Avoid Everything

Model-Free Collision Avoidance with Expert-Guided Fine-Tuning

Adam Fishman[1], Aaron Walsman[1], Mohak Bhardwaj[1], Wentao Yuan[1],
Balakumar Sundaralingam[2], Byron Boots[1], and Dieter Fox[1,2]

***Abstract*— The world is full of clutter. In order to operate effectively in uncontrolled, real world spaces, robots must navigate safely by executing tasks around obstacles while in proximity to hazards. Creating safe movement for robotic manipulators remains a long-standing challenge in robotics, particularly in environments with partial observability. In partially observed settings, classical techniques often fail. Learned end-to-end motion policies can infer correct solutions in these settings, but are as-yet unable to produce reliably safe movement when close to obstacles. In this work, we introduce *Avoid Everything*, a novel end-to-end system for generating collision-free motion toward a target, even targets close to obstacles. *Avoid Everything* consists of two parts: 1) *Motion Policy Transformer* (MπFormer), a transformer architecture for end-to-end joint space control from point clouds, trained on over $1{,}000{,}000$ expert trajectories and 2) a fine-tuning procedure we call *Refining on Optimized Policy Experts* (*ROPE*), which uses optimization to provide demonstrations of safe behavior in challenging states. With these techniques, we are able to successfully solve over 63% of reaching problems that caused the previous state of the art method to fail, resulting in an overall success rate of over 91% in challenging, partially observed manipulation settings.**

I. Introduction

The world is full of clutter. Humans effortlessly navigate through complex, unfamiliar spaces while constantly avoiding hazardous collisions. Robotics has not solved this key challenge, which is critical to real-world success of robotic actors [1]. Avoiding collision is one of the most important considerations in robot safety, and many important robotics settings such as kitchens, factories and warehouses are dynamic, restricted, and cluttered. For robotic arms, this problem is especially pronounced due to their complex kinematics (see Fig. 1).

However, many leading methods for collision avoidance rely on a stable, accurate, and fully observed representation of the robot's workspace. Traditional motion planning approaches [2], [3] attempt to explore free space to find a collision-free trajectory. Some of these techniques have rigorous theoretical guarantees that a plan (if one exists) will almost surely be found. Many strategies exist to find a valid path, often by searching through a predefined graph [4], [5] or by sampling the state space to incrementally build a tree [6]–[8]. These approaches can require hundreds or thousands of computationally expensive geometric collision checks based on an occupancy model, although there are many techniques to mitigate this disadvantage, *e.g.* lazy edge evaluation [9], [10] and parallelism [11]. Trajectory optimization seeks to find an ideal trajectory according to a set of objectives, typically using soft constraints defined by a differentiable environment model. While these methods can produce smooth motion quickly [12], [13], they require more information about an environment than typical motion planners, such as surface normals, to produce gradients. Whether these methods are using search, sampling, or optimization, they require an accurate world model that is either predefined or reconstructed from sensor data. When the world model is inaccurate, the planners may produce unsafe behavior. Furthermore, the real-time performance of these methods can be highly sensitive to the actual distribution of obstacles the robot encounters [14].

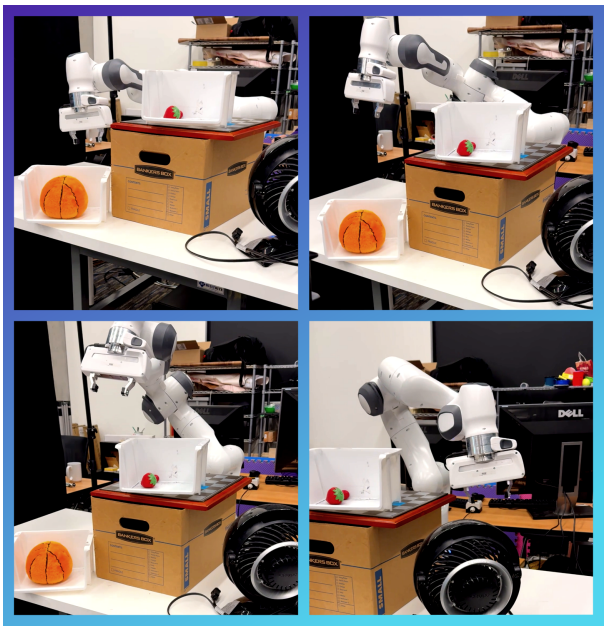

Fig. 1. *Avoid Everything* is able to generate collision-free trajectories around complex obstacles in real time, using input from a single depth camera.

Building an accurate world model, particularly in cluttered spaces, is an open problem [15]. End-to-end imitation learning is a popular technique that learns behavior without explicitly modeling the world, instead relying on patterns in how the expert behaves in response to the environment.

[1]University of Washington, {afishman, awalsman, mohakb, wentaoy, bboots, fox}@cs.washington.edu
[2]Nvidia, balakumars@nvidia.com

However, these methods face the challenge of learning to avoid collisions from collision-free demonstrations alone. To address this, traditional motion planners can be used to track the paths produced by the network [16], [17] or directly incorporate a predefined world model into the learning framework [18]. These systems have the same limitations as traditional ones. When the world model is inaccurate, the system may collide. While expert demonstrations can be made collision-free, learning collision avoidant behavior necessitates a deep understanding of the interplay between the scene geometry and the robot's kinematics. Successful approaches have employed large datasets [19], [20] or explicit losses to encourage obstacle avoidance [20]. However, these techniques still fail in complex problems, leading to constrained capabilities [20] or continued reliance on traditional collision checking techniques [18], [21].

To address these challenges, we present *Avoid Everything*, an end-to-end system that uses point clouds to generate goal-directed, collision-free motion for a robotic manipulator in cluttered 3D scenes. *Avoid Everything* uses a new network architecture *Motion Policy Transformer* (MπFormer) that is trained end-to-end using expert supervision from a motion planner. Our model predicts single-step changes in joint configuration using point cloud observations, the robot's current configuration, and a target end effector pose. We also introduce a fine-tuning approach inspired by hard negative mining [22], [23]: *Refining on Optimized Policy Experts* (*ROPE*). *ROPE* is critical to reducing the collision rate in reaching toward the target. Through experiments, we show that *Avoid Everything* is able to safely solve over 63% of problems where the previous state of the art method [20] fails, resulting in an overall success rate of over 91% in challenging, partially observed manipulation settings. We also demonstrate that *ROPE* can be used as a general tool to reduce collisions, even in conjunction with DAgger [24], a standard technique for improving imitation performance.

