2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Learning to Guide Random Tree Planners in High Dimensional Spaces

Jörg Röwekämper Gian Diego Tipaldi Wolfram Burgard

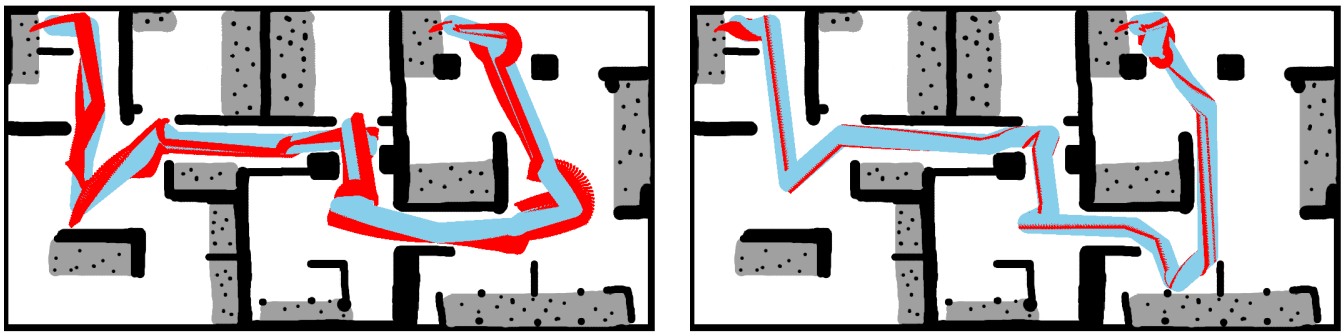


Fig. 1. Example paths for a mobile manipulation platform computed with RRT-Connect [13] (left) and our approach (right). Red indicates the area covered by the robot manipulators and blue that of the robot base. Note that our approach returns smoother paths with fewer movements of the manipulators.

***Abstract*— In this paper we present the projection and bias heuristic (PBH), a motion planning algorithm that makes use of low-dimensional projections to improve sampling-based planning algorithms. In contrast to other state-of-the-art methods, we do not assume that projections are either random or given by an expert user. Rather, our goal is to *learn* projections such that planning on them improves the efficiency and the quality of solutions. We present both, a method to learn effective projections and a sampling algorithm that makes use of them. We show that our approach can be easily integrated into popular sampling-based planners. Extensive experiments performed in simulated environments demonstrate that our approach produces paths that are in general shorter than those obtained with state-of-the-art algorithms. Moreover, it generally requires less computation time.**

I. Introduction

Path planning is a key requisite to obtain the desirable navigation autonomy for mobile robots. With the increasing complexity and structure of robots, one needs to consider high dimensional state spaces. This is especially true when considering mobile manipulation tasks, that require to plan for the mobile platform and the manipulator at the same time.

Given the high dimensionality, researchers started to move away from the classical paradigm of finding global optimal plans and started to explore randomized algorithms to trade efficiency with optimality. The randomized path planner (RPP) [2] can be considered the first work in this direction while the probabilistic roadmap planner (PRM) [11] can be considered the pioneer of current sampling based algorithms. The idea of sampling was also introduced for single-query tree-based planners with expansive space trees (EST) [8], and one of the most popular algorithms is represented by the rapidly exploring random tree (RRT) algorithm [15, 14]. The main idea is to grow a tree of motions from the current configuration of the robot (start) to a desired configuration (goal), depending on the performed task. The original idea of RRT has been extended and several effective heuristics to speed up the search have been proposed [22, 20, 16].

However, when the state space grows, deciding which part of the tree to expand and in which direction is not trivial. To overcome this problem, recently proposed approaches use a low-dimensional projection to guide the random sampling and mitigate the curse of dimensionality. For example, a common approach for mobile manipulators is to first plan motions in the low-dimensional projection of the mobile base towards the goal and use the full joint space only when the current configuration in the workspace is close to the goal. Although effective in most of the cases, this solution is highly engineered and requires the user to define valid projections and heuristics on when to switch between them. Automatically finding a good projection is still an open problem [19, 21].

