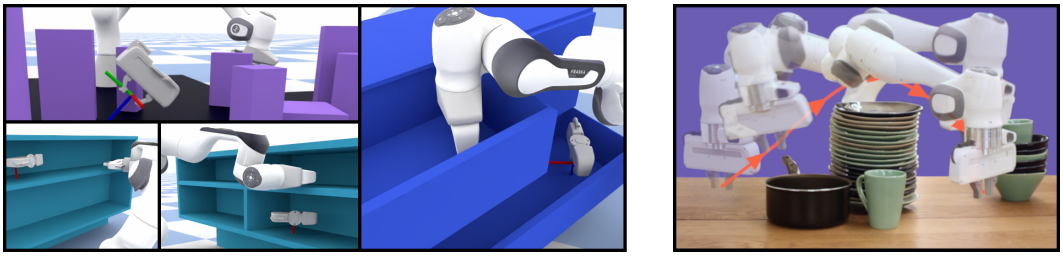

Figure 1: MπNets are trained on a large dataset of synthetic demonstrations (*left*) and can solve complex motion planning problems using raw point cloud observations (*right*).

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 and hence are not flexible enough to operate in novel scenes directly using raw sensor observations.

Global planners such as RRT [1] optimize for speed and completeness but fall short on optimality. Other sampling-based approaches choose to forgo speed for the sake of optimality [2, 3, 4, 5]. Similarly, optimization-based approaches [6, 7, 8] embrace locally optimal behavior in exchange for

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

completeness. Recent methods such as Geometric Fabrics [9] and STORM [10] 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 [11] 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 [12, 13], signed distance fields, etc. Reconstruction systems such as SLAM and KinectFusion [14] 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 [15] and NERF [16] are slow or do not yet generalize to novel scenes. Methods such as SceneCollisionNet [17] 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 [18, 19] or 3D [20] 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 [11].
- 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% better than prior work [11] 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* [21, 22, 23], discretize the state space and perform a graph search to find an optimal path. These algorithms are slow but find least-cost paths. Sampling-based planners [1] 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 typically much faster than search-based planners and some are even *asymptotically optimal* [2, 3, 4], but due to 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. However, optimal paths in configuration space can lead to unintuitive paths in a task space (*e.g.* Cartesian position of the robot's end effector). Motion Optimization [6, 7, 24] on the other hand, generates paths with non-linear optimization and considers 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 [25, 26, 9, 10]. 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.

Neural Motion Planning: Many deep planning methods [12, 27, 28, 29] seek to learn efficient samplers to speed up traditional planners. Motion Planning Networks (MPNets) [11] learn to directly plan through imitation of a standard sampling based RRT* planner [2] 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 [19], transformers by Chaplot et al. [30] and goal-conditioned RL policies [31]. These methods are challenging to generalize to unknown environments or have only been shown in simple 2D [18] or 3D settings [20]. 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

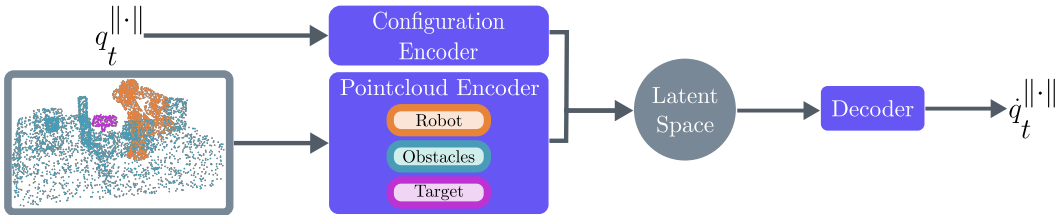

Figure 2: MπNets encodes state as a normalized robot configuration and segmented point cloud with three classes for the robot, the obstacles, and the target. The policy outputs a displacement in normalized joint space, which can then be applied to the input before unnormalizing to get q_{t+1}.

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++ [32] 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 [32], collision checking [17], and robotic grasping [33, 34]. The robot configuration encoder and the displacement decoder are both fully connected multilayer perceptrons. We discuss the architecture in detail in the Appendix.

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 many (1024) task spaces on 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 [11], but does capture the accumulated error of the kinematic chain as effectively (see Appendix).

Collision Loss In order to avoid collisions–a catastrophic failure–we apply an additional hinge-loss inspired by motion optimization [35].

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

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 only trained for single-step prediction, but we use it for closed-loop rollouts. The compounded noise in subsequent inputs equates covariate shift [36, 37]. 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 [38, 39]. We also sample novel scene point clouds on-the-fly during training. We train our network with a single set of weights across our entire dataset.

4 Procedural Data Generation

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.

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* describes how

the motion would change in response to a perturbation in any dimension of the planning problem. We considered two candidates for the expert - the *Global Planner* which is a typical state-of-the-art configuration space planning pipeline [5] 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 5cm from the target).

