Learning Obstacle Representations for Neural Motion Planning
Robin Strudel, Ricardo Garcia, Justin Carpentier, Jean-Paul Laumond, Ivan Laptev, Cordelia Schmid

To cite this version:
Robin Strudel, Ricardo Garcia, Justin Carpentier, Jean-Paul Laumond, Ivan Laptev, et al.. Learning Obstacle Representations for Neural Motion Planning. CoRL 2020 - Conference on Robot Learning, Nov 2020, Cambridge MA / Virtual, United States. hal-02944348

HAL Id: hal-02944348
https://hal.archives-ouvertes.fr/hal-02944348
Submitted on 21 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Learning Obstacle Representations for Neural Motion Planning

Robin Strudel
Inria

Ricardo Garcia
Inria

Justin Carpentier
Inria

Jean-Paul Laumond
Inria

Ivan Laptev
Inria

Cordelia Schmid
Inria

Abstract: Motion planning and obstacle avoidance is a key challenge in robotics applications. While previous work succeeds to provide excellent solutions for known environments, sensor-based motion planning in new and dynamic environments remains difficult. In this work we address sensor-based motion planning from a learning perspective. Motivated by recent advances in visual recognition, we argue the importance of learning appropriate representations for motion planning. We propose a new obstacle representation based on the PointNet architecture [1] and train it jointly with policies for obstacle avoidance. We experimentally evaluate our approach for rigid body motion planning in challenging environments and demonstrate significant improvements of the state of the art in terms of accuracy and efficiency.

Keywords: neural motion planning, obstacle avoidance, representation learning

1 Introduction

Motion planning is a fundamental robotics problem [2, 3] with numerous applications in mobile robot navigation [4], industrial robotics [5], humanoid robotics [6] and other domains. Sampling-based methods such as Rapidly Exploring Random Trees (RRT) [7] and Probabilistic Roadmaps (PRM) [8] have been shown successful for finding a collision-free path in complex environments with many obstacles. Such methods are able to solve the so-called piano mover problem [9] and typically assume static environments and prior knowledge about the shape and location of obstacles. In many practical applications, however, it is often difficult or even impossible to obtain detailed a-priori knowledge about the real state of environments. It is therefore desirable to design methods relying on partial observations obtained from sensor measurements and enabling motion planning in unknown and possibly dynamic environments. Moreover, given the high complexity devoted to exploration in sampling-based methods, it is also desirable to design more efficient methods that use prior experience to quickly find solutions for motion planning in new environments.

To address the above challenges, several works [10, 11, 12, 13, 14, 15] adopt neural networks to learn motion planning from previous observations. Such Neural Motion Planning (NMP) methods either improve the exploration strategies of sampling-based approaches [13] or learn motion policies with imitation learning [12, 15] and reinforcement learning [14]. In this work we follow the NMP paradigm and propose a new learnable obstacle representation for motion planning.

Motivated by recent advances in visual recognition [1, 16, 17], we argue that the design and learning of obstacle representations plays a key role for the success of learning-based motion planning.

In this work, we proposed a new representation based on point clouds which are first sampled from visible surfaces of obstacles and then encoded with a PointNet [1] neural network architecture, see Fig. 1. We learn our obstacle representation jointly with the motion planning policies in the SAC (Soft Actor Critic) reinforcement learning framework [18]. In particular, while using environments composed of 3D-box shapes during training, we demonstrate the generalization of our

---

1 Inria, École normale supérieure, CNRS, PSL Research University, 75005 Paris, France.
2 University Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.
Figure 1: Overview of our approach. (a) We aim to find a collision-free path for a rigid body from its current configuration \( q_t \) to the goal configuration \( q_g \). (b) We assume no prior knowledge about the scene and represent obstacles by points and normals sampled on object surfaces. (c) Our neural network learns the PointNet encoding of observed points and normals together with the motion policy. (d) The learned network generates actions that move the body towards the goal configuration along a collision-free path.

motion planning policies to complex environments with objects of previously unseen shapes. We also show our method to seamlessly generalize to new object constellations and dynamic scenes with moving obstacles. We evaluate our method in challenging simulated environments and validate our representation through ablation studies and comparison to the state of the art. Our method significantly improves the accuracy of previous NMP methods while being one order of magnitude more computationally efficient compared to close competitors.