In this paper, we aim at relaxing the need for an expert user to define valid heuristics and projections for different kinds of configuration spaces. Although valid solutions have been proposed for well known kinematic chains, no general and principled technique is known on how to adapt those heuristics to different robots or how to learn them from data. We strongly believe that it is not necessary to explore the whole state space every time, but that certain regions of the state space are more important for certain tasks or in particular steps of the planning phase. By allowing the search to be performed only in those regions, we are able to increase the efficiency of sample-based planning. This idea shares some grounds with the task and motion planning approach of Sucan *et al.* [19], where several planning problems on different configuration subspaces are run in parallel and the algorithm gives more computational resources to the most promising one. In this work, instead of trying different

All authors are with the Department of Computer Science at the University of Freiburg, Germany.

This work has partly been supported by the German Reserach Foundation under contract number EXC1086 and by the European Commission under contract number ERC-267686-LifeNav.

configurations, we aim at learning a function that returns an effective projection and sampling bias. The function is learned from a set of randomly generated planning problems by maximizing the success rate of planning, i. e., the ratio between rejected samples (due to collisions) and total generated samples.

We integrated our approach in both the RRT [15] and the RRT-Connect [13] algorithms and performed an extensive set of experiments using two simulated environments. We compared the performance of our approach with state-of-the-art planning algorithms for both unidirectional and bidirectional planners. The results show that our approach produces paths that are in general shorter than state-of-the-art algorithms and in less time in most of the cases.

II. Related Work

Sampling-based motion planning is an effective means to solve complex path planning problems and one of the most popular paradigms for planning in high dimensional spaces. Many different planning algorithms have been developed such as RRT [15], PRM [11], EST [8], kinematic planning by interior-exterior cell exploration (KPIECE) [18] and more. Works on projections for deterministic planning algorithms, such as [7, 5, 3], are also presented but are considered out of the scope of this work.

The RRT algorithm was introduced by La Valle and Kuffner [15]. Since then, several heuristics to improve the sampling have been developed. Urmson *et al.* [22] proposed a set of heuristic functions to bias the search towards low-cost solutions. The heuristic guidance is applied by weighting the probability density function used to build the random tree. Kuffner *et al.* [12] introduced the space-filling trees, the analogous of space-filling curves with a tree-like structure. The trees are defined by an incremental process to ensure that every point in the space is reachable by a finite length path. Bidirectional trees have also been introduced to improve efficiency [13]. The algorithm grows two separate trees: one from start to goal and another from goal to start. The motivation of the approach is that random trees tend to have a good coverage close to the starting point and a more sparse one in places far from them. By expanding two trees in parallel, one obtains a better coverage of the space and a faster algorithm. Karaman *et al.* [9, 10] modified the sampling algorithm and proved some optimality guarantees for sampling-based planners. The resulting algorithm, RRT*, is able to eventually find the optimal solution given that enough samples have been drawn. To improve the performance of RRT* in high dimensional spaces, Agkun *et al.*presented a modification of the sampling heuristic for RRT* [1] to address its limitations in high-dimensional configuration spaces. They introduced both a sampling bias and a simple node-rejection criterion to increase efficiency. Sucan *et al.* [20] presented a comprehensive survey on recently developed heuristics.

More related to the approach presented here are works on planning on state space projections to improve efficiency and coverage. Projections are used for example in KPIECE [18] to approximate the coverage of the configuration space. The authors employed a multi-level grid-based discretization to estimate the coverage of the state space and focus the exploration on less covered areas. While in the original approach projections were assumed to be given by a user, in a successive work [21] the performance of commonly used projections are compared to random projections. The results showed than random projections perform on par or even better that user-defined ones, while being easier to obtain. In a similar way, researchers explored the use of workspace projections and decompositions instead of state space ones. In the synergistic combination of layers of planning algorithm (SyCLoP), Plaku *et al.* [17] decompose the workspace and connect the resulting regions in a graph to encode physical adjacency. This graph is then searched to obtain sequences of regions that can be explored with sampling-based tree methods to obtain a final solution. Maly *et al.* [16] extended SyCLoP to perform the high-level decomposition over a given low-dimensional projected subspace. They evaluated the performance of SyCLoP with random projections, user provided ones and linear dimensionality reduction. Brock and Kavraki [4] propose to capture the relevant connectivity of the free space in a low-dimensional projection and then plan for the degrees of freedom in the original space. The projections used in those approaches, however, are either random or rely on expert users to provide them. On the contrary, the focus of this paper is to *learn* such projections in a way that planning on them simultaneously improves solution efficiency and quality.