Global Planner consists of off-the-shelf components of a standard motion planning pipeline–inverse kinematics [40], configuration-space AIT* [5], and spline-based, collision-aware trajectory smoothing [41]. The *Global Planner* is theoretically complete and optimal, but in practice, we observed several common failures types. The planner can time out without finding the optimal path or worse, a solution at all. We utilize discrete collision checking for speed, which can cause the smoother to produce collisions when very close to obstacles. Despite these types of failures, we generated 6.54 million trajectories for training across $773{,}068$ environments.

Hybrid Planner is designed to produce consistent motion in task space. It consists of task-space AIT* [5] and Geometric Fabrics [9]. AIT* ensures a minimal end-effector path and Geometric Fabrics produce geometrically consistent motion. Finally, we fit a spline to the path and resample to create a consistent configuration space velocity profile. 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 [42], 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 and others using three test datasets: a set of problems solvable by the *Global Planner*, problems solvable by the *Hybrid Planner*, and problems solvable by both. Each dataset has $1{,}800$ problems, with 600 in each of the three types of environments. Each problem has a unique, randomly generated environment, as well as a unique target and starting configuration.

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 $20\,\text{s}$ (based on consultations with the authors of [9] and [10]). 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 $1\,\text{cm}$ 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 [43]) 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) [44] 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.

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).

	Soln. Time (s)	CS Time (s)	Success Rate (%)			Smooth (%)
			Global	Hybrid	Both	
Global Planner [5]	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 [9]	0.15 ± 0.09	$2.4e\text{-}4 \pm 3e\text{-}5$	38.44	59.33	60.06	85.39
STORM [10]	4.03 ± 1.89	$13.4e\text{-}3 \pm 2.2e\text{-}3$	50.22	74.50	76.00	62.26
MPNets [11]						
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	$6.8e\text{-}3 \pm 7e\text{-}5$	75.06	80.39	82.78	89.67
Hybrid Expert	0.33 ± 0.08	$6.8e\text{-}3 \pm 7e\text{-}5$	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

		Evaluation Set	
	Training Set	MPNets-Style	Hybrid Expert Solvable (Ours)
MPNets [11]	MPNets-Style	78.70	49.89
MπNets (Ours)	MPNets-Style	33.70	5.50
MPNets [11]	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) [11] trained and evaluated on different datasets

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 [6, 7, 8] 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) [11] 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.

In addition to our data, we generated a set of tabletop problems, which we call *MPNets-Style*, akin to the Baxter experiments in [11], in order to fairly compare the two methods. The results of this experiment can be seen in Table 2. MπNets requires a large dataset for compelling performance, while MPNets' utilization of a traditional planning system is much more effective with a small dataset. However, the MPNets architecture does not scale to more complex scenes, even with more data, as we show in Fig. 3. When trained and evaluated on the Hybrid Planner-solvable dataset,

	% Env. Coll.	% Self Coll.	% Jnt Viol.	% Within			
				1cm	5cm	15°	30°
G. Fabrics [9]	8.61	0.11	0.44	69.89	75.17	83.44	85.11
STORM [10]	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

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 [10] and Geometric Fabrics [9] 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. Since our network compresses long-term planning information into a policy, it is much less prone to local minima and can reach between drawers with no additional scene information.

On problems solvable by the *Hybrid Planner*, MπNets ties or outperforms these other methods across nearly all metrics. 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 (2.06%) or Geometric Fabrics (7.94%). 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 [45].

5.2 Importance of the Expert Pipeline

We observed that the choice of the expert pipeline affects the performance of MπNets. We trained two policies: MπNets-G with 6.54M demonstrations from the *Global Planner* and MπNets-H with 3.27M demonstrations from the *Hybrid Planner*. 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* [5]) 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. To evaluate AIT*, we omitted the smoothing step, which can cause erroneous collisions as discussed in Sec. 4.2. We used a voxel-based reconstruction akin to the standard perception pipeline packaged

with MoveIt [46]. On the same test set, 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 the Appendix.

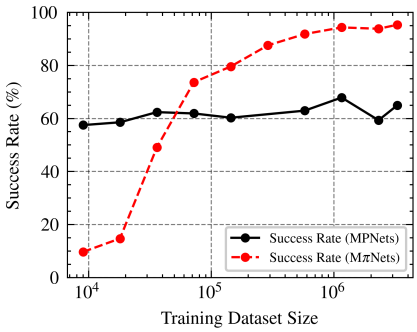

Figure 3: MπNets performance continues to increase with more training data, while MPNets performance stays relatively constant

MπNets Performance Scales with More Data As shown in Fig. 3, the performance of MπNets continues to improve with more data, although it saturates at 1.1M. Meanwhile, MPNets [11] 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 Real Robot Evaluation

We ran multiple qualitative experiments 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 before inference with MπNets. Rollouts are between 2 and 80 time steps long depending on the control loop frequency. Results can be viewed at https://mpinets.github.io and the attached video. As can be seen, MπNets 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. It will also struggle to generalize to out-of-distribution settings typical of any supervised learning approach. In future work we aim to improve MπNets with DAgger [45] 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 [17].

7 Conclusion

MπNets is a class of end-to-end neural molicy 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. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

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

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

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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.

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

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

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