The contributions of our work are threefold. First, we propose a new learnable obstacle representation based on point clouds and the PointNet neural architecture [1]. Second, we learn our representation jointly with policies for rigid body motion planning using reinforcement learning. Finally, we experimentally evaluate our approach and demonstrate significant improvements of the state of the art in motion planning in terms of accuracy and efficiency. Qualitative results along with a video are available on the project webpage [19].

2 Related Work

Sampling-based motion planning. Sampling-based methods such as RRT [7] have been extensively studied for motion planning [2, 20, 3, 7, 8]. Such methods can deal with complex environments, however, they typically assume a complete knowledge of the robot workspace and static obstacles. Given the complexity of sampling-based methods, recent work proposes efficient exploration schemes [21, 22, 23, 24], e.g., by reusing previously discovered paths in similar scenarios [23] or biasing RRT search to promising regions [24]. Our work does not assume any a-priori knowledge about the environment and can deal with moving obstacles. Moreover, while our method performs an extensive exploration during training, the learned policies directly generate feasible paths during testing in new environments.

Neural motion planning. Learning-based methods for motion planning have been introduced in [10, 11]. Motivated by the success of deep learning, a number of more recent methods explore neural networks for motion planning and obstacle avoidance. Ichter et al. [13] learns a policy with the imitation learning using RRT solutions as demonstrations. While improving the efficiency of RRT, [12, 13, 15] still require sampling at test time and are not easily applicable in scenes with moving obstacles. Our work is most related to Jurgenson and Tamar [14] who explore reinforcement learning (RL) to learn motion policies. Similar to [14] we use RL and learn to avoid obstacles using a negative collision reward. While [14] presents results in relatively simple 2D environments, we propose a new learnable obstacle representation that generalizes to complex 3D scenes. We experimentally compare our method to [13, 14, 15] and demonstrate improved accuracy.

Visual Representation. The NMP methods [14, 15, 13] assume full knowledge of the workspace and use an obstacle encoding either based on a 2D image of obstacles encoded with a convolutional neural network (CNN) [14], or an occupancy grid [13] or a volumetric point cloud [15] encoded with a multi-layer perceptron (MLP). We show that obstacles representation is critical to solve complex problems with rich workspace variations and propose to rely on a point cloud representation of obstacles coupled with PointNet [1]. Qi et al. [1] demonstrated the performance of PointNet to classify and segment point clouds, in this work we propose to use it to encode obstacles.
3 Method

3.1 Overview of the method

In this work we consider rigid robots with 2 to 6 degrees of freedom (DoF). Let $W \subseteq \mathbb{R}^3$ be the workspace of the robot, containing a set of obstacles $V$. We denote $C_{\text{free}}$ the open set of configurations where the robot does not collide with the obstacles $V$ and $C_{\text{collision}} = C \setminus C_{\text{free}}$. Given a start configuration $q_0 \in C_{\text{free}}$ and a goal configuration $q_g \in C_{\text{free}}$, motion planning aims at finding a collision-free path which connects the start configuration to the goal configuration. A continuous function $\tau : [0, 1] \rightarrow C$ is a solution if $\tau(0) = q_0$ and $\tau(1) = q_g$. A motion planning problem is thus defined by a start configuration, a goal configuration and a set of obstacles $V$.

In this work, we represent obstacles $V$ by sets of points (point clouds) and the corresponding normals sampled on the surface of obstacles. This representation is valid for arbitrary shapes and can be obtained with a depth sensor. We consider the robot as an embodied agent which senses surrounding obstacles with a panoramic camera (Fig.1a). The point cloud, expressed in the robot local coordinate frame (Fig.1b), is then processed with a PointNet architecture [1] to encode the obstacles (Fig.1c). The goal vector $g_0$ is expressed in the robot local coordinate frame and is concatenated to the PointNet obstacle encoding. This vector is processed by a MLP that generates actions, bringing the robot closer to the goal while avoiding obstacles (Fig.1d). We learn the policy jointly with the PointNet encoding of the obstacles (Fig.1c) in an end-to-end fashion with reinforcement learning. In the next two sections, we first give details on the obstacle representation and then describe policy learning.

3.2 Obstacle representation for motion planning

We aim to learn a function that encodes the obstacles and the goal configuration as a vector enabling subsequent motion planning. While many parametric functions could be used as an encoder, we follow advances in visual recognition [1, 16, 17] and define obstacle representations by a neural network learned jointly with the task of motion planning. We experimentally demonstrate the significant impact of the encoder on the performance of motion planning in Section 4.