From a different perspective, but closely related to projections for efficient planning, Sucan *et al.* introduced the concept of task and motion planning [19]. They decompose the planning problem into a set of smaller problems of reduced dimensionality and perform planning for a limited time on each of those problems. Furthermore, they use a high-level task planner to select which problem to consider next. Our approach follows a similar idea but applies directly in the motion planning algorithm. Instead of using high-level reasoning to decide which projection to use, we learn a function that returns the best one at any state in the configuration space.

III. Planning with Low-Dimensional Projections

We consider a motion planning problem p as a search problem in the configuration space $\mathcal{X}$ of a robot $\mathcal{R}$ aiming at finding a valid path in the free space $\mathcal{X}_{free} \subseteq \mathcal{X}$ from an initial state $\mathbf{x}_{start}$ to a goal state $\mathbf{x}_{goal}$.

In the remaining of the paper, we use a bi-manual mobile manipulator as a motivational example and follow the notation of [19] to describe the state space and its projections. We further restrict ourselves to consider a special class of projections, i. e., the ones implicitly defined by a subset of the available joints of the robot.

Let J be the set of joints of $\mathcal{R}$ and $\mathcal{X}_J$ the associated state space. Let $\mathbf{J} = \{J_i \mid J_i \in 2^J \setminus \emptyset\}$ the set of all possible combinations of joints in J, where 2^J is the power set of J and each subset J_i defines a low-dimensional space $\mathcal{X}_i$. Let further $\mathbf{J}_a \subseteq \mathbf{J}$ define the possible set of joints to plan

Algorithm 1 Expand Tree

T_k = **expandTree**(T_{k-1})
1: $\mathbf{x}_r \leftarrow Sample(\mathcal{X}_J)$
2: $\mathbf{x}_n \leftarrow SelectNode(\mathbf{x}_r)$
3: $(J_{active}, J_{passive}, \mathbf{x}_{bias}) \leftarrow H(d(\mathbf{x}_n, \mathbf{x}_{goal}))$
4: $\mathbf{x}_{active} \leftarrow ExpandLazy(\mathbf{x}_n, \mathbf{x}_r, J_{active})$
5: $\mathbf{x}_{passive} \leftarrow ExpandLazy(\mathbf{x}_n, \mathbf{x}_{bias}, J_{passive})$
6: $\mathbf{x} \leftarrow Lift_{active}(\mathbf{x}_{active}, \mathbf{x}_{passive})$
7: **if** $Collide(\mathbf{x})$ **then**
8: **return** T_{k-1}
9: **else**
10: **return** $GrowTree(T_{k-1}, \mathbf{x})$

motions from start to goal. Let $Project_i : \mathcal{X}_J \rightarrow \mathcal{X}_i$ be the function that *projects* a point in a low-dimensional space and $Lift_i : \mathcal{X}_i \times \mathcal{X}_J \setminus \mathcal{X}_i \rightarrow \mathcal{X}_J$ be the function that *lifts* a point from a low-dimensional space to the original one. Given $\mathbf{x} \in \mathcal{X}_i$ and $\mathbf{y} \in \mathcal{X}_J \setminus \mathcal{X}_i$, we define $\mathbf{z} = Lift_i(\mathbf{x}, \mathbf{y})$ to be the state that has the same values as $\mathbf{x}$ for the joints in J_i and the same value as $\mathbf{y}$ for the joints in $J \setminus J_i$. A common assumption is that $\mathbf{J}_a$ is provided by the user and contains a limited number of joint sets. Moreover, a random tree is grown in each $J_j \in \mathbf{J}_a$ by iteratively projecting and lifting a high-dimensional sample onto the low-dimensional space and vice-versa. This is the approach used in [19].

In this paper, we relax the assumptions that valid projections must be provided by the user and avoid that the planning algorithm must switch between each of these projections. Let us assume for now that there is a function

$$H : \mathbb{R}_{\geq 0} \rightarrow \mathbf{J} \times \mathcal{X}_J \tag{1}$$

that computes a projection $J_{active} \in \mathbf{J}$ and a configuration bias $\mathbf{x}_{bias} \in \mathcal{X}_J$ given the distance $d(\mathbf{x}_{curr}, \mathbf{x}_{goal})$ between the current configuration of the robot and the goal configuration. The idea behind this function is that by 1) growing the tree in the low-dimensional space J_{active} and 2) guiding the sampling towards $\mathbf{x}_{bias}$ in $J \setminus J_{active}$, we can reduce the number of samples needed to reach the goal. In the next section, we describe how such a function can be learned from randomly generated planning problems.

