

Copyright Notice

©2021 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Speeding Up Optimization-based Motion Planning through Deep Learning

Johannes Tenhumberg^{1,2}, Darius Burschka³ and Berthold Bäuml^{1,2}

Abstract—Planning collision-free motions for robots with many degrees of freedom is challenging in environments with complex obstacle geometries. Recent work introduced the idea of speeding up the planning by encoding prior experience of successful motion plans in a neural network. However, this “neural motion planning” did not scale to complex robots in unseen 3D environments as needed for real-world applications. Here, we introduce “basis point set”, well-known in computer vision, to neural motion planning as a modern compact environment encoding enabling efficient supervised training networks that generalize well over diverse 3D worlds. Combined with a new elaborate training scheme, we reach a planning success rate of 100 %. We use the network to predict an educated initial guess for an optimization-based planner (OMP), which quickly converges to a feasible solution, massively outperforming random multi-starts when tested on previously unseen environments. For the DLR humanoid Agile Justin with 19 DoF and in challenging obstacle environments, optimal paths can be generated in 200 ms using only a single CPU core. We also show a first successful real-world experiment based on a high-resolution world model from an integrated 3D sensor.

I. INTRODUCTION

At the core, robotic motion planning is about getting from the start joint configuration to a goal configuration while avoiding obstacles in the environment and self-collision along the path. Solving a motion task fast and efficiently does not only mean that the robot spends less time contemplating and more time moving. If a solver can find a feasible solution in a fraction of a second, there opens up the door for more reactive planning and integrating the global planner more tightly into the sensor/vision-action loop. Furthermore, we can tackle more high-level tasks with multiple smaller motion problems if each substep can be solved efficiently.

An interesting approach for speeding up motion planning is not to solve each planning problem anew but to use experience from having solved related motion tasks before. Important for the applicability of such experience-based planners to real-world problems is that they not only allow for arbitrary start and goal configurations but also for arbitrary environment geometries as an input. So, “related tasks” only means that the robot geometry is the same.

This paper presents a deep learning enhancement for an optimization-based planner that allows robot motion planning in high-resolution environments with complex obstacle geometries. For the DLR’s humanoid robot Agile Justin [1] with 19 DoF, feasible trajectories can be computed in only 200 ms on a single CPU core (see Fig. 1).

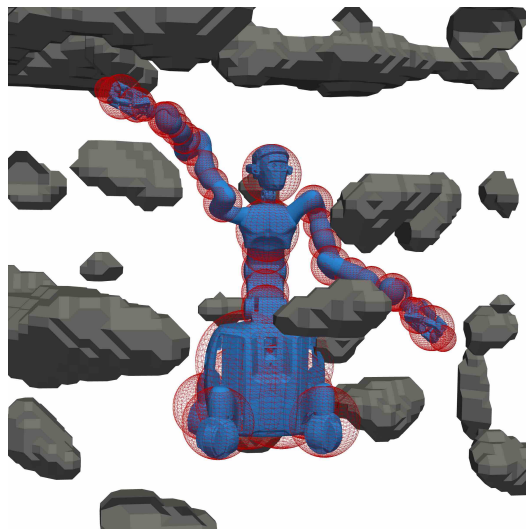


Fig. 1. DLR’s Agile Justin [1] in a challenging obstacle environment, we generate with simplex noise for training and testing. The sphere model of the 19DoF humanoid is shown as red wireframes. For videos of the motions, visit <https://dlr-ai.github.io/2022-iros-planning>.

A. Related Work

Two popular but fundamentally different approaches to solving a motion task in robotics are sampling-based planners (SMP) and optimization-based motion planners (OMP). SMP [2] use randomness at their core and can guarantee to globally find a feasible, i.e., collision-free, path (if there is any). Examples are rapidly exploring random trees (RRT) and their extensions, which explore the configuration space iteratively and build a graph of possible configurations until a branch finds the goal. As the search space grows exponentially with the robots’ degrees of freedom (DoF), vanilla variants of SMP don’t scale well to complex robots.

In OMP, the motion task is formulated as an optimization problem [3, 4, 5] and gradient-based techniques are used for non-convex optimization to find a solution. The computational complexity of those methods can, in principle, scale linearly with the DoF¹ and converge quickly to a smooth trajectory. However, they only find a local minimum and strongly depend on the initial guess.

For non-trivial robot kinematics and environments, the objective function usually has many local minima. Therefore, a multi-start approach is needed to find a feasible global solution, massively slowing down OMP in complex scenarios. STOMP [4] and CHOMP [3] try to mitigate the strong

¹DLR Institute of Robotics & Mechatronics, Germany; ²Deggendorf Institute of Technology, Germany; ³Technical University of Munich, Germany
Contact: johannes.tenhumberg@dlr.de
Supported by the Bavarian Ministry of Economics, project SMiLE2gether.

¹Think of gradient descent where for each iteration only the computation of the gradient and its addition to the current joint angles are performed – both operations linear in the DoF when using, e.g., automatic differentiation.

dependency on an initial guess by introducing stochasticity to the optimization. TrajOpt [5] uses convex hulls to represent the robot and its environment and increases the attraction basin for each optimum. But still, OMP needs multi-starts to find a feasible global solution.

OMP can be sped up by using experience from previously solved motions tasks by providing an *educated* initial guess. Jetchev and Toussaint [6] first proposed to save a database of feasible trajectories and look up a reasonable first guess for a new problem. Merkt et al. [7] improved the idea through more efficient database storage and tested it on a humanoid robot. But for both methods, only results for fixed or only slightly varying environments are shown.