In previous works [13, 14, 15], obstacles have been represented by occupancy grids encoded with a MLP [13, 15] or images encoded with a CNN [14] assuming global workspace knowledge. In our work we use points sampled on the surface of obstacles along with their oriented normals. Such measurements can be obtained using a depth sensor either placed on the robot or at other locations. A set of obstacles $V$ is represented by a finite set $S_{\text{normals}} = \{(x_i, n_i)\}_{i=1,...,N} \in \mathbb{R}^{N \times 2d}$ where $d = 2, 3$ and the points $x_i$ and normals $n_i$ are expressed in the robot local coordinate frame. We denote by $\alpha_i$ the couple $(x_i, n_i)$. We define the goal $g$ as the displacement to reach the goal configuration from the current robot configuration.

To process a point cloud, we use a PointNet [1] like network (Fig.1c) reduced to its core layers for computation efficiency which we describe below. The idea of PointNet is to use a function which is symmetric with respect to the input to get a model invariant to input permutations. Based on this idea we build a network composed of two MLPs $u$ and $w$ with Exponential Linear Units (ELUs) activation [25] as shown in Fig.1c. The first MLP $u$ is shared across point cloud elements $\alpha_i$ (Fig.1c blue block) and is used to generate local features of each element $u(\alpha_i)$. Then, a max operation is applied to obtain a global feature $v$ encoding the point cloud. The global feature $v$ is further concatenated with the goal $g$ (Fig.1c green block) and passed through a second MLP $w$ (Fig.1c orange block):

$$v(\alpha_1, ..., \alpha_N) = \max_{i=1,...,N} [u(\alpha_1), ..., u(\alpha_N)]$$

$$f(\alpha_1, ..., \alpha_N, g) = w(v(\alpha_1, ..., \alpha_N), g)$$

The max operator is a max-pooling operation: given a list of feature vectors of size $N \times d$, it outputs a vector of size $d$ by taking the maximum over the $N$ points in each coordinate.

The PointNet encoding of obstacles $u$ is trained jointly with the policy $w$. For the training of $u$ and $w$ we compare imitation learning and reinforcement learning schemes as described in the next section.
3.3 Learning policies for motion planning

We cast the motion planning problem as a Markov decision process (MDP) [26]. The state space $S$ is the configuration space of the robot $C$, $A$ is the space of valid velocities at the current configuration and $\hat{C}$ is a representation of the workspace along with the goal configuration $g$. Given a robot configuration $q \in \hat{C}_{\text{free}}$ and $v$ an admissible velocity vector, we denote $q(v)$ as the configuration reached by applying the velocity vector $v$ to the robot for a fixed time. As the robot can hit into obstacles, we consider $q_{\text{free}}(v)$ which returns the last collision free configuration on the path connecting $q$ to $q(v)$. Then the dynamics $p$ is defined as $p(q,v) = q_{\text{free}}(v)$.

We aim at learning policies to solve motion planning problems. For that purpose, we explore and compare policies trained with imitation learning (behavioral cloning) and reinforcement learning. To train a policy with imitation learning [27], we collect a dataset $D$ and compare policies trained with imitation learning (behavioral cloning) and reinforcement learning.

We aim at learning policies to solve motion planning problems. For that purpose, we explore and compare policies trained with imitation learning (behavioral cloning) and reinforcement learning. To train a policy with imitation learning [27], we collect a dataset $D$ and compare policies trained with imitation learning (behavioral cloning) and reinforcement learning.

To train a policy with reinforcement learning [26], we define a reward function as follows. Given a learnable policy $\pi$, we minimize the $L^2$ loss, $\mathcal{L}(\pi) = ||a_t - \pi(a_t)||_2$, between the expert action $a_t$ and the policy applied to the expert observation $o_t$.

To train a policy with reinforcement learning [26], we define a reward function as follows. Given a learnable policy $\pi$, we minimize the $L^2$ loss, $\mathcal{L}(\pi) = ||a_t - \pi(a_t)||_2$, between the expert action $a_t$ and the policy applied to the expert observation $o_t$.

