 http://dx.doi.org/10.1109/ICRA48891.2023.10161339

Sample-Driven Connectivity Learning for Motion Planning in Narrow Passages

Sihui Li Neil T. Dantam

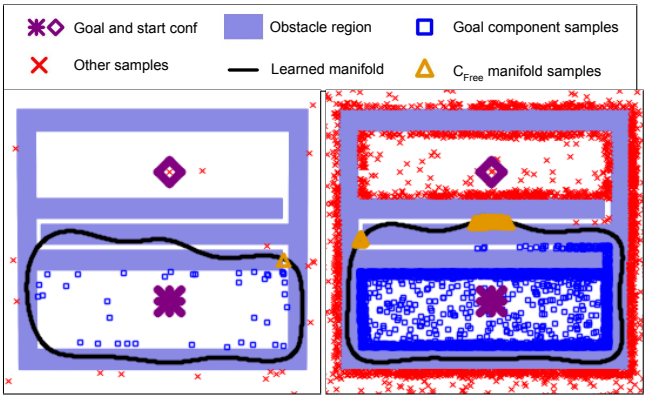
***Abstract*— Sampling-based motion planning works well in many cases but is less effective if the configuration space has narrow passages. In this paper, we propose a learning-based strategy to sample in these narrow passages, which improves overall planning time. Our algorithm first learns from the configuration space planning graphs and then uses the learned information to effectively generate narrow passage samples. We perform experiments in various 6D and 7D scenes. The algorithm offers one order of magnitude speed-up compared to baseline planners in some of these scenes.**

I. Introduction

Sampling-based motion planners are effective and widely-used tools for high-dimensional motion planning. Their key advantages are guarantees on convergence [1], [2] and a convenient problem formulation—essentially, requiring only a configuration space validity checker to apply to new scenarios. Typically using random sampling, these planners efficiently expand the search to cover the configuration space [3], [4], [5], [6], [7]. However, narrow passages along feasible paths increase planning difficulty and time. With uniform random sampling, the probability of sampling part of the free space is proportional to its volume. Although random sampling is generally effective, sampling in narrow passages— particularly in high-dimensional environments—presents challenges due to the small volume of narrow passages compared to the entire configuration space. The key to addressing this challenge is to infer locations of narrow passages from previous samples or local geometric information and then guide sampling towards those regions, as applied in previous works [8], [9], [10].

Machine learning offers the potential to gather information from previous samples, due to the generality of learning techniques and their effective use of modern GPU computing. Learning has been previously applied to improve planning performance [11], [12], [13], [14]. Most such prior approaches learn from workspace information, e.g., trajectories or visual data. In contrast, this paper explores *configuration space learning*. In previous work [15], we introduced a configuration space learning approach to prove motion planning infeasibility. Now, we show that the same learned structures used to construct infeasibility proofs also offer key information about configuration space connectivity and provide an effective heuristic to guide sampling in difficult motion planning problems containing narrow passages.

The authors are with the Department of Computer Science, Colorado School of Mines, USA. Email: {li, ndantam}@mines.edu. This work was supported in part by NSF IIS-1849348, ARL DCIST W911NF-17-2-0181, and Office of Naval Research grant N00014-21-1-2418.



(a) 76 nodes for training (b) 6769 nodes for training

Fig. 1: As the number of training nodes grows, the learned manifold is gradually "pushed" onto the narrow passage regions. As a result, samples on the manifold in $\mathcal{C}_{\text{free}}$ (identified by yellow triangles) reveal the narrow passage. New samples on the manifold are added to the planning graph and used for learning in the next iteration.

We introduce an algorithm to learn configuration space connectivity and guide sampling through narrow passages. We call the approach *sample-driven connectivity learning (SDCL)*, and we integrate the learning procedure with a Probabilistic Roadmap (PRM) [16]. SDCL has two major procedures, (1) learning a manifold and (2) sampling the manifold, which runs in parallel with the construction of PRM. This approach effectively leverages the increasing parallelism of modern multi-core and GPU computing in these two procedures.

The first step of SDCL is learning a binary classier from the planning graph—though we will not use this classifier in a typical fashion. Geometrically, the classifier is a manifold in the configuration space, that essentially learns the critical areas for connecting the separated graph components. The second step of SDCL is to sample on the manifold, i.e., the decision boundary of our classifier, and add these points to the planning graph. We emphasize that (1) no training data beyond the planning graph is required and (2) this learning step happens as part of planning and learning times are included in our experimental runtime results; modern GPU-accelerated learning techniques [17] require only a small fraction of total planning time to learn the manifold.