Our primary contributions are:

1) We present *Motion Policy Transformer*, a new model architecture designed for predicting goal-directed robot motion from a point cloud and target location.
2) We present *Refining on Optimized Policy Experts*, a novel fine-tuning algorithm for learning collision avoidance in robot motion generation.
3) We demonstrate empirically that *Avoid Everything* reduces the collision rate of the previous state of the art by over 77% and improves success rate by 63%.

II. Related Work

a) Reactive Control and Motion Planning: Robot motion generation has traditionally been studied in the context of motion planning with a vast literature of methods [25], [26] based on graph search [4], [5], [27], sampling-based motion planning [2], [6], [10], [28]–[30], and *trajectory optimization* [13], [31]–[33]. While modern motion planning frameworks can achieve low control latency [12], [34], [35], they assume complete knowledge of the environment and make strong assumptions about obstacle representations for fast collision checking. Perception-driven reactive control of robots also has a rich history. Operational Space Control (OSC) methods such as [36]–[38] can enable robots to perform highly dynamic tasks at high control frequencies. However, their myopic nature can lead to local minima in the presence of obstacles. In a similar spirit to our work, Model-Predictive Control (MPC) approaches [39], [40] try to balance reactivity and planning horizon; however, real-time requirements often warrant the use of simple obstacle representations and short horizons that can still lead to local minima.

b) Point Cloud Processing: Point clouds, unordered sets of 3D points, are a lightweight and convenient 3D representation. Unlike other 3D representations such as meshes or Signed Distance Functions (SDFs), it is much easier and faster to obtain 3D point clouds from sensors such as depth cameras. As a result, many recent works choose to infer semantics and affordance from point clouds directly, skipping the need for 3D reconstruction. Leveraging powerful neural backbones that process point sets [41], [42], existing networks can segment objects [41], plan grasps [43] and check collisions [44] from partial point clouds only. Following the same spirit, in this work we show how to reliably generate collision-free joint trajectories from raw 3D point clouds.

c) Imitation Learning: Imitation learning describes a broad class of techniques to learn a policy from demonstrations, often made by a privileged expert [45]. Among imitation learning techniques, behavior cloning [46], [47] describes a set of techniques where a policy is directly trained to mimic an expert's actions. In manipulation, these actions are often phrased as end effector waypoints [16], [17], [48], [49], but these methods require a separate planner and collision checker to perform tasks safely. Recently [50], [51] have demonstrated strong capabilities for using transformers [52] to solve complex manipulation tasks with images as input and joint controls as output. Inspired by these methods, our architecture produces joint space controls given point cloud input.

One common challenge with imitation learning is the problem of covariate shift. Learned policies typically have some small error in prediction. As the error accumulates, the policy will encounter unseen regions of the state space. While many techniques have been proposed to address this issue, a common strategy is to add a wider variety of possible states into the training dataset. This can be done with noise injection—Laskey et al. [53] use a purposefully noisy expert during data collection, while Ke et al. [54] use training-time noise injection to augment previously collected data. It can also be done by fine-tuning a pretrained policy. DAgger [24] uses the pretrained policy to augment the training data by providing expert demonstrations from states visited by the policy. A related technique can be found in computer vision called Hard Negative Mining [22], [23], which augments the training data by labeling the explicit failures (the *hard negatives*) from a pretrained model before either retraining or fine-tuning. Our technique draws inspiration from both

DAgger and Hard Negative Mining to explicitly correct the difficult states found from a pretrained model.

d) Learned Motion Planning: For the task of motion planning, imitation learning can be used either end-to-end or as a component of a traditional system. Some methods use learning to guide a traditional planner, either through a learned sampler [55]–[58] or a learned heuristic function [14], [59]. Other techniques [19], [40] rely on a learned collision model [60]. Motion Planning Networks [21] uses a point cloud neural network to generate waypoints that are then verified with a traditional collision checker. Saha et al. [18] uses a diffusion model to produce plans based on the the SDF representation of the environment. Our neural architecture is most similar to Motion Policy Networks (MπNets) [20], which expects a segmented, calibrated point cloud and produces joint space controls. Despite its strong performance on a variety of problems, MπNets is trained with an expert that is smooth but incapable of reaching close to obstacles. As we discuss in Section V-A.3, when the MπNets architecture is trained and evaluated on more challenging problems (using a more expressive expert), the policy often collides.

III. Methodology

In the following section, we describe our policy architecture, training implementation, and ***ROPE***, our fine-tuning strategy that introduces hard negatives and explicit corrections.

A. Behavior Cloning for Collision Avoidance

Avoid Everything is a single-step policy that takes in a point cloud of the scene P_t, a 6-DoF target end effector pose p, and the robot's joint configuration q_t, where t represents the current timestep. The scene point cloud P_t is augmented with points sampled from a mesh of the end effector placed at the target pose p and points sampled from the robot arm at joint configuration q_t, as shown in Fig. 2. This geometric representation of joint state q_t and target pose p has superior performance over a numerical representation [20]. The output of *Avoid Everything* at timestep t is a delta joint configuration Δq_t, which is added to the current joint state q_t to form a position target for the robot to follow.

1) Architecture: MπFormer uses PointNet++ [41] to encode the point cloud and a transformer [52] to fuse the point cloud features with a representation of the current joint state. The input point cloud has a feature vector of length 4 for every point. All obstacles are assigned the same feature, all target points are assigned the same feature, and each robot point, which are sampled deterministically from the robot's mesh, is assigned a unique feature to disambiguate points on the arm. Our PointNet++ encoding architecture consists of three Set Aggregation (SA) layers. SA layers are a sparse 3D analog to convolutional layers. Each layer receives a point cloud where each point has a feature and outputs a smaller point cloud by using furthest point sampling to select $\frac{1}{4}$ of the points. Then, each sampled point is used as the center of a ball query. The ball query samples up to 64 points inside the ball and concatenates the ball center's coordinates to each point's feature vector. A four-layer MLP is then run on each point and MaxPool [61] collects the points inside the ball to produce a single feature per ball. The layers' ball queries have radii of 5, 30, and 50 centimeters respectively. Our input point cloud always has $6{,}272$ points–$4{,}096$ obstacle points, $2{,}048$ robot points, 128 target points. The downsampled point cloud after the third set aggregation layer has 98 points. Finally, we add 3D positional encoding to each of these 98 points, similar to [43].