This function can be easily integrated in any tree-based sampling algorithm. We choose to integrate it within both RRT [15] and RRT-Connect [13]. We replaced the sampling method of RRT with the following algorithm, whose pseudocode is shown in Algorithm 1. First, a node in the current tree is selected for expansion by sampling a random state $\mathbf{x}_r$ from $\mathcal{X}_J$ (1) and selecting the closest state $\mathbf{x}_n$ in the tree (2). Then, we compute the distance between $\mathbf{x}_n$ and $\mathbf{x}_{goal}$ to compute J_{active}, $J_{passive} = J \setminus J_{active}$ and $\mathbf{x}_{bias}$ from $H(d(\mathbf{x}_n, \mathbf{x}_{goal}))$. We project $\mathbf{x}_n$ and $\mathbf{x}_r$ onto the low-dimensional space J_{active}, compute $\mathbf{x}_{active}$ and connect it to the tree as in the standard RRT algorithm. A similar expansion is then performed in $J_{passive}$, where we project $\mathbf{x}_n$ and $\mathbf{x}_{bias}$ instead and compute $\mathbf{x}_{passive}$. The two projected trees are then lifted to the original space $\mathcal{X}_J$ using $Lift_{active}(\mathbf{x}_{active}, \mathbf{x}_{passive})$. The sample is finally accepted if no collision is detected in $\mathcal{X}_J$. To account for probabilistic completeness of the algorithm, we sample in the full configuration space with probability 0.05.

Algorithm 2 Rank Low-Dimensional Projections

$\hat{J}$ = **rankProjections**$(N_p, J, a_i, a_{i+1}, t)$
11: $P \leftarrow GenerateRandomProblemsFreeSpace(N_p, a_i, a_{i+1})$
12: $\hat{J} \leftarrow \emptyset$
13: **for all** $\mathbf{J}_i \in \mathbf{J}$ **do**
14: $n_{total} \leftarrow SolveLowDimensionalProblems(P, J_i, N_r, t)$
15: **if** $n_{total} \neq null$ **then**
16: $\hat{J} \leftarrow \hat{J} \cup \{(J_i, n_{total})\}$
17: **return** $SortProjections(\hat{J})$

For RRT-Connect, we only modified the sampling method, similarly to the RRT, without changing the connection strategy. The only difference is that the role of $\mathbf{x}_{start}$ and $\mathbf{x}_{goal}$ is switched for the start tree.

IV. Learning Low-Dimensional Projections

In this section we describe our algorithm to learn the function $H : \mathbb{R}_{\geq 0} \rightarrow \mathbf{J} \times \mathcal{X}_J$ from data. Our aim is to obtain a generic solution that requires as little external prior information from an expert user as possible. The only components required by our approach are:

- A model of environment in which we can simulate a set a planning problems.
- A collision checking module that tests whether a configuration collides with obstacles.
- A distance function $d(\mathbf{x}_i, \mathbf{x}_j)$ between states $\mathbf{x}_i$ and $\mathbf{x}_j$.

To make the learning process computationally feasible, we discretize $\mathbb{R}_{\geq 0}$ in a set of intervals and assume H to be a piecewise constant function on those intervals. Given a minimum size for the intervals a_{min} and a number of intervals k, we chose the intervals to be $[a_i, a_{i+1})$, where $a_0 = 0$, $a_i = a_{min} 2^{i-1}, i = 1, \cdots, k-1$ and $a_k = \infty$. By exploiting the dynamic programming principle, it is possible to learn the function independently for each interval. Suppose for any state $\mathbf{x}$ with $d(\mathbf{x}, \mathbf{x}_{goal}) < a_i$ there exist a motion to reach the goal q_{goal} using some H. This is true if we suppose that a solution to the planning problem using the full configuration space exists and have $H(\cdot) = (J, \mathbf{x}_{goal})$. Then we only need to learn the function H to compute J_{active} and $\mathbf{x}_{bias}$ such that we can reach some state $\mathbf{x}_n$ with $d(\mathbf{x}_n, \mathbf{x}_{goal}) < a_i$ from any state $\mathbf{x}$ with $a_i \leq d(\mathbf{x}, \mathbf{x}_{goal}) < a_{i+1}$.

The learning process is composed of two parts. In the first part, we simulate a set of planning problems in free space to obtain a ranking of the set of projections $\mathbf{J}$. We then simulate a set of planning problems in an environment with obstacles to select the best projection J_i and bias $\mathbf{x}_{bias}$ that minimize the number of rejected samples due to collisions.