We evaluate our method in a number of different environments, namely, 2D and 3D environments used in [13, 15], our own 3D environments and an environment based on a classic motion planning problem [20] where a S-shape with 6 DoF should go through a thin slot. We consider rigid body robots which are either a 2D/3D sphere with 2/3 DoF or a S-shape body with 6 DoF. We use distinct workspaces for training and evaluation so that policies are evaluated on workspaces unseen during training. We evaluate the success rate of a policy over 400 rollouts. At the end of a rollout, the environment is reset: a new random workspace is sampled along with a start and goal configuration. A rollout is considered successful if it reaches a configuration near the goal defined by an epsilon neighborhood before the maximum number of steps is reached. We describe details for each of our environments below.

2D-Narrow [13]: we generate 2D environments from Icht et al. [13] using publicly available code [30]. The environment contains a sphere robot navigating in 2D workspaces composed of 3 obstacles.

4 Experimental Results

Below we describe our experimental setup and implementation details in Sections 4.1 and 4.2. Section 4.3 evaluates alternative obstacle representations while Sections 4.4 and 4.5 compare our approach to the state of the art in challenging environments.

4.1 Environments

We evaluate our method in a number of different environments, namely, 2D and 3D environments used in [13, 15], our own 3D environments and an environment based on a classic motion planning problem [20] where a S-shape with 6 DoF should go through a thin slot. We consider rigid body robots which are either a 2D/3D sphere with 2/3 DoF or a S-shape body with 6 DoF. We use distinct workspaces for training and evaluation so that policies are evaluated on workspaces unseen during training. We evaluate the success rate of a policy over 400 rollouts. At the end of a rollout, the environment is reset: a new random workspace is sampled along with a start and goal configuration. A rollout is considered successful if it reaches a configuration near the goal defined by an epsilon neighborhood before the maximum number of steps is reached. We describe details for each of our environments below.

2D-Narrow [13]: we generate 2D environments from Icht et al. [13] using publicly available code [30]. The environment contains a sphere robot navigating in 2D workspaces composed of 3 obstacles.
randomly generated narrow gaps as shown in Fig. 2 and random start and goal configurations. We set the maximum number of policy steps to 50. 

**3D-Qureshi [15]:** 3D workspaces with a sphere robot from Qureshi et al. [15] contain axis-aligned boxes with fixed center and varying sizes. We used open-source code from Qureshi [31] to generate the workspaces.

**3D-Boxes:** our environment composed of 3 to 10 static boxes generated with random sizes, positions and orientations as illustrated in Fig. 4(a). The maximum number of steps is set to 80.

**3D-Synthetic:** a variant of 3D-Boxes composed of unseen synthetic obstacles such as capsules, cylinders and spheres instead of boxes as illustrated in Fig. 4(b).

**3D-YCB:** a variant of 3D-Boxes composed of real objects from the YCB dataset [32] recorded with a RGB-D camera, see Fig. 4(c).

**3D-Dynamic:** a variant of 3D-Boxes with dynamic obstacles moving in real time. For each box two placements are sampled uniformly and the current box placement is interpolated between the two.

**Slot:** a S-shaped robot with 6 DoF which has to go through a thin gap with varying size, 2 times to 8 times wider than the smallest robot link. This is a classic problem in motion planning [20] illustrated in Fig. 4(d).

### 4.2 Implementation details

Below we describe implementation details for our method and for other methods used for comparison. To train policies with reinforcement learning, we use Soft Actor Critic (SAC) [18] with automatic entropy tuning, a learning rate of \(3 \times 10^{-4}\) with Adam optimizer [33], a batch size of 256 and a replay buffer of \(10^6\) steps combined with Hindsight Experience Replay (HER) [29] with 80% of the trajectories goal relabeled, as in the original papers. We train a policy on \(2 \times 10^6\) environment steps before reporting the results. We use the open-source implementation of Pong [34]. To train policies with imitation learning, we use a learning rate of \(10^{-3}\) with Adam optimizer [33] and a batch size of 256. For Bi-RRT, once a solution is found, we shorten its length by randomly sampling two points along the solution, if the shortcut made out of these two points is collision free we modify the solution to include it. The maximal size of an edge in RRT corresponds to the maximal size of a policy step for RL or BC, in this way computing collision checking on an edge has the same cost for both.

For the MLP network, we use 3 hidden layers of size 256. For the convolutional neural network (CNN), we use 3 convolutional layers with 32 filters of size \(3 \times 3\) followed by a max-pooling operator and 3 hidden layers of size 256. For PointNet, we use 3 hidden layers of size 256 for the point encoder \(u\) and the global feature network \(w\) respectively. For the MLP, the goal is concatenated to the list of points. For the CNN, the current and goal configurations are concatenated to the feature vector after max-pooling.