The transformer takes a sequence of tokens as input, consisting of the 98 output features of the third SA layer, a token for the current joint configuration, and a learned constant token, similar to the decoder tokens in [50]. We get the joint angle token by passing the joint angles, which are normalized to be between -1 and 1, through a single linear layer. Our transformer has 8 layers with an embedding dimension of 512 and a feed-forward dimension of $2{,}048$. To produce the final output Δq, we take the last token of the output sequence and map it through a single linear layer.

2) Loss Function: We train *Avoid Everything* according to the same loss functions as MπNets [20]: a task-space behavior cloning loss to encourage the policy to mimic the expert's behavior in task space, as well as a collision-avoidance loss. These losses are applied on predicted joint states, which are computed by adding the model's output (joint angle deltas) to the input joint angles and clamping the sum at the joint limits.

a) Task Space Loss: The aim of this loss is to compare the physical positions of the policy's predicted robot joint space configuration and the expert's joint space configuration. For both configurations, we use forward kinematic functions $\phi^{\{i\}}(\cdot)$ to map joint angles of the robot q to $1{,}024$ points $x^{\{i\}}$ on the robot's surface, represented in 3D coordinates.

$$L_{\text{BC}}(\hat{\Delta} q) = \sum_{i=0}^{1{,}024} \|\hat{x}^i - x^i\|_2 + \|\hat{x}^i - x^i\|_1 \qquad (1)$$

Like MπNets, we sum $L1$ and $L2$ distances in the loss because the sum penalizes both large and small errors. We use a task space loss following MπNets, which demonstrated it to be more effective when reasoning about collision avoidance as small perturbations along the kinematic chain can lead to large deviations for the end effector.

b) Collision Avoidance Loss: The training data was generated in simulation, giving us access to privileged information unavailable during inference, including a signed-distance representation of the scene. To avoid collisions, we use a hinge-based loss on $D(x)$, the signed distance from a point x on the robot to the nearest surface in the scene. Inspired by motion optimization [13], [31], [62], this loss effectively pushes the robot out of regions of collision. As in Equation 1, we use $1{,}024$ points $x^{\{i\}}$ on the robot's surface

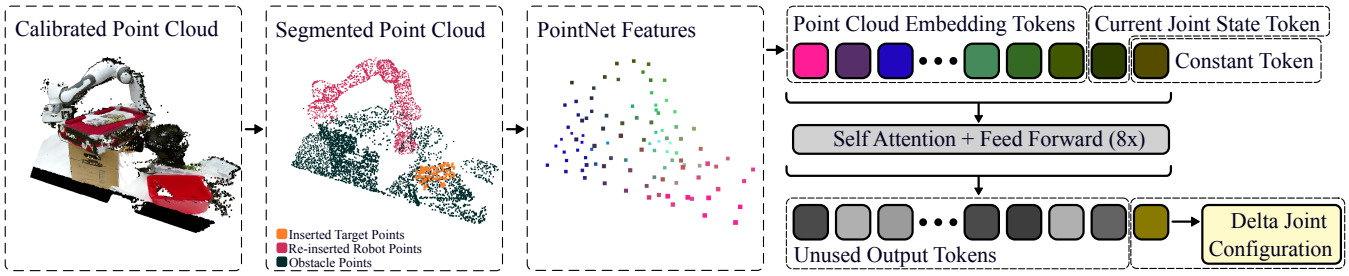

Fig. 2. The input to MπFormer is a labeled point cloud, consisting of 4,096 points from the depth image (with robot removed), 2,048 points sampled from the robot mesh at the current configuration and 128 points from the gripper mesh placed at the desired target. The point cloud is encoded with 3 Set Aggregation [41] layers. The resulting features, along with an encoding of the current joint state and a learned query token, are passed through 8 transformer layers. Finally, the output token that corresponds to the query token is decoded into a delta joint configuration.

to measure collision.

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

3) Training Implementation: *Avoid Everything* was trained on an NVIDIA 4090 in batches of 50 using AdamW [63] with a learning rate of 5e$-$5 and a linear warmup of 5000 steps from 1e$-$5. On the cubby environment, the model was trained for 1.2 million steps, which took approximately four days.

During training, we add small amounts of random noise to the input configurations, which [54] showed leads to improved robustness. Like MπNets, the training scenes are constructed from primitives, so point clouds can be generated on the fly during training by sampling points from the surfaces of these primitives. Robot points are sampled deterministically from the mesh of the robot. When *Avoid Everything* runs on the real robot, we mask out the robot points in the depth cloud and re-insert them using the same deterministically sampled points from training.

B. Expert-Guided Fine-Tuning

After pretraining on a large dataset of expert state-action pairs, we observe that the policy is highly capable of reaching the target pose. Despite the reaching success, however, it still collides with objects in a significant percentage of problems in the held-out validation set (see Section V-A.1). When we roll out the pretrained policy in simulation, we observe that the first obstacle penetrations are typically shallow and can be pulled out of collision by optimizing the configuration with respect to Equation 2. Based on this observation, we introduce a novel technique of refining the pretrained policy for improved collision avoidance using online fine tuning, inspired by Hard Negative Mining [22], [23] and trajectory optimization methods [13], [31], [63]. We outline our method, which we call *Refining on Optimized Policy Experts (ROPE)*, in Algorithm 1. During the refining stage, we take mini-batches of training data—the same data used for pretraining—and roll out trajectories for a fixed horizon. These trajectories can reach the target, collide, or neither. If the trajectory collides, we capture the state preceding the

Algorithm 1: Refining on Optimized Policy Experts

Result: π

$\pi \leftarrow \pi_{\text{pretrained}}$
$b \leftarrow$ Batch Size
$r \leftarrow$ Correction Ratio
D_{expert} ▷ Dataset containing expert demos
$B_{coll} \leftarrow \{\}$ ▷ Collision correction demos
$B_{free} \leftarrow \{\}$ ▷ Collision-free expert demos
for {state, next_state, tgt, scene} **in** D_{expert} **do**
  $s \leftarrow$ state
  for $j \leftarrow 1$ **to** N **do**
    $s' \leftarrow \pi(s, \text{tgt})$
    ▷ If s' collides, correct & add to buffer
    if COLLIDES(s', scene) **then**
      $\bar{s}' \leftarrow \text{CORRECT}(s', \text{scene})$ ▷ Eq. 2
      $\text{ADD}(B_{coll}, \{s, \bar{s}', \text{tgt}, \text{scene}\})$
      break
    end
    ▷ If rollout finishes w.o. collision, add orig. example to buffer
    if REACHED(s', tgt) **or** $j = N$ **then**
      $\text{ADD}(B_{free}, \{\text{state, next\_state, tgt, scene}\})$
      break
    end
    $s \leftarrow s'$
  end
  if $|B_{coll}| > rb$ **and** $|B_{free}| > (1-r)b$ **then**
    ▷ Make batch & clear buffers
    $B \leftarrow \{\text{POP}(B_{coll}, rb), \text{POP}(B_{free}, (1-r)b)\}$
    ▷ Compute loss, perform gradient update
    $\pi \leftarrow \text{UPDATE}(\pi, B)$
  end
  ▷ Reached validation accuracy or timeout
  if TERMINATION_CONDITION(π) **then**
    terminate
  end
end