This learning and sampling process continues until we find a valid path. Figure 1 shows this process for a 2D scene with a long narrow passage. We learn the manifold that separates the two classes; free space samples on the manifold discover the must-connect regions between the two classes of nodes;

and we add these samples to the planning graph for training in the next iteration.

In section V, we perform experiments in narrow passage scenes to evaluate the algorithm and an easy planning problem scene to identify possible overhead. While a small overhead is introduced in the easy problem, SDCL produces one order of magnitude speedup in some of the narrow passage scenes when compared with previous planners. Besides this comparison, we note that SDCL may be applicable to other planners beyond PRMs. Learning and sampling the manifold could be used to guide sampling for other bi-directional or multi-directional motion planners to improve planning in narrow passages.

II. Related Work

A. Guided Sampling

Sampling-based motion planning methods [2], [16] rapidly cover the search space, typically using uniform random sampling. For planning in narrow passages, however, uniform random sampling usually requires a large number of samples to discover the small free spaces in the configuration space, especially in high dimensions [16]. Since sampling is key to finding plans, one approach is to employ guided sampling rather than uniform sampling. One of the generally applicable sampling methods is Gaussian sampling [18], which produces samples close to the obstacle region and is thus more likely to cover narrow passages. Another sampling method, a bridge test based sampling [19], is designed to increase the sampling density in narrow passages based on local geometrical features of narrow passages.

Previous work has also used topological tools in motion planning to guide sampling. Some variations of RRT [8], [9] control the exploring domain of the samples dynamically using previous samples' modified Voronoi regions. In recent work [20], the authors use discrete Morse theory to identify critical points near the obstacle region and build a topology map for improved planning time and path cost, though this method requires pre-processing to build the topology map. The work in [21] represents fixed rotation sub-configuration space explicitly with Minkowski sums, then uses it to create collision-free samples in PRM. This method works for robots with ellipsoid parts and translation-dominated motions.

There are also other strategic ways to guide sampling. The KPIECE algorithm [22] uses multi-level grids in the search space to guide sampling in less explored areas. Bi-directional search [5] and lazy collision checking [23] combined with KPIECE are effective in many scenarios. Motion planning in quotient-spaces [24], [25] projects the configuration space onto a series of lower dimensional spaces such that the lower-dimensional paths serve as heuristics to guild sampling in higher dimensions to improve the planning time, and the authors also proposed a method for efficient exploration around narrow passages [26]. The algorithm works best when the configuration space is more decoupled (e.g., a mobile base and robot manipulator on the base; the position and orientation of single object).

Some approaches use information from previous samples to guide current sampling. The tree expansion in [10] is guided by calculating the useful directions and distances of nearby spaces from previous sampling information. Learning methods have been employed to guide sampling. In [27], the authors learn a model of the configuration space, which is improved online and used to determine sample validity, reducing overall collision checking time in PRM. In [28], the authors employ principal component analysis (PCA) to detect narrow passage areas and improve sampling along the passages. The work in [29] takes a multi-tree RRT approach based on bridge tests and clustering, where a reinforcement learning approach guides the selection of trees. In [30], the algorithm biases sampling towards optimal path using a model trained with previous planning/demonstration data.

Our work develops a novel guided sampling strategy that learns from previous samples and interprets the learning results geometrically to sample in narrow passages. Experiments show improved runtime when compared with previous works. This algorithm is also applicable to bi-directional and multi-directional planners if we consider it as a general sampler like Gaussian sampling [18] and bridge-test sampling [19], since it requires minimal change to the base planner.

B. Connection with Infeasibility Proofs

This work is inspired by the construction of infeasibility proofs in [15], [31]. An infeasibility proof ensures that there is no valid path between a start and a goal in a given motion planning problem. In [15], infeasibility proof construction uses the learned manifold as a strong heuristic to form a polytope in $\mathcal{C}_{\text{obs}}$ that completely separates the start and the goal. While this current work also uses configuration samples to learn a manifold, the emphasis is now on using the manifold to identify samples in narrow passages and returning these samples for planning purposes. The two pieces naturally follow one another. When constructing infeasibility proofs, there is a negligible cost for the additional step to return the samples for planning. On the other side, close integration of SDCL and the base planner may also accelerate the learning process to construct infeasibility proofs.