Instead of databases, neural networks are often used to encode the experience. The expectation is that they are more memory efficient, encode various solutions implicitly, learn a general understanding of feasible trajectories, and produce useful predictions in unseen settings. Qureshi et al. [8, 9] coined the term Motion Planning Network (MPNet) and then improved on the idea. They used an RRT planner to collect feasible trajectories in 2D and 3D worlds and encode the environment with point clouds. Their method works on the Baxter robot over ten known table scenes with 1000 paths per scene. Strudel et al. [10] showed that they could outperform these results by employing the PointNet [11] architecture to encode the point clouds of the environments. They achieved good results in 3D with a sphere and a rigid S-shape with three translational and rotational DoFs but did not consider a robotic application. Bency et al. [12] and Lembono et al. [13] applied variations of the idea successfully to the two humanoids, Baxter and PR2. However, they only used a single fixed environment without generalization to different worlds. Lehner and Albu-Schäffer [14] trained a Gaussian mixture model to steer the search in a probabilistic roadmap. The approach was demonstrated on a real 7 DoF robot but equally only for one fixed world.

Also, other learning methods have been applied to this problem. For example, Jurgenson and Tamar [15] used reinforcement learning with convolution layers to process the occupancy map of the world for 2D serial robots. They invoke a classical planner only for cases where random exploration fails to find a feasible solution, so they implicitly use this expert knowledge to guide the training. Pandey et al. [16] introduces a different approach, where the dataset generation is skipped entirely, and the network is directly trained using the objective function of the planning problem as the training loss, i.e., no supervision is used. However, they only use geometric primitives to represent environments with few obstacles, limiting the flexibility.

For a more detailed overview of the literature, also refer to Surovik et al. [17]. They introduce the term “data-driven trajectory initialization” (DDTI) to generalize the ideas of “trajectory prediction” or “memory of motion”. To our best knowledge, up to now, there is no experienced-based method for speeding up motion planning that can predict feasible paths for a complex robot in a large set of challenging and previously unseen 3D worlds.

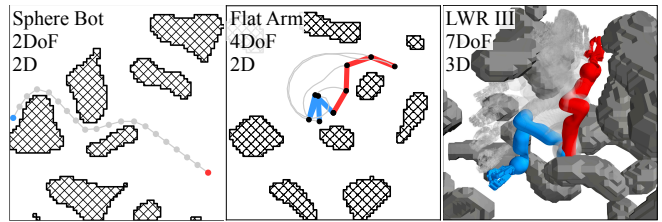


Fig. 2. Three different robots in random 2D and 3D environments, generated with simplex noise. A motion problem is described by the world (hatched), the start configuration (blue), and the goal configuration (red). The solution, the shortest feasible path from start to goal, is drawn in gray. See Table I for an overview of the different robots and worlds in numbers.

B. Contributions

We propose a neural network that encodes the experience of optimal trajectories over various tasks and worlds. We train the network supervised to predict the correct path for a given motion problem consisting of a world (obstacle environment) and start and end configurations. We then use the prediction of the network to warm-start an OMP, which ensures feasibility and smoothes the path. Our main contributions are:

- Introducing substeps into CHOMP[3] to explicitly calculate the swept volume to facilitate that no collisions are missed between discrete waypoints.
- The introduction of the basis point sets (BPS), described by Prokudin et al. [18], into learning-based motion planning. This memory- and computational-efficient encoding enables training and generalization for complex robots in challenging environments.
- Building new demanding datasets of motion tasks and optimal solution paths (via OMP) for training and testing. The dataset includes autogenerated random worlds with configurable complexity.
- A training scheme, where we combine the network and the objective function as a metric to efficiently clean, extend and boost an initial dataset.
- Extensive experiments in simulation for different robots ranging from a simple 2D sphere bot up to the 19 DoF humanoid Agile Justin in 3D. For Agile Justin, we also report a first real-world experiment, showing that Sim2Real transfer is working.

II. CHOMP-LIKE MOTION PLANNING

For generating the training samples for our network as well as for online post-processing, we use OMP. Ultimately, the neural network should do the primary workload in solving a motion task. Therefore the runtime efficiency of the OMP is not of utmost importance. We implemented an OMP similar to CHOMP [3], where the robot is modeled as spheres and the environment by a signed distance field (SDF). A significant extension to CHOMP is that we introduce additional substeps interpolating between the time steps. This way, the swept volume can be computed to any arbitrary accuracy, i.e., guaranteeing to miss no obstacle, without increasing the number of optimization variables (see Fig. 3).

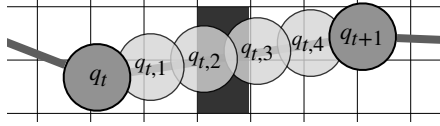


Fig. 3. Substeps $q_{t,u}$ between two discrete waypoints q_t and q_{t+1} to explicitly calculate the swept volume of the path in higher resolution.

OMP formulates the motion task of getting from point A to point B as an optimization problem with the desired path as the optimum. The path Q consists of a discrete set of waypoints of joint configurations

$$Q = [q_1, \dots, q_{N_t}], q_t \in \mathbb{R}^{N_q}, \quad (1)$$

and the task is encoded by an objective function $U(Q)$ measuring the quality of a given path. While, in general, the specific objective can be chosen freely, the two usually used terms to get a collision-free and short path are

$$U(Q) = U_C(Q) + \lambda U_L(Q). \quad (2)$$

The length cost U_L is given by

$$U_L(Q) = \frac{N_t - 1}{|q_{N_t} - q_1|^2} \sum_{t=1}^{N_t-1} |q_{t+1} - q_t|^2, \quad (3)$$

which favors short and smooth trajectories. It is often convenient to scale the length cost by the minimal possible path length, the direct connection $|q_{N_t} - q_1|^2$. Then, the shortest possible path always has a cost of one.