A. Ranking Low-Dimensional Projections in Free Space

For each interval we first employ Algorithm 2 to rank the set of projections $\mathbf{J}$. This will be used in the next step to speed-up the search of the best pair of projection and bias. The algorithm proceeds as follow. First we generate a set of N_p random planning problems P in absence of obstacles for the interval $[a_i, a_{i+1})$, that is with starting configuration $\mathbf{x}_{start}$ s.t. $a_i \leq d(\mathbf{x}, \mathbf{x}_{start}) < a_{i+1}$ and

Algorithm 3 Select Bias and Projection

$(J_i, \mathbf{x}_{bias})$ = **learnFunction**$(N_p, N_{bias}, N_r, \hat{J}, a_i, a_{i+1}, t)$
$P \leftarrow GenerateRandomProblems(N_p, a_i, a_{i+1})$
repeat
  $(J_{active}, J_{passive}) \leftarrow SelectBestProjection(\hat{J})$
  $Q \leftarrow GenerateRandomStates(N_{bias}, J_{passive})$
  for all $\mathbf{x} \in Q$ **do**
    $n_{success} \leftarrow SolveProblems(P, J_i, \mathbf{x}, N_r, t)$
    if $n_{success} \neq null \wedge n_{success} > n_{best}$ **then**
      $n_{best} \leftarrow n_{success}$
      $\mathbf{x}_{bias} \leftarrow \mathbf{x}$
  $\hat{J} \leftarrow \hat{J} \setminus J_{active}$
until $\mathbf{x}_{bias} \neq null$
return $(J_{active}, \mathbf{x}_{bias})$

termination condition $d(\mathbf{x}, \mathbf{x}_{goal}) < a_i$. We then solve each planning problem in P, with an RRT planner in the low-dimensional space X_i, for each joint set $J_i \in \mathbf{J}$ for N_r times. When a solution is found, we return the number of tree expansion steps n_{total} that the RRT performed and add the pair (J_i, n_{total}) to the set $\hat{J}$. If no solution is found after a time t we abort the search and discard the projection J_i. The set $\hat{J}$ is then sorted according to n_{total}.

B. Computing the Best Projection and Configuration Bias

Once the possible projections have been computed and sorted, a similar algorithm is performed to obtain the best combination of projection and bias (Algorithm 3). We again consider each interval separately and start by generating a set of N_p random planning problems P in the presence of obstacles for the interval $[a_i, a_{i+1})$. The algorithm then iterates until a valid pair $(J_{active}, \mathbf{x}_{bias})$ is found. We first extract the best J_{active} from $\hat{J}$. We then generate a set Q of N_{bias} random states $\mathbf{x}_{bias}$ for the joints in $J_{passive} = J \setminus J_{active}$ as candidate biases. Then, we solve each planning problem in P with an RRT by only sampling in the low-dimensional space X_i and fixing the remaining joints on $\mathbf{x}_{bias}$ for each bias $\mathbf{x}_{bias} \in Q$ for N_r times. When a solution is found, we return the success rate of the planner $n_{success}$, i.e., the ratio between the number of times the tree was expanded and the number of total trials. This number represents the success rate of a random sample to expand the tree in a collision free area. If no solution is found after a time t we abort the search and discard the projection J_i. Otherwise, we return the pair $(J_{active}, \mathbf{x}_{bias})$, whose $\mathbf{x}_{bias}$ reported the maximum success rate.

V. Experiments

We performed an extensive set of experiments using two simulated environments: office-like and maze-like. A third environment was used to learn the projections and the biases. As robotic platform, we simulated an omnidirectional mobile manipulator with a cylindrical base and two manipulators with 5 degrees of freedom each, resulting in a 13 DOF in total. The simulated environments are modeled as 2.5D with size of $20\,m$ by $10\,m$: the manipulators are constrained to planar motion and obstacles can collide with the base only, the arms only or all of them. Figure 3 shows the environment we used during the learning phase, while Figures 5 and 6 show the office-like and the maze-like environments respectively. The start and goal position for the experiments are also indicated in the figures.