For Ichter et al. [13], we used the code provided in [30] along with the dataset to train the conditional variational auto-encoder. Once trained we combined the learned sampling with Bi-RRT to report the results. For Qureshi et al. [15], we adjust the implementation provided by Qureshi [31] to our environments. We followed the training procedure described in [15]. For Qureshi neural replanning (NR), we run neural planning for 80 steps then use neural replanning recursively for 10 iterations each comprising 50 steps maximum. For Qureshi hybrid replanning (HR), if neural replanning fails to find a solution after 10 iterations, Bi-RRT is used to find a path between states where there is still a collision. For Jurgenson and Tamar [14], we adapted the open-source implementation [35] to run on our environment. For training, we use the same parameters as [14], image based workspace representation and reward definition. We follow their DDPG-MP method. The training dataset consists of \(10^4\) workspaces. For pre-training, we use 400 random steps per workspace and for training, we use Bi-RRT generated 10 expert trajectories per workspace.

Finally, the rigid bodies and obstacles are modeled using Pinocchio [36, 37] and collision checks are computed by FCL [38].

### 4.3 Comparison of obstacle representations and learning approaches

In this section we compare different policies on the 2D-Narrow environment [13]. Policies are trained with Behavioral Cloning (BC) or Reinforcement Learning (RL) using different obstacles representations presented in Fig. 2. The occupancy grid is a \(64 \times 64\) image (Fig. 2a), the point clouds are composed of 128 points either sampled on the interior of obstacles (Fig. 2b) or at their boundary (Fig. 2c, 2d). 128 points is a good trade-off between speed and accuracy as adding more points did...
Table 1: Comparison of different obstacles representations and policies training methods on the 2D-Narrow environment.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Behavioral Cloning</th>
<th>Reinforcement Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>64×64 Image - CNN</td>
<td>64.5%</td>
<td>44.5%</td>
</tr>
<tr>
<td>Interior points - MLP</td>
<td>42.0%</td>
<td>29.0%</td>
</tr>
<tr>
<td>Interior points - PointNet</td>
<td>76.0%</td>
<td>83.0%</td>
</tr>
<tr>
<td>Boundary points - PointNet</td>
<td>85.0%</td>
<td>93.8%</td>
</tr>
<tr>
<td>Boundary points and normals - PointNet</td>
<td>86.5%</td>
<td>99.5%</td>
</tr>
</tbody>
</table>

not improve the results. In Table 1, we report the success rate of RL policies after 1000 epochs which corresponds to $10^6$ interactions with the environment, a stage at which the policies performance have plateaued. For BC, we collect a dataset of solutions using Bi-RRT, containing $10^6$ steps in total and train for 200 epochs. This allows a fair comparison of BC and RL which have been trained on the same dataset size. As a baseline, we consider a policy which goes from start to goal in a straight line and has a success rate of 45.5%.

Table 1 shows that the choice of obstacles representation greatly impacts the policy success rate both for BC and RL. Using a representation based on an occupancy grid encoded with a CNN yields poor performance for both BC and RL. Similarly, if interior points are encoded with a MLP the performance is low for both BC and RL. In both case results with RL are below BC. For BC, the training set is composed of expert demonstrations which is fixed for every representation. In contrast, as the RL training set is generated by the learning policy, the obstacles representation impacts the quality of the training set. Encoding interior points with PointNet results in a significant gain, +34% for BC and +54% for RL. Using PointNet in combination with boundary points and normals increases the performance by +10% for BC and +16% for RL. We can also observe that normals improve the performance over boundary points alone. Furthermore, with a more stable obstacle encoding, RL outperforms BC by a margin, +13% in the case of boundary points and normals.

The difference between policies trained with BC or RL is illustrated in Fig.3(a,b). For each policy, a vector field has been generated by using a grid covering the environment and computing the policy output at each point given a fixed goal plotted in red. We observe that policies trained with BC can fail close to edges of obstacles. This is a typical problem of imitation learning [39] which is limited to perfect, expert trajectories in the training set and does not observe failure cases. In contrast, RL explores the environment during training and generates actions pointing away from obstacles as it has been trained explicitly to avoid collisions.

We compare our approach to state-of-the-art neural motion planning approaches [13, 14, 15] and Bi-RRT [7] in Table 2. We compare in terms of success rate, number of configurations (nodes) explored
Table 2: Comparison to the state of the art on the 2D-Narrow environment.