To calculate the collision cost between the robot and the environment, we need a model for both. The forward kinematics $F = f(q)$ maps from joint configurations to the link frames F_i and each link's geometry is described by a set of spheres $\mathcal{S}_i = \{x_{ik}, r_{ik}\}_{k=1}^{N_{s,i}}$ with centers and radii. The world is represented by a SDF $D(x)$ which gives the distance to the closest obstacle for each point x in the workspace. The collision cost is then given by the sum of all the collisions of the different body parts along the path

$$U_C(Q) = \sum_{t,u}^{N_t, N_u} \sum_{i=1}^{N_f} \sum_{k=1}^{N_{s,i}} c(D(F_i(q_{t,u}) \cdot x_{ik}) - r_{ik}), \quad (4)$$

$$q_{t,u} = q_t + \frac{u}{N_u}(q_{t+1} - q_t). \quad (5)$$

In extension to the original CHOMP algorithm [3], we subdivide the path between two waypoints into substeps $q_{t,u}$ via linear interpolation (see Fig. 3). This way, a collision-free path can be guaranteed, when the number of substeps N_u is adjusted to the step length and the voxel size of the SDF. So, the swept volume of each sphere is *explicitly* computed instead of the *implicit* computation CHOMP performs via a projection of the cartesian velocity vector of the moving spheres.

The smooth clipping function only considers the parts which are in collision by setting positive distances to 0. Thus,

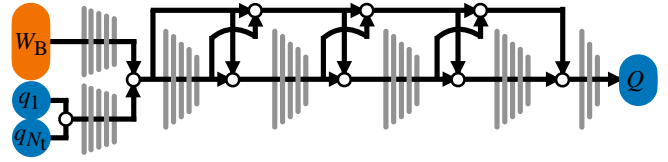


Fig. 4. Network architecture to map from a given motion task (world, start and end configuration) to an optimal path. Blocks of tapered Fully Connected Layers (gray) are combined like the DenseNet architecture [19] via skip-connections and concatenations. See the bottom of Table I for the number of network parameters used for the different robots.

a collision-free path has a collision cost of 0.

$$c(d) = \begin{cases} -d + \frac{\epsilon}{2} & , \text{ if } d < 0 \\ \frac{1}{2\epsilon}(d - \epsilon)^2 & , \text{ if } 0 \leq d \leq \epsilon \\ 0 & , \text{ if } \epsilon < d \end{cases} \quad (6)$$

In addition to collisions with the world, complex robots must also account for self-collision. Again, the cost sums up all the penetrations between the different body pairs

$$U_s(Q) = \sum_{t,u}^{N_t, N_u} \sum_{j>i}^{N_f, N_f} \sum_{k,l}^{N_{s,i}, N_{s,j}} c(|F_i(q_{t,u}) \cdot x_{ik} - F_j(q_{t,u}) \cdot x_{jl}| - r_{ik} - r_{jl}). \quad (7)$$

Here again, we use substeps to guarantee collision-free paths via explicitly checking the swept volume. To our knowledge, no other OMP-based planner is doing this.

With this formulation, the optimal path and solution to the motion task is the one with the lowest objective

$$Q^* = \underset{Q}{\operatorname{argmin}} U(Q). \quad (8)$$

We use gradient descent for iteratively finding a minimum of the objective function, starting with the initial guess Q_0 ,

$$Q_{i+1} = Q_i - \alpha \frac{\partial U(Q_i)}{\partial Q_i}. \quad (9)$$

We use vanilla gradient descent as efficiency in the OMP part is not the primary concern of our method and it allows for easy adaption and parameterization (only α)

III. DATASET ADAPTION FOR EFFICIENT LEARNING

The idea is to no longer rely on random multi-starts and speed up the planning time by using a neural network to predict an initial guess for OMP. The network should encode the experience of successful paths by learning a mapping from a motion task, consisting of a world and a start q_1 and end configuration q_{N_t} , to the intermediate waypoints of an optimal path $Q^* = [q_2, \dots, q_{N_t-1}]$. Fig. 4 shows the network architecture we use. Besides encoding the in- and output of the network, a crucial point for supervised learning is the dataset. The following section discusses several insights into the generation and usage of such a dataset with and for OMP. Our methods substantially increase the final prediction quality of our network and make training more efficient. See Section VI-A for experimental validation of these methods.

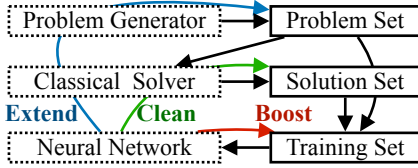


Fig. 5. Scheme showing the connection between non-learning-based solver, dataset, and neural network. The colored arrows indicate the information flow for cleaning, extending, and boosting the dataset with the guidance of the network. Cleaning: use the solver to update labels in the solutions set; Boost: overrepresent existing hard examples in training; Extend: generate new hard examples for the network and add them to the problem set.