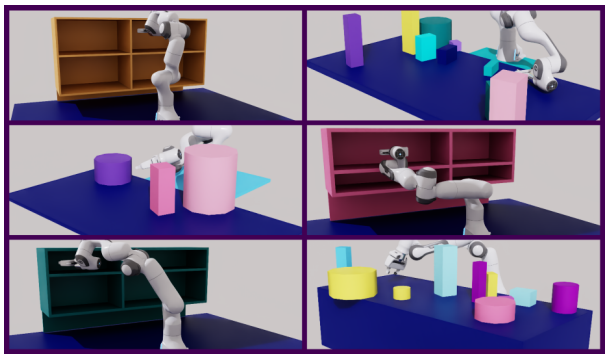

Fig. 3. *Avoid Everything* is trained separately in two classes of procedurally generated environments. The first class is a 2x2 cubby with random dimensions and positions. The second is a table of varying dimensions with 3 to 15 objects placed on it. We randomly sample start and goal poses and use inverse kinematics to produce joint configurations to match the poses. Our expert plans are generated in two steps: 1) run AIT* [8] with a configuration space path length objective to find a collision free path; 2) smooth the path with a spline-based shortcutting method [64] and resample the path with a fixed step interval.

collision as input and optimize the colliding configuration using Equation 2 to use as supervision and store this state-action pair in a buffer. If the trajectory does not collide, whether by successfully reaching the target or hitting the maximum rollout length, we use a separate buffer and store the input and output from the training batch, unmodified. In order to perform a weight update on the policy, we use a modified mini-batch made up of some proportion r of corrected examples and some proportion $1-r$ of unmodified expert examples. Once there are sufficiently many examples in both buffers, we remove these examples, assemble them into a modified mini-batch, and perform a weight update according to the losses used during pre-training. As discussed in Section V-A.5, increasing r leads to lower collision rate but poor target convergence.

During fine-tuning, we continually use the latest policy to perform rollouts, even as it is updated. We perform our optimization procedure using AdamW for simplicity, although we expect other methods typical to motion optimization such as Gauss-Newton or Levenberg-Marquardt may lead to a faster fine-tuning procedure. In our best-performing fine-tuning experiment, we reached peak performance after 21 hours of training.

IV. Data Generation Pipeline

We trained *Avoid Everything* on a large dataset of expert demonstrations in procedurally generated environments, examples of which are shown in Figure 3. The environments themselves were generated randomly and lie within two categories: 2x2 cubbies with randomized dimensions, cubby sizes, and world placement; and tabletops with a collection of randomly placed obstacles. These environments are similar to those demonstrated in MπNets, but they differ in two key ways: we augmented the cubby design to encourage reasonable expert behavior by adding a floor beneath the robot, and we increased the complexity of the tabletop environment by adding more objects and increasing the range of reachable poses. Within these constructed environments, we randomly sample end effector poses and their corresponding inverse kinematics (IK) solutions, which we compute using IKFast [65]. For the cubby environments, the poses are all grasping positions inside a cubby. For the tabletop, the poses are grasps pointing toward the lower hemisphere and placed either near the table's surface or on top of the objects. We also add neutral configurations drawn from uniform distribution around the robot's default pose to the tabletop data. These poses, for both types of environments, must be at least 5mm away from obstacles. We then use AIT* [8] with a path-length objective combined with a spline-based shortcutting [64] to generate expert demonstrations. In our planning pipeline, we impose a 20 second time limit in which we sample uniformly from the robot's configuration space, marking any sample that is either in self-collision or within 5mm of an obstacle as invalid. During the smoothing stage, we fit a collision and dynamics-aware spline to the planned path while shortcutting. We then sample from the spline at a fixed timestep, leading to paths with similar velocities, but varying lengths.

We chose this sampling pipeline because it enables us to produce expert demonstrations that lie precariously close to obstacles. Previously, MπNets [20] demonstrated strong performance when trained with a so-called *Hybrid Expert*, which uses a reactive controller [66] to follow a planned end effector path. While this expert is effective for learning, it is highly conservative, preferring to stay far away from obstacles. In their experiments, the authors demonstrated that the *hybrid expert* demonstrations are insufficient to learn to solve problems that lie very close to obstacles. With our sampling expert, we chose a 5mm buffer from obstacles because this is sufficiently close for most tasks. As we designed our expert, we observed that increasing the collision margin improves learned collision avoidance, but this limits the expert's ability (and thus, the policy's ability) to plan to targets near obstacles.

We trained *Avoid Everything* separately on each class of environments. For the cubby model, we used 1.25 million problems across $21{,}604$ environments. For the tabletop model, we used 2 million problems across $43{,}646$ environments. To generate this data, we used a single desktop with a AMD Ryzen Threadripper 3990X 64-Core Processor. Generating the cubby and tabletop data took four and six days respectively.

V. Experiments

In order to evaluate *Avoid Everything*'s performance, we used a mix of quantitative experiments in simulation and qualitative tests on physical hardware. Our simulated experiments are in environments drawn from the same distribution as our training data. However, there are no shared environments between the evaluation and training problem sets.

A. Simulated Experiments

Unless otherwise noted, the experiments described below are performed on partially observed point clouds generated

TABLE I

Avoid Everything COMPARED TO STATE-OF-THE-ART PATH PLANNERS UNDER PARTIAL OBSERVABILITY

Planner	Perception	Environment	
		Cubby	Tabletop
		SR (%) / SCR (%) / RSR (%)	SR (%) / SCR (%) / RSR (%)
Avoid Everything		95.51 / 2.09 / 99.00	91.89 / 3.68 / 98.11
MπFormer		87.39 / 10.51 / 99.46	83.13 / 15.18 / 99.52
MπFormer w. *ROPE*		91.39 / 5.06 / 97.96	86.97 / 8.11 / 96.54
MπNets		81.63 / 15.21 / 98.98	77.58 / 16.51 / 95.06
MπNets w. *ROPE*		88.76 / 5.44 / 96.50	82.55 / 9.37 / 92.21
RRTConnect	Octomap	32.68 / 67.16 / 99.52	46.52 / 53.30 / 99.62
cuRobo	NvBlox	73.06 / 22.88 / 94.74	76.86 / 22.11 / 98.68

with synthetic depth images. *Avoid Everything* is not trained with partially observed point clouds, but it is designed for use in scenarios with only partial observability.