<table>
<thead>
<tr>
<th>Method</th>
<th>Success Rate</th>
<th>Nodes</th>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bi-RRT [7]</td>
<td>100%</td>
<td>358</td>
<td>0.65</td>
</tr>
<tr>
<td>Ichter $\lambda = 0.5$ [13]</td>
<td>100%</td>
<td>232</td>
<td>1.30</td>
</tr>
<tr>
<td>Ichter $\lambda = 0.9$ [13]</td>
<td>100%</td>
<td>207</td>
<td>1.22</td>
</tr>
<tr>
<td>Qureshi (NR) [15]</td>
<td>68.0%</td>
<td>102</td>
<td>0.52</td>
</tr>
<tr>
<td>Qureshi (HR) [15]</td>
<td>95.0%</td>
<td>950</td>
<td>0.69</td>
</tr>
<tr>
<td>Qureshi (HR) [15]</td>
<td>95.0%</td>
<td>950</td>
<td>0.69</td>
</tr>
<tr>
<td>Jurgenson [14]</td>
<td>47.0%</td>
<td>12</td>
<td>0.56</td>
</tr>
<tr>
<td>Us</td>
<td>99.5%</td>
<td>10</td>
<td>0.63</td>
</tr>
</tbody>
</table>

Table 2: Comparison to the state of the art on the 2D-Narrow environment.

before finding a successful path and in terms of length of the found solution. We chose to compare
the number of configurations explored to find a solution because connecting two configurations
requires to perform collision checking which represents 95% of time spent by motion planning
algorithms [40]. Bi-RRT achieves a success rate of 100% as a solution is found if the algorithm
is run long enough but it requires 35 times more nodes than RL to find a solution and shorten the
path. Ichter et al. [13], which is also based on RRT, requires 20 times more nodes than RL to find a
solution and yields longer solutions overall. The neural replanning (NR) and hybrid replanning (HR)
approaches of Qureshi et al. [15] allow to find short paths at the price of extensive use of collision
checking, which is limiting in scenarios with time constraints. Jurgenson and Tamar [14] uses an
image representation of obstacles, yielding a low success rate which correlates with the results of
Table 1, it mostly solves problems with straight solutions which explains the short path length.

In Fig.3(c,d) we illustrate the number of nodes required to find a path. While RL outputs short paths
leading directly to the goal (Fig.3c), Bi-RRT explores the space in several directions before finding
a suboptimal path which then needs to be shortened (Fig.3d).

4.4 Towards a realistic setup: 3D environments with local observability

![3D Environments](image)

Figure 4: 3D environments with S-shaped robot composed of: (a) boxes obstacles, (b) sphere, cylin-
der and cone obstacles, (c) YCB dataset [32] obstacles, (d) a thin slot. We plot the start configuration
in red, the current configuration in yellow and the goal configuration in green.

<table>
<thead>
<tr>
<th>3D-Boxes</th>
<th>3D-Synthetic</th>
<th>3D-YCB</th>
<th>3D-Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere - Global</td>
<td>95.8%</td>
<td>91.3%</td>
<td>91.2%</td>
</tr>
<tr>
<td>S Shape - Global</td>
<td>89.0%</td>
<td>85.3%</td>
<td>88.7%</td>
</tr>
<tr>
<td>Sphere - Local</td>
<td>99.3%</td>
<td>87.0%</td>
<td>87.7%</td>
</tr>
<tr>
<td>S Shape - Local</td>
<td>97.0%</td>
<td>87.3%</td>
<td>96.0%</td>
</tr>
</tbody>
</table>

Table 3: Comparison of sphere and S-shaped agents trained with global or local obstacle observation.

We compare agents trained with either global observations, where points are sampled for all ob-
stacles surface as in Section 4.3, or local observations where obstacles are only observed locally,
through a camera which is closer to a realistic setup. For the local observation, we consider agents
equipped with a panoramic camera observing the local geometry (Fig.1b). To model such agents,
we use ray tracing and cast rays from the center of the robot in every direction by uniformly sam-
pling points on the sphere. Each ray hitting an obstacle provides a point with its normal and the
agent observation consists of the point cloud formed by the union of these points. For local observa-
tions (Local), we use a point cloud of 64 points, corresponding to a density of 60 points per meter
squared. For global observations (Global), we use 256 points sampled uniformly on the obstacles
surface which corresponds to the same point density as Local. The two observations thus have the
same geometry resolution.
Table 4: Comparison of our approach to Qureshi’s [15] environment and our environment