A. Challenging Samples

The training data distribution should represent the actual application and focus on challenging examples. Suppose the dataset is too easy, and direct linear connections from A to B drastically outweigh more complex trajectories. In that case, training the network can quickly get stuck in a local minimum and predict only straight lines, regardless of the given task. One possibility of generating a dataset with more challenging motion problems is to consider only samples where OMP using the direct connection as an initial guess does not converge to a feasible path.

B. Consistent Samples

Besides finding challenging samples, the ambiguity of motion planning (more than one feasible solution) can become a problem for trajectory regression. Even if we assume that we can resolve the ambiguity of the optimal path via the objective for the shortness of the path, there will be “close calls” in the dataset (almost the same objective values but fundamentally different paths). Furthermore, the classical planner using a limited number of multi-starts can only produce suboptimal labels, making it hard for the network to learn consistent mapping. This is especially true for challenging tasks where the classical planner often fails and only produces feasible paths in a small fraction of the tries.

C. Symmetries of Motion Planning

Data generation is costly, so one can use symmetries in motion planning to efficiently use the information in each sample. If one has the optimal path from A to B, one also has the solution from B to A. This assumption is no longer valid if terms in objective function break the temporal symmetry. Furthermore, many robots also have spatial symmetry axes. Often, it is possible to align the first joint of the robot with one axis of the cartesian coordinate system of the environment. Doing so allows us to rotate the world and this first joint simultaneously without changing the optimality of the resulting trajectory in the new world. Chamzas et al. [20] use this spatial symmetry to store paths in their database more efficiently. While ideally, one wants to integrate these symmetries directly in the data representations or the network architecture, we used them to augment and increase the dataset.

Algorithm 1 Improvement of network and dataset

```

procedure MAIN
  create initial dataset  $G = \{(x_i, y_i)\}_{i=1}^{N_G}$  with OMP
  train net  $\Theta$  on dataset  $G$ 
  while improvement on testset do
    CleanDataset( $G, \Theta$ )
    ExtendDataset( $G, \Theta, N = |G|/20$ )
    BoostDataset( $G, \Theta, p_{\text{perc}} = 0.8, p_{\text{ratio}} = 0.9$ )
    train net  $\Theta$  on dataset  $G$ 
procedure CLEANDATASET( $G, \Theta$ ) ▷ Clean
  for  $(x_i, y_i)$  in  $G$  do
     $y_p \leftarrow \Theta(x)$ 
     $y_p^* \leftarrow \text{OMP}(x, y_p)$ 
    if  $U(y_p^*) \leq U(y)$  then
       $G.\text{replace}(y \leftarrow y_p^*)$ 
procedure EXTENDDATASET( $G, \Theta, N$ ) ▷ Extend
  for  $k \leftarrow 1$  to  $N$  do
     $x_i \leftarrow \text{sampleNewProblem}()$ 
    if  $V_p < U(\Theta(x_i))$  then
       $y_i \leftarrow \text{OMP}(x_i)$ 
       $G.\text{append}((x_i, y_i))$ 
procedure BOOSTDATASET( $G, \Theta, p_{\text{perc}}, p_{\text{ratio}}$ ) ▷ Boost
   $V \leftarrow [U(\Theta(x_i)) \text{ for } (x_i, y_i) \text{ in } G]$ 
   $V_p \leftarrow \text{percentile}(V, p_{\text{perc}})$ 
  for  $(x_i, y_i)$  in  $G$  do
    if  $(U(\Theta(x)) < V_p \text{ and } \text{random}(0, 1) < p_{\text{ratio}})$  then
       $G.\text{remove}((x_i, y_i))$ 

```

D. Interplay between Network and Dataset

We propose to use the neural network Θ to correct and enhance its own training data $G = \{(x, y)\}$. This approach is possible whenever synthetic data is used for training, and one has an objective metric to measure the quality of a prediction. The assumption is that the network can learn some aspects of the problem even on an imperfect dataset and its predictions become better than random guessing.

When generating a dataset with and for OMP, we can use the objective $U(Q)$ as a universal quality metric. The idea of interweaving the network training closer with the dataset generation and improvement is summarized in Fig. 5 and described by Algorithm 1. In what follows, we give a detailed explanation of the different methods.

1) *Clean*: First, the network can be used as guidance to double-check where the dataset is inconsistent. If the label and the network’s prediction Q_p are close, and the respective objective $U(Q_p)$ is small, no action is necessary. However, if there is a discrepancy between prediction and label, it is worthwhile to use more multi-starts with the OMP to get a better label for the sample. If a prediction has a better objective than the current label, we can replace it without adding any bias to the dataset. Doing so will improve the labels and make the dataset more consistent. This makes it easier for the network to find the underlying patterns.

2) *Boost*: After some training, the network has learned to predict good paths for the relatively simple samples, but