III. Problem Definition

We consider a motion planning problem [32] consisting of a configuration space $\mathcal{C}$ of dimension n, start configuration $\boldsymbol{q}_{\text{start}}$, and goal configuration $\boldsymbol{q}_{\text{goal}}$. The configuration space $\mathcal{C}$ is the union of the closed set obstacle region $\mathcal{C}_{\text{obs}}$ and the open set free space $\mathcal{C}_{\text{free}}$. Both $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are in $\mathcal{C}_{\text{free}}$. $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ are implicitly defined through a validity checker which provides a binary response indicating whether a configuration $\boldsymbol{q}$ is valid ($\boldsymbol{q} \in \mathcal{C}_{\text{free}}$) or not ($\boldsymbol{q} \in \mathcal{C}_{\text{obs}}$). A feasible plan is defined as a path σ such that $\sigma[0,1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \boldsymbol{q}_{\text{start}}$, and $\sigma[1] = \boldsymbol{q}_{\text{goal}}$. The goal is to efficiently find such a path when there exist narrow passages.

The current work applies to kinematic motion planning problems without differential constraints. We assume that invalid configurations are only caused by the obstacle region

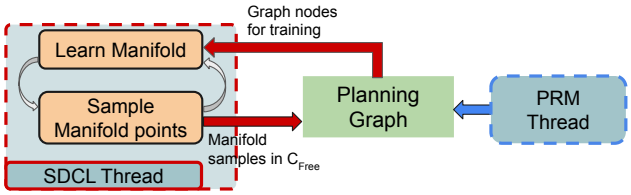
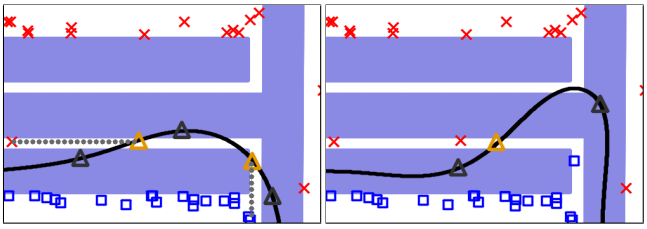


Fig. 2: Block diagram of SDCL. The planning graph is shared between two threads. Both threads contribute to the graph, and the SDCL thread uses the graph for training. todo: change graph to SDCL



(a) Current Iteration (b) Next Iteration

Fig. 3: Part of the learned manifold and points sampled on the manifold, before and after we add the two $\mathcal{C}_{\text{free}}$ manifold points (yellow triangles) to the planning graph. The $\mathcal{C}_{\text{free}}$ manifold points connect to their nearest neighbors in the planning graph (shown with dashed lines), and become part of the training data in the next iteration.

and configuration space boundaries. Planning for dynamic cases with narrow passages is part of our future work.

To apply the learning technique, we also assume that the configuration space is a Euclidean space: configurations are real-valued vectors with a Euclidean distance metric. This assumption is suitable for a variety of configuration spaces, including typical serial manipulators with joint limits. We treat the boundaries of the configuration space (e.g. joint limits) as a special obstacle region, which is described in more detail in previous work [31].

IV. Algorithm

In this section, we explain our SDCL algorithm and integration with PRMs [16]. SDCL thread runs in parallel with the PRM thread and requires only minor modifications to the PRM algorithm. We first briefly review PRM construction and then describe the integration with SDCL.

The main data structure in a PRM is the roadmap—an undirected graph G—where nodes are configurations and edges indicate a feasible connection between two nodes. Growing the roadmap iterates between two procedures. First, the construction step picks a random sample, adds this new sample to the roadmap, and tries to connect the new sample to neighboring nodes in G. Second, the expansion step improves the connectivity of G by choosing the difficult-to-connect points and performing a random-bounce walk from these points to reach other components in G. Integration with SDCL requires a slight change in the construction step. When sampling a new configuration, we not only add the free space configurations to G, but also save the obstacle region configurations in a separate set that we will later use to sample points on the learned manifold.

SDCL directly contributes to the roadmap G by providing more constructive samples for the narrow passage regions. The overall process iterates between two steps: learning the manifold and sampling the manifold (see Figure 2). The following sub-sections cover these steps in more detail.

A. Learning the Manifold

In this step, we learn the manifold using the planning graph G. If no plan has been found, then the start and goal components in G are disconnected. We take all nodes in G's goal component (the graph component connectable to the goal) as one class, and all other nodes of G as the other class. Thus, we separate the nodes of G into two classes, which are the input data to learn the manifold.

We learn the manifold as a classifier with the input data. Our implementation uses a support vector machine (SVM) with Radial Basis Function (RBF) kernel, which learns a classification function $F(\mathbf{x})$. While other learning methods may also work for SDCL, the RBF-kernel SVM has advantages for this application. It provides a closed-form function with only one hyper-parameter (over-fitting parameter γ), and SDCL is not sensitive to this parameter. We use $\gamma = 1$ for all the experiment scenes. Further detail on the using RBF-kernel SVMs to learn configuration space manifolds is described in [15], [31].