For these evaluations, we used 5,000 problems in each of our cubby and tabletop environments (10,000 total). For each environment, we captured a synthetic depth image from a randomly positioned camera facing the scene. For each of these images, we placed the robot at a fixed neutral starting configuration and segmented the robot out of the image. To randomize the camera, it was first placed in the scene at a predefined location facing the robot and obstacles, and was then rotated randomly by up to 30° about the z-axis (rotating side to side), then again by up to 10° about the camera's local x-axis (tilting up and down). Both of these rotations were applied using a fixed pivot point directly in front of the camera. Finally, the camera was translated randomly along the global z axis and y axes by up to 25cm.

Tables I and II report three metrics. *Reaching Success Rate* (RSR) is the percentage of problems for which each method could provide a path (collision-free or not) to within 1cm and 15° of the goal. *Scene Collision Rate* (SCR) is the percentage of these paths that had a collision with the scene. *Success Rate* (SR) is the percentage of problems that had a collision-free solution to the goal, including self-collisions.

1) Avoid Everything: Avoid Everything is very robust to partially observed point clouds. Robustness to perspective changes and incompleteness is a well-understood property of PointNet [67]. As described in Section III-A.1, the PointNet layers use MaxPool [61] to aggregate data across spatial regions. MaxPool selects a single salient feature and will have a similar response given the presence of redundant information, which leads the PointNet to be highly robust to missing points. Without fine-tuning, MπFormer succeeds in 87.39% and 83.13% of our cubby and tabletop problems. However, after using DAgger and *ROPE*—the version we label *Avoid Everything* in Table I—we see it succeed in 95.51% and 91.89% in the cubby and tabletop settings respectively.

2) Classical Methods: While classical motion planners are highly capable of finding collision-free solutions, some even providing guarantees [70], this hinges on the ability to verify states with a good perceptual model. In practice, we have found that many of the perceptual models used in

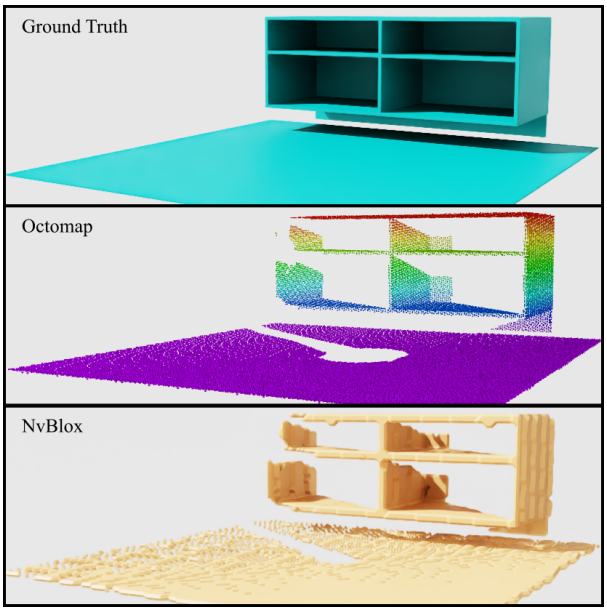

Fig. 4. A typical failure case for classical planners is that they do not account for collisions in unobserved regions. In this example, the reconstructions from both Octomap [68] and NvBlox [69] leave large holes due to occlusion. *Avoid Everything* is able to leverage learned priors to produce safe movement without an explicit reconstruction.

classical planning lead to erroneous solutions, *i.e.* solutions that the planner reported as valid when in fact they might have a collision. We evaluated this with two different styles of planners and their perceptual representations–see Figure 4 for examples of the perception. First, we evaluated with the commonly used motion planning library MoveIt! [71] paired with Octomap [68] for perception. We used an Octomap with a resolution of 5mm and RRTConnect [72] (with a 5s timeout) as the planner. In the cubby settings, we found that the planner found a solution in 99.52% of the problems and we attribute the remaining to noise that could be addressed with a longer timeout. However, of these successful plans, over 67% of them had collisions. We attribute this to the randomness in the path due to the sampling procedure, which leads the robot to move unnecessarily in unobserved space. RRTConnect produced fewer collisions (53%) in the tabletop setting, likely due to fewer or smaller holes in the point cloud. We ran a similar test using a trajectory optimization method designed to produce smooth trajectories, cuRobo [12] and NvBlox [69]. This technique finds a path in 94.74% of of cubby problems, but 22.88% of these trajectories have collisions. We set the nvBlox resolution to 1cm for this test after consulting with the authors of cuRobo [12]. While cuRobo also performed better in the tabletop setting, the difference was not as large as RRTConnect (see Table I). An advantage of these classical methods is that they did not require special tuning or training for either environment. Despite *Avoid Everything* having stronger performance in both environments, we do not expect it to generalize to wholly new settings as classical methods can.

TABLE II

Avoid Everything METRICS WITH VARYING REFINEMENT TECHNIQUES

F.T. Stage 1	F.T. Stage 2	Environment	
		Cubby	Tabletop
		SR (%) / SCR (%) / RSR (%)	SR (%) / SCR (%) / RSR (%)
No F.T.		87.39 / 10.58 / 99.46	83.13 / 15.18 / 99.52
ROPE		91.39 / 5.06 / 97.96	86.97 / 8.11 / 96.54
DAgger		91.23 / 7.30 / 99.57	88.21 / 8.63 / 99.12
DAgger	*ROPE*	93.92 / 3.64 / 99.41	89.97 / 5.67 / 97.94
Cons. DAgger		94.50 / 3.22 / 99.10	91.43 / 4.71 / 98.79
Cons. DAgger	*ROPE*	95.51 / 2.09 / 99.00	91.89 / 3.68 / 98.11

3) Motion Policy Networks: Our system design is most similar to MπNets, which is the state of the art for learned end-to-end collision free motion. In order to evaluate our method, we trained MπNets on our expert data and compared it to MπFormer without any fine-tuning. We also fine tuned both models using *ROPE* and compared the performance. These results are shown in Table I. Without any fine-tuning, we found MπFormer to outperform MπNets in both environments. We attribute this difference to the attention layer in our model, which maintains spatial structure of the point cloud, which MπNets flattens in its decoder. Additionally, we find that *ROPE* significantly improves the performance of both models, cutting collision rates approximately in half. However, after running *ROPE* on both algorithms, we find that the reaching success rate degrades more for MπNets through the fine-tuning process. MπFormer is better able to adapt to the hard negative examples without losing the ability to reach the target. Despite its improved success rate, *Avoid Everything* is less suitable for high-frequency control than MπNets. Running on a NVIDIA 4090 GPU, we can run *Avoid Everything* at 33Hz; meanwhile, MπNets runs at 150Hz, although we have observed faster speeds on updated hardware.