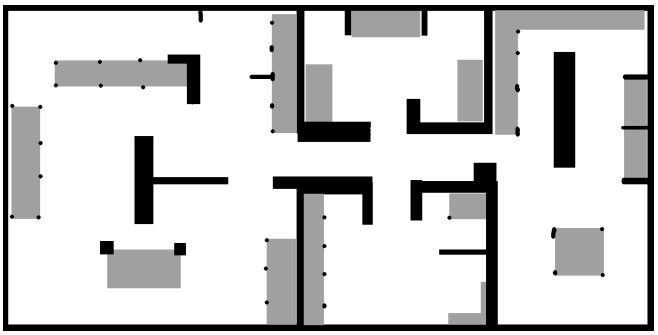
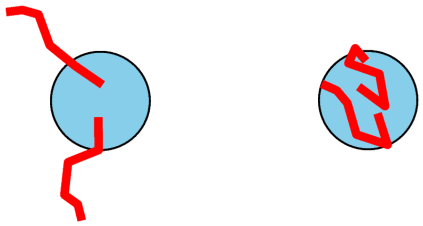


Fig. 3. Environment used to learn the heuristic function.



$n_{success} = 0.48$ $n_{success} = 0.68$

Fig. 4. Examples of bad (left) and good (right) configurations of x_{bias}, learned for the passive set $J_{passive}$. The base of the robot is depicted in blue, while the arms in red. Note that the system implicitly learned that folding the arms lead to a reduced footprint and a better chance of success.

Different planning problems with different characteristics were tested. Scenarios A→B and B→A represent problems where the start and goal configurations are very far away in the state space. In the scenarios A→C and C→B we tested how the planner performs if the start or goal configurations are in a restricted area where several maneuvers are needed to avoid collisions. Finally, D→E represents scenarios of middle distances and F→G scenarios where the movement of the base is very minimal.

We used the Open Motion Planning Library [6] as a reference implementation to compare our approach with standard planners from the library such as KPIECE, bi-directional KPIECE (BKPIECE), RRT, RRT*, and RRT-Connect. All experiments were performed on the same desktop PC. We ran each planning problem 50 times and averaged the results. If a planning problem did not finish after $t = 10\,min$ we abort the search and did not use the results for computing the average. For all planners we used a goal bias of 5%. We used Euclidean distance in configuration space for the nearest neighbor search with a planner range set to 5. For KPIECE, we tried several projections to determine the coverage. The best results were produced by projecting the right end effector into the workspace and using a discretization of $0.25\,m$. We evaluated our approach with unidirectional (Table I) and bidirectional (Table II) planners.

For learning the proposed function H, we used the following values for the parameters: $N_p = 10$, $N_{bias} = 1500$, $N_r = 20$, $a_{min} = 0.25$, $k = 4$, and $t = 5\,min$. As a distance function $d(\mathbf{x}_i, \mathbf{x}_j)$ we used the Euclidean distance between

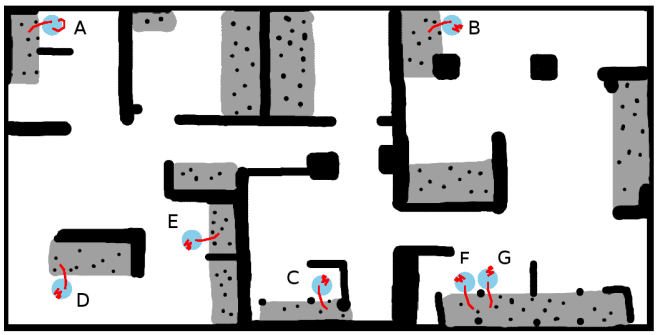


Fig. 5. Office environment used for comparison with the start and goal configurations for the different problems tested in the experiments.

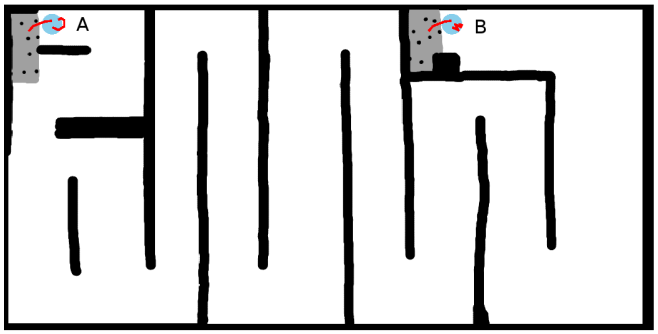


Fig. 6. Maze environment used for comparison with the start and goal configurations for the different problems tested in the experiments.

the end-effector pose of the right manipulator at the two states. Finally, the special $SE(2)$ space of the mobile base was considered as one joint with 3 DOF. In this scenario, the algorithm learned that the best projection up to $0.5\,m$ distance to the goal is the one given by only the base joints. In the interval $[0.25, 0.5)$ the best projection consist of the base, the first three joints of the right manipulator and the third joint of the left manipulator. When getting closer to the goal, the whole configuration space is used. Figure 4 shows two examples of x_{bias} for the interval $[1, \infty)$. The image on the right of the figure shows the best configuration with $n_{success} = 0.68$ and the one on the left an example of a poor configuration with $n_{success} = 0.48$.