<table>
<thead>
<tr>
<th></th>
<th>3D-Qureshi [15]</th>
<th>3D-Boxes</th>
<th>3D-Synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qureshi (NR) [15] - MLP</td>
<td>96.0%</td>
<td>84.0%</td>
<td>81.0%</td>
</tr>
<tr>
<td>Qureshi (HR) [15] - MLP</td>
<td>99.5%</td>
<td>88.0%</td>
<td>84.5%</td>
</tr>
<tr>
<td>Sphere - Global</td>
<td>100%</td>
<td>95.8%</td>
<td>91.3%</td>
</tr>
</tbody>
</table>

We consider 3D environments and compare agents controlling either a sphere robot with 3DoF or a S-shaped robot with 6DoF as shown in Fig.4 and report results in Table 3. All the policies are trained on the 3D-Boxes environment (Fig.4a) exclusively. The S-shape problem is harder to solve than the sphere one but our approach still yields good results with a performance of 97% on 3D-Boxes for the local observation. Overall Local1 and Global1 policies yield similar performances which shows that our approach still works on harder problems where only local knowledge of obstacles is available. When tested on 3D-Boxes, Local1 yields better results than Global1 while the generalization performance are better for Global1 when tested on unseen environments (Fig.4b,c). While solely trained on problems with static obstacles from 3D Boxes (Fig.4a), the policies generalize to unseen scenes containing new set of synthetic obstacles (3D-Synthetic) and real obstacles recorded with depth sensor from the YCB dataset (3D-YCB). The policies trained with our approach also solve challenging scenarios with dynamic obstacles moving in real time (3D-Dynamic). This highlights the advantage of using a policy which directly computes the next action instead of RRT-based approaches relying on offline path planning [7, 15, 41].

We also compare our approach to Qureshi et al. [15]. For a fair comparison we use global obstacles representation for our approach as [15] uses full obstacles knowledge and report results in Table 4. On 3D-Qureshi, we show that our approach successfully solves the proposed problems. On 3D-Boxes and 3D-Synthetic we show that our approach has better generalization abilities while only needing 20 nodes on average to solve problems where [15] requires more than 400 nodes.

4.5 S-shape motion planning

We consider a challenging problem in motion planning composed of a S-shaped robot and a thin gate it has to go through, introduced by Latombe [20] and shown in Fig.4(d). The width of the gate determines the difficulty of the problem. We consider problems with a gate of random width, sampled to be 2 times to 8 times wider than the smallest dimension of each robot link, as a comparison, the gate of [20] is 2.5 times wider. We compare the performance of an agent trained with Local1 observations to Bi-RRT with an allocated budget of 50000 nodes for each problem. The learned policy has a success rate of 97.7% whereas Bi-RRT has a success rate of 62.5% on average when tested on 400 planning problems. In contrast with experiments of Section 4.3, Bi-RRT does not succeed to solve every problem with the allocated nodes. While our trained policy provides solutions in real-time composed of 40 nodes on average, the computational burden of RRT is increasing significantly as the number of explored configurations increases which is typically the case for this environment where many nodes need to be expanded to find a solution. We have also noted that our policy adapts its behavior to minimize the path length according to the slot size. Indeed, when the slot is thin, e.g. 2 times wider than the smallest robot link, the motions are quite constrained, the policy inserts a link by translating the S-shape, rotates by 90 degrees and translates again which was also the only solution found when using RRT. When the slot is wider we observe that the policy uses the wider space provided by the diagonal of the slot to reduce overall motion.

5 Conclusion

This paper introduces a new framework for neural motion planning. Obstacles are represented by point clouds and encoded by a PointNet architecture. PointNet encoding and motion policy are trained jointly with either behavioral cloning or reinforcement learning. We show that PointNet encoding outperforms state-of-the-art representations based on image-based CNNs and latent representations. Furthermore, we show that RL learns better models than BC. Future work will address rigid robots with multiple links as for example robotic arms performing manipulation tasks in cluttered environments directly captured by a camera.
Acknowledgments

We thank Loïc Estève for the helpful discussions. This work was partially supported by the HPC resources from GENCI-IDRIS (Grant 20XX-AD011011163), Louis Vuitton ENS Chair on Artificial Intelligence, and the French government under management of Agence Nationale de la Recherche as part of the Investissements d'avenir program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

References