4) DAgger: One of the most common techniques for fine-tuning a learned policy is DAgger [24]. DAgger aids in accounting for distribution shift by asking the expert to provide demonstrations at every state the pretrained policy would visit. Likewise, *ROPE* can be seen as a way to account for distribution shift by correcting the policy when it fails. While DAgger is a generally useful tool for imitation learning, it requires making many costly calls to the expert. In our case, each expert demonstration requires 20 seconds of computation time, which adds up quickly if a demonstration is needed at every state visited by the policy. We implemented two versions of DAgger as comparisons and show the performance in Table 5. In the first version, we ran the pretrained *Avoid Everything* through its entire training data, collected the trajectories with collisions, and requested an expert demonstration at every step leading up the collision. We found that this technique can improve performance, reducing the pretrained collision rate of 10.58% in cubby setting to 7.30%, but it is not better than *ROPE*. We attribute this to the fact that the DAgger corrections use the same expert, which often veers very close (5mm) to obstacles. To verify this, we tested a second version of DAgger that uses a more conservative expert for corrections– one with a 2cm collision buffer. We label this technique *Cons. DAgger* in Table II. As discussed in Section IV, this expert is more limited in the problems it can solve, *e.g.* not those that either start or end within 2cm of obstacles. However, we found that this version of DAgger significantly improves collision avoidance without negatively impacting reaching performance, dropping collision rate in the cubby setting to 3.64%. We observe a similar drop in the tabletop setting, bringing pretrained collision rate from 15.18% to 4.71%. Collecting DAgger demonstrations for the policy's failures on our training dataset required nearly five days on a desktop with an NVIDIA 3090 GPU and an AMD Ryzen Threadripper 3990X 64-Core Processor.

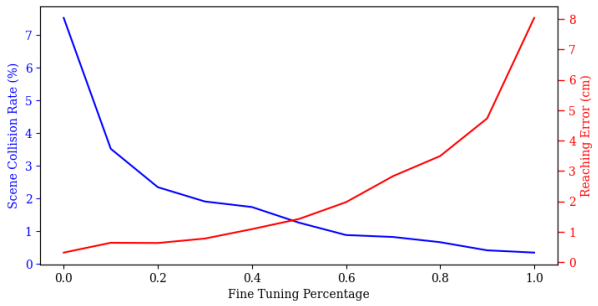

Fig. 5. Fine-tuning can be run with different proportions r of hard negative examples. As r increases, the collision rate goes down and target error increases. We attribute this phenomenon to the model overfitting to the hard negatives and forgetting the original behavior cloning objective.

When used alone, *ROPE* outperformed DAgger with the original 5mm expert in both the cubby and tabletop settings. Meanwhile, fine-tuning with *Cons. Dagger* outperforms both. However, we did not find *ROPE* to be to be mutually exclusive of DAgger. With both versions of DAgger, we were able to further improve performance by using *ROPE* as a second fine-tuning step. The best performance came from stacking the conservative DAgger technique with *ROPE*, with success rates of 95.51% and 91.81% in the cubby and tabletop settings respectively.

5) Balancing Collision Avoidance and Success Rate: We aimed to determine the efficacy of *ROPE* by varying the ratio of hard negative examples in each fine-tuning batch. We set this parameter r as a constant value for the entire fine-tuning procedure and studied how different values change the performance (see Figure 5). For these experiments, we looked only at the cubby setting and used fully observed point clouds, similar to those used during training. We evaluated 10,000 problems in unseen environments and observed a monotonic decrease in collision rate as r increased. However, we also observed a monotonic increase in the reaching error, *i.e.* the minimum distance from the target after rolling out for 70 time steps. With no fine-tuning, we measured an average reaching error of 0.32cm and a collision rate of 7.5%. At $r = 20\%$, we observe an average reaching error of 0.63cm with a collision rate of 2.3%. At $r = 60\%$, collision

rate is below 1%, but reaching error averages 1.97cm. We chose $r = 20\%$ for our other experiments, but the choice of this parameter should be determined by the downstream application and the criticality of collision avoidance. We did not experiment with varying r during fine-tuning as a function of performance, but we hypothesize that setting it as a function of performance would improve results.

B. Performance on Real Robot Hardware

We deployed *Avoid Everything* on a physical Franka Emika Panda robot using point clouds from a calibrated depth camera. We used a dual-computer setup running ROS. The control computer, which runs a real-time linux kernel, has Intel(R) Core(TM) i7-4770 CPU with 16 Gigabytes of RAM. The second computer, which runs *Avoid Everything*, has an Intel(R) Core(TM) i9-9900K CPU, 32 Gigabytes of RAM, and an NVIDIA Titan RTX GPU. We use a Kinect V2 for perception, which captures point clouds at approximately 10Hz. We use [73] for eye-on-hand calibration and [74] to remove the robot from the depth cloud; we then re-insert these robot points into the cloud using the deterministic sampling method described in Section III-A.3. We are able to run the model at approximately 25Hz on our hardware, which allows for reactive motion. We send each predicted action directly to a lower level joint controller [75].

The model is able to react to moving obstacles in the scene, but due to speed of our camera, it can take up to 140ms—100ms for the camera update, 40ms for the model update—for the robot to react to an obstacle. We expect that this reactivity could be improved with a faster camera, a faster GPU, or both. We used our best performing checkpoint, which was first fine-tuned with the conservative DAgger pipeline and then fine-tuned with *ROPE* (see Section V-A.4). We observed that the model is excellent at avoiding obstacles on the table when those obstacles are at least partially observable by the camera. We commonly saw collisions into obstacles that were fully occluded or out of the camera's field of view. We expect this issue could be improved with additional cameras to obtain a more complete point cloud. Many of the obstacles placed in front of the robot were far outside the training distribution, yet the model was able to avoid them easily. However, we found that highly complex obstacles, particularly those with thin structures (e.g. an office chair on its side) can result in collision. Not only was this obstacle out of distribution, but the rear legs were unobserved by the camera, leading to a compounding of our two main challenges.