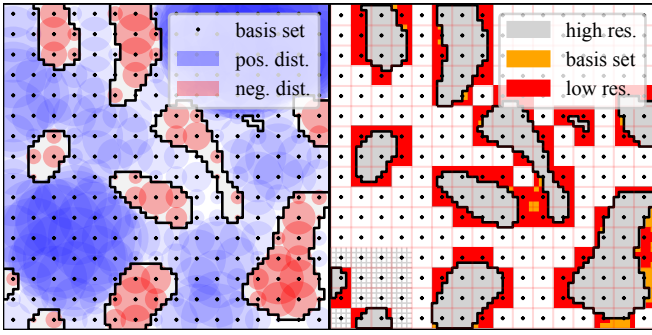


Fig. 6. Visualization of the representation capabilities of basis point sets. On the left, a regular BPS with $16 \times 16 = 256$ points and their respective distances to the closest obstacle in an occupancy map with $64 \times 64 = 4096$ pixels is shown. Blue areas describe positive distances to obstacles and are guaranteed free of obstructions. Red regions show negative distances and are completely inside barriers. We can draw no direct conclusion from the white areas, but they have to be marked as obstacles to be conservative. The BPS representation for this regular grid is equivalent to subsampling a high resolution 64×64 SDF. The right image shows the reconstruction of the BPS to the full image with the errors marked in orange. A patch of the original grid is added to highlight the difference in resolution. The far more conservative result of reducing the resolution of the occupancy grid by the same factor is highlighted in red. This comparison demonstrates that the basis set preserves more information than conservatively shrinking the occupancy grid by the same factor. A further advantage of the untruncated distances is that even if there are no nearby data points close for a basis point, it can nevertheless help in representing a surface further away.

the more challenging outliers are still not solved. Therefore, we use boosting to select challenging samples with higher probability during training, increasing the incentive to learn these samples. We steer this kind of curriculum again by using U as a metric: the higher the difference between the predicted and actual cost for a given sample is, the more challenging it is. In our experiments, the challenging tasks for the network correlated well with the relative path length and the number of obstacles in the scene.

3) *Extend*: Lastly, one can use the network to generate new samples. The idea is to use the network’s performance on a new sample as a metric for information gain. To improve the network, one wants specifically to add samples where the network performs poorly. To decide this before spending resources to produce a new label using OMP, one can use the objective of the prediction $U(Q_p)$. If it is small, the network can solve this task already. However, if the objective is significant, the task is challenging for the network, and we include it in the training set.

IV. BASIS POINT SET AS EFFICIENT WORLD ENCODING

If a network should speed up the OMP approach with a valuable warm-start, it needs to “understand” the robot motion in arbitrary unseen environments. Hence, a suitable encoding of the world is essential.

Two central environment representations in robotics and computer vision are occupancy grids W_O and point clouds W_P . Both have their advantages and disadvantages. While occupancy grids can be processed like images in 2D, their memory inefficiency and the high computational cost for the convolution operations become a burden in 3D. On the other

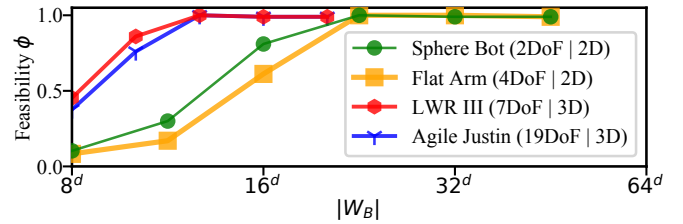


Fig. 7. Influence of the size of BPS $|W_B|$ on the feasibility ϕ of the network prediction after OMP. The $|W_B|$ on the x-axis is scaled by the world dimension. Too few points can’t represent the obstacles in enough detail for the network to make useful predictions. However, the required number of points is significantly smaller than the resolution of the underlying occupancy grid (64^d). It was not feasible to increase the input layer to 64^3 basis points in 3D.

hand, point clouds are a denser representation. Still, they have no fixed size and no inherent ordering, making it hard for a network to learn a permutation invariant mapping.

Prokudin et al. [18] recently introduced a new idea to represent spatial information in computer vision, which is especially suited for deep learning. Choosing a fixed set of basis points once and measuring the distances relative to this set for all new environments does not have the problem of varying permutations and lengths as point clouds have. It is also far more efficient in terms of memory and computation than voxel grids, allowing fast training in high-resolution 3D environments.

Formally, the BPS is an arbitrary but fixed set of points

$$B = [b_1, \dots, b_{N_b}], \quad b_i \in \mathbb{R}^d. \quad (10)$$

The feature vector W_B passed to the network consists of the distances to the closest point in the environment for all basis points. If the environment is defined by a point cloud W_P this can be calculated by

$$W_B = [\min_{x_i \in W_P} |b_1 - x_i|, \dots, \min_{x_i \in W_P} |b_{N_b} - x_i|]. \quad (11)$$

Alternatively, if the environment is given by an occupancy grid or a distance field D like we used for OMP in Section II, one can directly look up the feature vector

$$W_B = [D(b_1), \dots, D(b_{N_b})]. \quad (12)$$

With the second approach, it is possible to use signed distances. This adds directly a notion of inside and outside to the representation of the world.

Before using this representation to train a neural network, in Fig. 6 we analyze its properties with regard to efficiency and safety in a setting without learning.

Although an occupancy grid directly represents the cartesian workspace, the mapping to a robot’s configuration space can be complex. But the planning network has to understand this mapping to connect movements in joint space with its implications in the world to be able to plan collision-free paths. In Section VI, we show in experiments that the basis point set enables to learn that connection for complex robot kinematics and diverse environments.