A. Results

The results of our experiments for the office environment are shown in Table I and Table II. For PBH, the timings only report planning time and not learning time. We only learned H once in an offline phase and used he same H for all the problems and the environments. For the unidirectional case, KPIECE and our approach (RRT + PBH) are the only planners that returned a solution for all the cases in scenarios A→B and B→A. If we look at the resulting path length we see that RRT + PBH consistently provide shorter paths than both KPIECE and RRT. As for planning time, RRT + PBH is either the fastest planner (A→B) or the second fastest (B→A). The results for D→E are similar to B→A.

Scenario A→C shows some of the limitations of the proposed approach. The planner gets stuck in the confined space at the beginning and is not able to successfully return a valid path all the time. Its behavior is similar to that of RRT without projections but still better than RRT*. The coverage property of KPIECE lets it achieve a 100% success rate. The main reason behind the failure of PBH is the fact that the robot tends to move the arms towards the bias and this results on many samples being discarded.

TABLE I. Unidirectional planners map office environment

Scenario	Planner	Time (s)	Success (%)	Length
A→B	KPIECE	12.344	100	423.6
	RRT	9.277	16	124.2
	RRT*	25.075	26	70.9
	RRT + PBH	6.211	100	83.7
B→A	KPIECE	18.100	100	436.8
	RRT	197.700	34	153.7
	RRT*	245.272	20	70.9
	RRT + PBH	22.164	100	105.5
A→C	KPIECE	77.067	96	311.6
	RRT	79.361	8	98.04
	RRT*	-	0	-
	RRT + PBH	443.687	2	87.5
C→B	KPIECE	4.374	100	218.8
	RRT	45.303	34	90.0
	RRT*	16.086	26	45.7
	RRT + PBH	5.319	100	64.5
D→E	KPIECE	2.001	100	95.6
	RRT	80.580	26	36.1
	RRT*	165.773	14	20.1
	RRT + PBH	6.357	100	39.9
F→G	KPIECE	0.161	100	61.6
	RRT	-	0	-
	RRT*	-	0	-
	RRT + PBH	0.891	100	36.9

TABLE II. Bidirectional planners office environment

Scenario	Planner	Time (s)	Success (%)	Length
A→B	BKPIECE	49.088	100	675.0
	RRT-Connect	2.696	100	126.0
	RRT-Connect + PBH	1.401	100	73.8
B→A	BKPIECE	47.225	100	454.1
	RRT-Connect	2.714	100	125.7
	RRT-Connect + PBH	1.642	100	73.6
A→C	BKPIECE	47.551	100	454.1
	RRT-Connect	3.084	100	88.9
	RRT-Connect + PBH	2.829	100	60.4
C→B	BKPIECE	10.104	100	401.3
	RRT-Connect	1.320	100	77.9
	RRT-Connect + PBH	2.338	100	59.0
D→E	BKPIECE	0.343	100	137.5
	RRT-Connect	0.110	100	30.7
	RRT-Connect + PBH	0.168	100	22.9
F→G	BKPIECE	0.175	100	121.6
	RRT-Connect	0.028	100	21.8
	RRT-Connect + PBH	0.028	100	22.0

In Scenario C→B the start position is still confined but to a minor extent. There, RRT + PBH obtain a success rate of 100%, the same as KPIECE, while RRT and RRT* only have a success rate of 34% and 26%. RRT + PBH also return the shortest paths. In the last planning scenario F→G, no path was found for both RRT and RRT*, where our approach and KPIECE have a success rate of 100%.