We found signs of generalization as well as challenges with distribution shift. When we placed the target inside an obstacle, we observed that the model tends to hover above the obstacle without attempting to go in. This is despite the fact that none of the targets in training were ever inside obstacles. However, while this behavior occurred in the majority of cases, the robot did sometimes try to push through an obstacle to reach a target, especially when the obstructing face of the obstacle was obscured from the camera. For example, the top side of a tall box might not be visible by the Kinect, which leads the robot to attempt to push through the top to reach a target placed inside the box. Additionally, the gripper of the Franka is nearly symmetric about the axis that points from the wrist to the midpoint of the fingers. Our training data consisted of randomly generated poses, but these poses typically sampled from only half of the rotations about this axis. When we provided an out-of-distribution pose where the 180° rotation about this axis would be in distribution, we observed the robot typically tries to exploit the symmetry of the gripper and reach the symmetric in-distribution pose. Depending on the application, these 180° rotations may or may not be acceptable. We believe this could be fixed by increasing the variation of target poses in the training set, adding a unique per-point embedding to the gripper points to distinguish orientations, or both.

VI. Limitations

While *Avoid Everything* can be trained to have extremely low collisions in complex environments, there are open challenges. First and foremost is the problem of generalization. *Avoid Everything* performs well for in-distribution tasks, but we do not expect it to perform well in obstacle configurations that are far beyond anything seen during training or fine tuning. Likewise, we would expect a high reaching error for target poses that lie well beyond the training distribution. Second, we used a simple, gradient-based optimization to provide corrections during fine-tuning. For particularly complex environments, providing adequate corrections may require more sophisticated techniques, *e.g.* those used in [12]. Additionally, like other black box learned systems [20], *Avoid Everything* provides no guarantees. Future work could combine *Avoid Everything* with a traditional planner in fully observed settings, similar to [21]. Finally, *Avoid Everything* requires a significant amount of data and compute to train, which can be expensive and environmentally harmful.

VII. Conclusion

Avoid Everything is an end-to-end system that can create safe, collision-free motion toward a goal using only a partially observed point cloud. The system consists of two novel components, MπFormer and *ROPE*. MπFormer is an end-to-end transformer architecture that produces joint space controls toward a target. With no fine-tuning, MπFormer is significantly better than MπNets, the existing state of the art method for end-to-end collision-avoidant motion generation. *ROPE* is a fine-tuning technique used to improve performance by leveraging optimization to correct states where the pretrained policy collides. While we find that *ROPE* can be used to substantially improve the performance of MπFormer, we also found that it can be used to improve MπNets as well. When MπFormer and *ROPE* are used together as *Avoid Everything*, we find that the result is markedly more capable at generating collision-free reaching motion to a goal in partially observed settings than other techniques.

REFERENCES

[1] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, no. 1, p. null, 2024. [Online]. Available: https://doi.org/10.1146/annurev-control-061623-094742

[2] S. M. LaValle, J. J. Kuffner, B. Donald *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001. [Online]. Available: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf

[3] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 625–632. [Online]. Available: https://ieeexplore.ieee.org/document/5152399

[4] 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.*, vol. 4, pp. 100–107, 1968. [Online]. Available: https://ieeexplore.ieee.org/document/4082128

[5] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *NIPS*, 2003. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2003/file/ee8fe9093fbbb687bef15a38facc44d2-Paper.pdf

[6] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998. [Online]. Available: http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846 – 894, 2011. [Online]. Available: https://arxiv.org/abs/1105.1186

[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)*, pp. 130–136, 2020. [Online]. Available: https://arxiv.org/abs/2002.06589

[9] C. Dellin and S. Srinivasa, "A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 2016, pp. 459–467. [Online]. Available: https://arxiv.org/abs/1603.03490

[10] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020. [Online]. Available: https://arxiv.org/abs/1707.01888

[11] W. B. Thomason, Z. K. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," *ArXiv*, vol. abs/2309.14545, 2023. [Online]. Available: https://arxiv.org/abs/2309.14545

[12] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos *et al.*, "Curobo: Parallelized collision-free minimum-jerk robot motion generation," *arXiv preprint arXiv:2310.17274*, 2023. [Online]. Available: https://arxiv.org/abs/2310.17274

[13] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs." [Online]. Available: https://www.researchgate.net/publication/304532888_Motion_Planning_as_Probabilistic_Inference_using_Gaussian_Processes_and_Factor_Graphs

[14] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. Scherer, and D. Dey, "Data-driven planning via imitation learning," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1632–1672, 2018. [Online]. Available: https://arxiv.org/abs/1711.06391

[15] S. Garg, N. Sünderhauf, F. Dayoub, D. Morrison, A. Cosgun, G. Carneiro, Q. Wu, T.-J. Chin, I. Reid, S. Gould, P. Corke, and M. Milford, "Semantics for robotic mapping, perception and interaction: A survey," *Foundations and Trends® in Robotics*, vol. 8, no. 1–2, pp. 1–224, 2020. [Online]. Available: http://dx.doi.org/10.1561/2300000059

[16] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *ArXiv*, vol. abs/2212.06817, 2022. [Online]. Available: https://arxiv.org/abs/2212.06817

[17] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*, 2022. [Online]. Available: https://arxiv.org/abs/2209.05451

[18] K. Saha, V. R. Mandadi, J. Reddy, A. Srikanth, A. Agarwal, B. Sen, A. Singh, and M. Krishna, "Edmp: Ensemble-of-costs-guided diffusion for motion planning," *ArXiv*, vol. abs/2309.11414, 2023. [Online]. Available: https://arxiv.org/abs/2309.11414

[19] A. Murali, A. Mousavian, C. Eppner, A. Fishman, and D. Fox, "Cabinet: Scaling neural collision detection for object rearrangement with procedural scene generation," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1866–1874, 2023. [Online]. Available: https://arxiv.org/abs/2304.09302

[20] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox, "Motion policy networks," in *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022. [Online]. Available: https://arxiv.org/abs/2210.12209

[21] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion planning networks," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2118–2124, 2019. [Online]. Available: https://arxiv.org/abs/1806.05767

[22] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1627–1645, 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5255236

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, 2005. [Online]. Available: https://ieeexplore.ieee.org/document/1467360