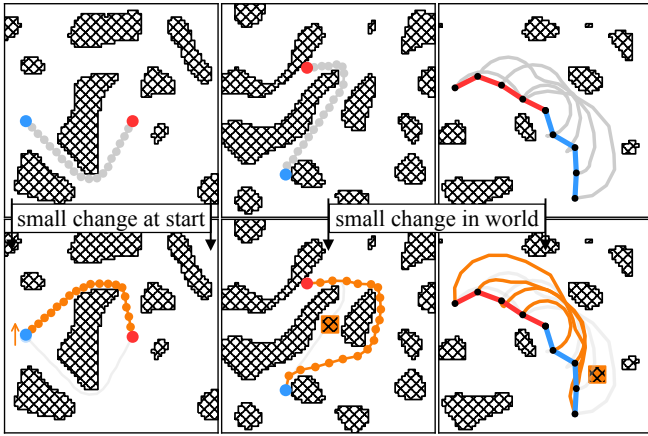


Fig. 8. Three examples of the networks’ prediction for only slightly altered problems. The changes in the start configuration (left) and the environment (middle, right) are highlighted in orange. The networks’ predictions after optimization for those new problems are shown in the bottom row, indicating they can react sharply to small input changes. For easier visibility in 3D, we refer to animations for the other robots on the website

V. EXPERIMENTAL SETUP

A. Dataset

We constructed extensive datasets for different robots (see Figs. 1, 2 and 8) to validate our methods. For 2D, we investigated a simple sphere robot with 2DoF and a serial arm with 4DoF. Furthermore, we used two real 3D robots: the LWR III [21], a robotic arm with 7DoF, and humanoid robot DLR Agile Justin [1] with 19DoF distributed over an upper body and two arms.

To generate diverse and challenging worlds, we used simplex noise [22]. This gradient noise is used in video games to create random but naturally-looking levels. A typical example is a continuous height map. By changing the cut-off threshold and the resolution of this noise, we can vary the density and form of the obstacles in the binary occupancy grid. To ensure that the environments are not too densely packed with obstacles, at least 200/1000 random robot configurations q must be feasible to include a world in the dataset. Examples can be seen in Figs. 1 and 2.

To generate the labels for the samples, we used the OMP approach described in Section II, with naive gradient descent and fixed step size. The paths consist of $N_t = 20$ waypoints, and to make the dataset challenging, we only included hard tasks that were not solvable starting from a straight line as initial guess. All other paths were discarded as too easy. We used up to 100 multi-starts and always picked the shortest feasible solutions as the correct label. The heuristic for generating the initial guesses was to use 1 to 3 random points in the configuration space and connect them linearly with the start and endpoint. See Table I for an overview of the robots and the datasets². The table also shows initially mislabeled paths and paths specifically included to challenge the network.

²All datasets plus additional information on their generation and use in training are provided at <https://dlr-ai.rwth-aachen.de/2022-iros-planning>.

TABLE I

OVERVIEW OF THE DATASETS AND NETS FOR THE DIFFERENT ROBOTS.

	Sphere Bot	Flat Arm	LWR III	Agile Justin
DoF	2	4	7	19
World size	$10 \times 10 \text{ m}^2$	$1.0 \times 1.0 \text{ m}^2$	$1.2 \times 1.2 \times 1.2 \text{ m}^3$	$3 \times 3 \times 3 \text{ m}^3$
Grid dimensions	64×64	64×64	$64 \times 64 \times 64$	$64 \times 64 \times 64$
# Worlds	10^4	10^4	10^4	10^4
# Paths	0.6×10^6	6.5×10^6	2.2×10^6	3.7×10^6
Avg. time p. core	0.1 s	0.8 s	3.1 s	8.4 s
# Improvements	0.1×10^6	0.3×10^6	0.2×10^6	0.3×10^5
# Extensions	0.5×10^5	1.5×10^5	3.0×10^5	5.0×10^5
Avg. $U_L(Q)$	1.589	1.7136	1.551	1.483
Avg. Feas. ϕ	67.3 %	32.6 %	54.5 %	44.1 %
Avg. time p. core	0.1 s	0.8 s	3.1 s	8.4 s
Net				
# In \rightarrow # Out	516 \rightarrow 39	520 \rightarrow 72	2062 \rightarrow 126	2086 \rightarrow 342
$ W_B $	512	512	2048	2048
N_t	18	18	18	18
# Parameters	3.4×10^6	7.5×10^6	2.4×10^7	4.1×10^7

B. Network

The last lines of Table I show the network details in numbers and Fig. 4 displays the general architecture. All the networks were trained purely supervised with a mean squared error between the predicted path Q_p and the label Q as loss function. As the encoding for the path, the deviation from the straight line is used. This representation implies that even an untrained network producing only random noise around zero can make meaningful predictions. For start and end, we use the normalized joint vectors q_1 and q_{N_t} as input. As environment encoding, we use the BPS described in Section IV with a hexagonal closed packing and only consider points inside the robots’ maximal reach. See Fig. 7 for an analysis of the dependency of the prediction quality on the size of the BPS.

VI. EVALUATION RESULTS

From the 10000 environments we generated for each robot, we used 9000 for the network’s training and the remaining unseen worlds for testing. All the results in Section VI are based on this unseen test set with 10000 hard motion tasks. As a quality measure we report the feasibility rate ϕ , i.e., the quotient of the number of feasible paths and the size of the test set.