Table II shows the results of bidirectional planners. All planners had a success rate of 100%. It should be noted that although the same projections were used, BKPIECE takes more time than KPIECE in most of the scenarios. This is probably due to the fact that both directions are covered and when start and goal are farther apart more time is needed. Another important aspect to note is that when we apply PBH to RRT-Connect, we obtain 100% success rate in all the scenarios. We believe that this is due to the fact that our

TABLE III. Bidirectional planners maze environment

Scenario	Planner	Time (s)	Success (%)	Length
A→B	BKPIECE	10.875	100	1058.8
	RRT-Connect	37.396	100	402.7
	RRT-Connect + PBH	5.745	100	146.5
B→A	BKPIECE	11.668	100	1068.3
	RRT-Connect	43.309	100	396.9
	RRT-Connect + PBH	6.730	100	147.7

heuristic now implicitly depends on both the distance to the goal (goal-tree) and to the start position (start-tree). With respect to planning time and path length, RRT-Connect + PBH consistently provides shorter paths and requires less time most of the times. Figure 1 shows two example paths returned by RRT-Connect and RRT-Connect + PBH for the scenario A→B. In the path returned by RRT-Connect the arms of the robot constantly move, resulting in a longer overall path. With our approach, the arms do not move much during motion and the footprint is minimized. Please note that no prior knowledge was used to force this behavior. Rather it has been learned from randomly generated planning problems.

Finally, to show how the learned heuristic generalizes to different environments we also evaluated the bidirectional planners in a maze environment (Table III). The advantage of J_{active} and x_{bias} becomes important in this case and the experimental results clearly demonstrate that. In this environment, RRT-Connect + PBH is always the fastest planner and provides the shortest paths.

B. Discussion

Although the experiments demonstrate that PBH can drastically improve the performance of both RRT and RRT-Connect, there are situations where its usage results in a decrease in performance. One example is provided by scenario A→C, where the planner could not reach the goal configuration in most of the cases. The learned projection J_{active} did not include the part of the joint space needed to obtain a valid path, since none of the random planning problems used for learning contained a scenario similar to A→C.

Please also note that the way KPIECE and our approach employ projections is also different. KPIECE uses the projection to estimate the coverage of the state space to decide where to extend the tree, while we combine it with RRT and RRT-Connect and still randomly sample a state and extend the tree from the nearest state. We mainly use the projection to have a selective bias and extend the tree towards the random sample in J_{active} and towards x_{bias} in $J_{passive}$. This results in KPIECE trying to explore the whole state space while our approach tends to explore only parts of it.

VI. Conclusion

In this paper we presented PBH, a planning algorithm that makes use of low-dimensional projections to improve sampling-based planning algorithms. In contrast to other methods, our approach does not rely on user-defined projections, but instead uses a heuristic function to obtain good projections according to the distance between the current and the goal configuration. We further show how such a function can be learned from a set of randomly generated planning problems and obtained by maximizing the success rate of planning. We integrated our approach into both the RRT and the RRT-Connect algorithms and compared it with state-of-the-art planners. The experimental results demonstrated that PBH produces paths that are in general shorter then those obtained with state-of-the-art algorithms and furthermore requires less time in most of the cases.

References

[1] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys. (IROS)*, 2011.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. Journal of Robotics Research*, 10(6):628–649, 1991.

[3] D. Berenson, P. Abbeel, and K. Goldberg. A robot path planning framework that learns from experience. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2012.

[4] O. Brock and L. E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2001.

[5] B. J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2010.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[7] K. Gochev, A. Safonova, and M. Likhachev. Planning with adaptive dimensionality for mobile manipulation. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2012.

[8] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 1997.

[9] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. of Robotics: Science and Systems (RSS)*, 2010.

[10] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the RRT*. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2011.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Rob. and Aut.*, 12(4):566–580, 1996.

[12] J. J. Kuffner and S. M. LaValle. Space-filling trees: A new perspective on incremental search for motion planning. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys. (IROS)*, 2011.

[13] J. J. Kuffner Jr and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2000.

[14] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. Journal of Robotics Research*, 20(5):378–400, 2001.

[15] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 1999.

[16] M. R. Maly and L. E. Kavraki. Low-dimensional projections for syclop. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys. (IROS)*, 2012.

[17] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[18] I. Şucan and L. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. *Algorithmic Foundation of Robotics VIII*, pages 449–464, 2009.

[19] I. A. Sucan and L. E. Kavraki. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2011.

[20] I. A. Sucan and L. E. Kavraki. On the implementation of single-query sampling-based motion planners. In *IEEE Int. Conf. on Rob. & Aut. (ICRA)*, 2010.

[21] I. A. Sucan and L. E. Kavraki. On the performance of random linear projections for sampling-based motion planning. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys. (IROS)*, 2009.

[22] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *IEEE/RSJ Int. Conf. on Intel. Rob. and Sys. (IROS)*, 2003.