[24] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2010. [Online]. Available: https://arxiv.org/abs/1011.0686

[25] S. LaValle, "Planning algorithms," *Cambridge University Press google schola*, vol. 2, pp. 3671–3678, 2006. [Online]. Available: https://lavalle.pl/planning/

[26] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005. [Online]. Available: https://biorobotics.ri.cmu.edu/book/

[27] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437020800060X

[28] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. [Online]. Available: https://ieeexplore.ieee.org/document/508439

[29] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 521–528. [Online]. Available: https://ieeexplore.ieee.org/document/844107

[30] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015. [Online]. Available: https://arxiv.org/abs/1306.3532

[31] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 489–494. [Online]. Available: https://ieeexplore.ieee.org/document/5152817

[32] K. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, vol. 27, no. 3, pp. 321–330, 2009. [Online]. Available: https://www.cambridge.org/core/journals/robotica/article/abs/using-optimization-to-create-selfstable-humanlike-running/855871E6530CCB6CA4AB1DADC4CB0DDE

[33] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 295–302. [Online]. Available: https://ieeexplore.ieee.org/document/7041375

[34] J. Schulman, Y. Duan, J. Ho, A. 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, vol. 33, no. 9, pp. 1251–1270, 2014. [Online]. Available: https://dl.acm.org/doi/10.1177/0278364914528132

[35] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1477–1496, 2016.

[36] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018. [Online]. Available: https://arxiv.org/abs/1801.02854

[37] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "Rmp flow: A computational graph for automatic motion policy generation," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 441–457. [Online]. Available: https://arxiv.org/abs/1811.07049

[38] C. D. Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3359–3365. [Online]. Available: https://ieeexplore.ieee.org/document/8206174

[39] J. Jankowski, L. Brudermüller, N. Hawes, and S. Calinon, "Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 125–10 131. [Online]. Available: https://arxiv.org/abs/2210.04067

[40] 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," in *Conference on Robot Learning*. PMLR, 2022, pp. 750–759. [Online]. Available: https://arxiv.org/abs/2104.13542

[41] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.02413

[42] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, and J. Zhou, "Pointr: Diverse point cloud completion with geometry-aware transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 498–12 507. [Online]. Available: https://arxiv.org/abs/2108.08839

[43] W. Yuan, A. Murali, A. Mousavian, and D. Fox, "M2t2: Multi-task masked transformer for object-centric pick and place," *arXiv preprint arXiv:2311.00926*, 2023. [Online]. Available: https://arxiv.org/abs/2311.00926

[44] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-dof grasping for target-driven object manipulation in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6232–6238. [Online]. Available: https://arxiv.org/abs/1912.03628

[45] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Found. Trends Robotics*, vol. 7, pp. 1–179, 2018. [Online]. Available: https://arxiv.org/abs/1811.06711

[46] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NIPS*, 1988. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

[47] M. Bain and C. Sammut, "A framework for behavioural cloning," in *Machine Intelligence 15*, 1995. [Online]. Available: https://dl.acm.org/doi/10.5555/647636.733043

[48] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, K. Choromanski, T. Ding *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *ArXiv*, vol. abs/2307.15818, 2023. [Online]. Available: https://arxiv.org/abs/2307.15818

[49] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees *et al.*, "Octo: An open-source generalist robot policy," https://octo-models.github.io, 2023. [Online]. Available: https://octo-models.github.io/

[50] T. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *ArXiv*, vol. abs/2304.13705, 2023. [Online]. Available: https://arxiv.org/abs/2304.13705

[51] H. Bharadhwaj, J. Vakil, M. Sharma, A. Gupta, S. Tulsiani, and V. Kumar, "Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking," 2023. [Online]. Available: https://arxiv.org/abs/2309.01918

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[53] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *Conference on Robot Learning*, 2017. [Online]. Available: https://arxiv.org/abs/1703.09327

[54] 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)*, pp. 6185–6191, 2021. [Online]. Available: https://arxiv.org/abs/2011.06719

[55] 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)*, pp. 1488–1495, 2019. [Online]. Available: https://arxiv.org/abs/1907.09574

[56] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7087–7094, 2018. [Online]. Available: https://arxiv.org/abs/1709.05448

[57] 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)*, pp. 1283–1289, 2021. [Online]. Available: https://arxiv.org/abs/2010.15335

[58] 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)*, pp. 3654–3661, 2018. [Online]. Available: https://arxiv.org/abs/1806.01968

[59] M. Bhardwaj, S. Choudhury, and S. A. Scherer, "Learning heuristic search via imitation," in *Conference on Robot Learning*, 2017. [Online]. Available: https://arxiv.org/abs/1707.03034

[60] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2021. [Online]. Available: https://arxiv.org/abs/2011.10726

[61] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, 2012. [Online]. Available: https://arxiv.org/abs/1202.2745

[62] 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)*, pp. 11 221–11 228, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9341369

[63] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017. [Online]. Available: https://arxiv.org/abs/1711.05101

[64] 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*, pp. 2493–2498, 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5509683

[65] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[66] K. V. Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. N. Peele *et al.*, "Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior," *IEEE Robotics and Automation Letters*, vol. 7, pp. 3202–3209, 2022. [Online]. Available: https://arxiv.org/abs/2109.10443

[67] 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)*, pp. 77–85, 2017. [Online]. Available: https://arxiv.org/abs/1612.00593

[68] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189 – 206, 2013. [Online]. Available: https://link.springer.com/article/10.1007/s10514-012-9321-0

[69] A. Millane, H. Oleynikova, E. Wirbel, R. Steiner, V. Ramasamy, D. Tingdahl, and R. Siegwart, "nvblox: Gpu-accelerated incremental

signed distance field mapping," 2023. [Online]. Available: https://arxiv.org/abs/2311.00626

[70] K. Goldberg, "Completeness in robot motion planning," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, ser. WAFR. USA: A. K. Peters, Ltd., 1995, p. 419–429. [Online]. Available: https://goldberg.berkeley.edu/pubs/completeness.pdf

[71] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6174325

[72] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, 2000. [Online]. Available: https://ieeexplore.ieee.org/document/844730

[73] M. Esposito, "realtime_urdf_filter," https://github.com/IFL-CAMP/easy_handeye, 2024.

[74] N. Blodow, "realtime_urdf_filter," https://github.com/blodow/realtime_urdf_filter, 2024.

[75] M. Bhardwaj, "franka_motion_control," https://github.com/mohakbhardwaj/franka_motion_control, 2024.