A. Dataset Adaption

Table II shows the influence of the methods for dataset adaption during training as discussed in Section III. As metric we use the feasibility rate ϕ of the predicted paths after further iterations with the OMP as described in Section II. First, we compare the hard and the easy dataset. Because the easy dataset consists only of paths produced from straight lines, the overall variance is too slight, and the network does not learn to avoid the obstacles. This network is not able to solve the test set of hard examples. Next, we introduced the different modes of data augmentation to increase the size of the dataset. The temporal and spatial symmetries improve the feasibility rate ϕ without additional computing costs. The number of cleanings describes how often the labels were updated with the help of the neural network. Each iteration brings the labels closer to the optimal solution and makes

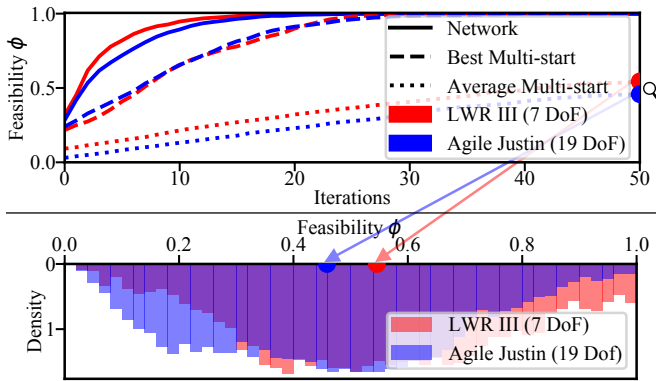


Fig. 9. Top: Average convergence to feasibility ϕ of OMP for different initial guesses. The prediction of the network outperforms the average and even the best out of 100 multi-starts significantly. Bottom: Distribution of the average feasibility ϕ of the random multi-starts after 50 OMP iterations.

the dataset more consistent, leading to better results. At this stage, the network performs already well with a success rate of over 85 %, but there are still tasks the net can not solve. We add the boosting technique to overrepresent more challenging samples during training to increase the feasibility further. As the final step, we use the trained network to generate more challenging samples. With this approach, we achieved 100 % feasibility on the hard unseen test set.

B. Comparison to Random Multi-start

The capabilities of our method become apparent when we compare the prediction of the network to the heuristic with random multi-starts used to create the dataset. In Fig. 9 the convergences to a feasible path of different initial guesses are displayed for the LWR III and Agile Justin. Without any experience, the best one can do, is to try random multi-starts and hope one of them converges. From the 100 multi-starts we used per task, only 50 % converge to a feasible solution after 50 iterations. Even the lucky initial guess, which converged the fastest for each problem, gets outperformed by the network’s prediction. The crucial difference is that our network does not depend on chance but can reliably predict initial guesses that converge after a few iterations to a feasible solution.

The actual speed gain is even more prominent when looking at the distribution over different motion tasks (see bottom of Fig. 9). There are problems for the LWR III and Agile Justin where only 10 % or less of all multi-starts converge to a feasible path. If one wants to find a solution to such a problem with 90 % confidence, one would need more than $\log(1 - 0.1) / \log(1 - 0.9) > 20$ multi-starts, making the initial guess of the network effectively over 20 times faster.

On our test machine (Intel i9-9820X @ 3.30 GHz, 32 Gb RAM), a single iteration of gradient descent for one path of Agile Justin takes 10 ms on a single core. Using the network’s prediction as a warm-start and stopping each sample after convergence leads to an average run time of $182(\pm 29)$ ms with a worst-case of 334 ms.

TABLE II
ANALYSIS OF DIFFERENT DATASET DISTRIBUTIONS AND EXTENSIONS & TRAINING METHODS FOR AGILE JUSTIN.

# Cleans	Dataset		Training		Feasibility ϕ	
	Distribution	Aug.	Boost	Network	+OMP	
0	easy		no	no	0.042	0.347
0	hard		no	no	0.126	0.653
0	hard	axis	no		0.133	0.691
0	hard	time	no		0.138	0.736
0	hard	both	no		0.143	0.772
0	hard	both	yes		0.171	0.859
1	hard	both	yes		0.196	0.893
2	hard	both	yes		0.217	0.925
3	hard	both	yes		0.223	0.941
3	hard + ext.	both	yes		0.283	1.00

C. World Encoding

Prokudin et al. [18] demonstrated that the BPS with fully connected layers is superior to occupancy maps with CNNs or point clouds with a PointNet architecture, both in terms of required network parameters and training performance. We can confirm those findings for motion planning. The large memory requirements in 3D made training prohibitively slow and hard to iterate on network architecture or training methods. Furthermore, looking towards the application on Agile Justin, the BPS can readily be integrated with the high-resolution SDFs acquired from the robot’s depth camera [23].

The BPS representation and the proposed training scheme on the worlds from simplex noise were robust enough to even generalize to some first results on a real robot (see Fig. 10). Only trained on those random worlds, the network was able to make valuable predictions from the data collected by Agile Justin’s depth camera [23]. The predictions as warm-start for OMP could solve the unseen motion tasks which needed multi-starts otherwise in under 200 ms.

Learning-based motion planning for such a complex robot was not tackled before in unseen environments, so we only compare it against simpler problems. MPNetPath, for example, takes 0.59 s to plan for the 7 DoF Baxter arm in a known table scene [9]. Non-learning-based methods like CHOMP take multiple seconds for similar scenes [5].

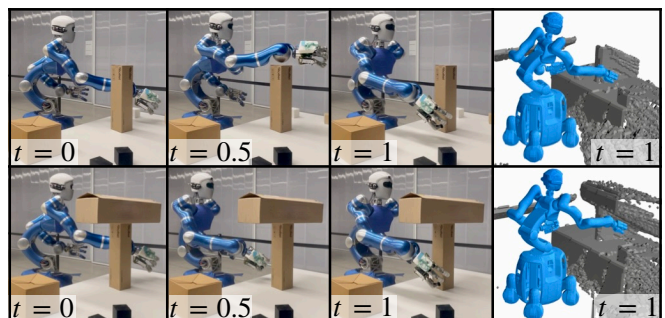


Fig. 10. Agile Justin in two table scenes with boxes. The robot applies a different strategy for obstacle avoidance after the top route is blocked. The rendered voxel model on the right shows the input for the neural network.

Fig. 8 shows a qualitative analysis of the network predictions. In motion planning, small changes in the problem often lead to fundamentally different solutions. Our worlds and training were challenging enough that the networks react sharply to small changes in the input, predicting completely different solutions to only slightly altered problems.

VII. CONCLUSIONS

We successfully trained motion planning networks using supervised learning on diverse and challenging datasets that predict paths close to the global optimum for previously unseen environments. Using this prediction as a warm-start for optimization-based motion planning massively outperforms random multi-start. For the complex robot Agile Justin with 19 DoF, planning takes only 200 ms on a single CPU core. This shows for the first time that learning-based motion planning works in previously unseen environments for such a complex robot.

One key to success is the basis point set encoding for the environment borrowed from computer vision which we introduced to motion planning and scales well to high-resolution 3D worlds. In addition, we autogenerate a training dataset of hard examples, i.e., situations for which the vanilla OMP struggles and for which the trained network should later provide an educated initial guess. We also introduced a scheme to further adapt the dataset during training by cleaning, boosting, and extending the dataset based on a metric defined by the (current) neural network and the objective function of the OMP. This approach leads to a challenging and consistent dataset on which a network can efficiently be trained and improved.

In the future, we extend the planning problem towards manipulation and grasping by incorporating the inverse kinematics so that no longer a goal configuration but only the goal pose of the end-effector has to be provided. We will also further investigate and increase the real-world capabilities of our method. As it is expensive to create a vast amount of real-world data, the goal is that our architecture and the autogenerated dataset allow for a robust transfer of the experience to real scenes.

REFERENCES

- [1] B. Büuml *et al.*, “Agile justin: An upgraded member of DLR’s family of lightweight and torque controlled humanoids,” in *Proc. IEEE International Conference on Robotics and Automation*, 2014.
- [2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [3] M. Zucker *et al.*, “CHOMP: Covariant Hamiltonian optimization for motion planning,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 8 2013.
- [4] M. Kalakrishnan *et al.*, “STOMP: Stochastic trajectory optimization for motion planning,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4569–4574, 2011.
- [5] J. Schulman *et al.*, “Motion planning with sequential convex optimization and convex collision checking,” *International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
- [6] N. Jetchev and M. Toussaint, “Fast motion planning from experience: Trajectory prediction for speeding up movement generation,” *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, 2013.
- [7] W. Merkt, V. Ivan, and S. Vijayakumar, “Leveraging Precomputation with Problem Encoding for Warm-Starting Trajectory Optimization in Complex Environments,” in *International Conference on Intelligent Robots and Systems (IROS)*, 10 2018, pp. 5877–5884.

- [8] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, “Motion Planning Networks,” in *International Conference on Robotics and Automation (ICRA)*, vol. 2019-May, 2019, pp. 2118–2124.
- [9] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, “Motion Planning Networks: Bridging the Gap between Learning-Based and Classical Motion Planners,” *IEEE Transactions on Robotics*, vol. 37, pp. 48–66, 2021.
- [10] R. Strudel *et al.*, “Learning Obstacle Representations for Neural Motion Planning,” in *Conference on Robot Learning (CoRL)*, 2020.
- [11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3D classification and segmentation,” *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 77–85, 2017.
- [12] M. J. Bency, A. H. Qureshi, and M. C. Yip, “Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- [13] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, “Memory of Motion for Warm-Starting Trajectory Optimization,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.
- [14] P. Lehner and A. Albu-Schäffer, “The repetition roadmap for repetitive constrained motion planning,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3884–3891, 2018.
- [15] T. Jurgenson and A. Tamar, “Harnessing Reinforcement Learning for Neural Motion Planning,” in *Robotics: Science and Systems XV. Robotics: Science and Systems Foundation*, 6 2019, pp. 1–13.
- [16] M. Pandey, D. Lenton, and R. Clark, “Unsupervised Path Regression Networks,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 9 2021, pp. 1413–1420.
- [17] D. Surovik *et al.*, “Learning an Expert Skill-Space for Replanning Dynamic Quadruped Locomotion over Obstacles,” in *Conference on Robot Learning*, 2020.
- [18] S. Prokudin, C. Lassner, and J. Romero, “Efficient Learning on Point Clouds with Basis Point Sets,” in *International Conference on Computer Vision (ICCV)*, 8 2019, pp. 4332–4341.
- [19] S. Jégou *et al.*, “The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation,” *CoRR*, vol. abs/1611.09326, 2016.
- [20] C. Chamzas, A. Shrivastava, and L. E. Kavraki, “Using Local Experiences for Global Motion Planning,” in *International Conference on Robotics and Automation (ICRA)*, vol. 2019-May. IEEE, 5 2019, pp. 8606–8612.
- [21] G. Hirzinger *et al.*, “DLR’s torque-controlled light weight robot III - are we reaching the technological limits now?” in *Proc. IEEE International Conference on Robotics and Automation*, 2002, pp. 1710–1716.
- [22] S. Gustavson, “Simplex noise demystified,” Linköping University, Sweden, Tech. Rep., 2005.
- [23] R. Wagner, U. Frese, and B. Büuml, “3D modeling, distance and gradient computation for motion planning: A direct GPGPU approach,” in *2013 IEEE International Conference on Robotics and Automation*, 2013.