Since we use the samples in G for training, the distributions of samples also affect the learning results. For the experiments we run in section V, Gaussian sampling [18] has advantages over uniform sampling in some cases. Specifically, Gaussian sampling produces samples close to the obstacle regions, which may force the manifold into the narrow passage regions faster. This is similar in concept to the analysis in [31].

The manifold is a separation in $\mathcal{C}$ for the two classes of nodes in the planning graph. Figure 1 shows learned manifolds for 2D case with a narrow passage as more points are added to G and used for training. More closely in Figure 3, we see that the two growing classes of nodes "push" the manifold to cross the narrow passage regions, so we can use the manifold as a heuristic to sample in the narrow passages for connectivity. We describe this second step, sampling the manifold, next.

B. Sampling the manifold

The two classes of nodes from disconnected components of G must connect with each other to form a path. Since the manifold as the classifier decision boundary separates the two classes, any valid path must cross the manifold, and more precisely, a $\mathcal{C}_{\text{free}}$ configuration on the manifold. To locate $\mathcal{C}_{\text{free}}$ configurations that could possibly form a valid path, we sample points on the manifold and then use the validity checker to test whether the point is in $\mathcal{C}_{\text{free}}$.

Sampling the manifold is similar in concept to the sampling process in constrained motion planning [33]. Here, the "constraint function" is the function of the manifold $F(\boldsymbol{q})$. We use the projection method to sample on the manifold.

Starting from a given seed configuration $\boldsymbol{q}_s$, we solve the following optimization problem to project this configuration onto the manifold,

$$\begin{aligned} \min_{\boldsymbol{q}_m} \quad & \text{abs}\left(F(\boldsymbol{q}_m)\right) \\ \text{s.t.} \quad & \boldsymbol{q}_m \in \mathcal{C} \ , \end{aligned} \tag{1}$$

where $\boldsymbol{q}_m$ is the sample on the manifold we want to find, $\boldsymbol{q}_m \in \mathcal{C}$ means the sample needs to follow the requirements of the configuration space, e.g. joint limits. The optimization problem minimizes the absolute value of the manifold function, which could potentially be any learned manifold function, as long as the value and gradient of the function can be calculated. We solve equation (1) using sequential least-squares quadratic programming (SLSQP) [34], [35], [36], though other optimization methods may also be applicable.

We use samples found in the PRM's construction step as seeds to solve (1). All samples, including the samples in $\mathcal{C}_{\text{obs}}$, are potentially useful for sampling the manifold, which is why we modify the PRM's construction step to save the samples in $\mathcal{C}_{\text{obs}}$. However, there are trade-offs in the set or subset of PRM samples to use as seeds. Using fewer seeds requires less time for the projection calculations. On the other hand, more seeds generate more samples on the manifold, increasing the likelihood that the samples will lie in the narrow passage $\mathcal{C}_{\text{free}}$ regions. This selection of which PRM samples to use as seeds could be a user-specified option to SDCL. We evaluate and discuss the impacts of seed selection further in section V.

While sampling on the manifold, we add all $\mathcal{C}_{\text{free}}$ points on the manifold to G. Adding these $\mathcal{C}_{\text{free}}$ points to G is similar to adding new samples in the PRM construction step. We first add the $\mathcal{C}_{\text{free}}$ point to G as a separate component by itself, then try to connect the point to its nearest neighbors in G, which is described in more detail in [16]. Figure 3 shows the result of adding the $\mathcal{C}_{\text{free}}$ manifold points in two consecutive iterations.

The process of learning and sampling the manifold continues until a valid path is found or the time limit is reached (see algorithm 1). First, we acquire the input data from the planning graph G (line 2) and use it to train the manifold (line 3). Then, we sample the manifold by solving (1) (line 5) for each given configuration. If we successfully solve the optimization and the resulting manifold point is in $\mathcal{C}_{\text{free}}$, we add the point to G. Sampling the manifold for each seed configuration is embarrassingly parallel, enabling efficient use of modern multi-core CPUs.

C. Discussion

SDCL retains the probabilistic completeness of the underlying planner because it only adds additional samples to the planning graph. The planning graph is still generated by an underlying planner (e.g., a PRM). In the worst case, samples added by SDCL are useless (and might make planning slower), but the PRM still has its own sampling process and is probabilistically complete.

SDCL is broadly applicable to geometric motion planning problems. The learning and sampling process does not require any additional functions other than the validity checker, which is common to all sampling-based motion planners. Additionally, the user can specify whether to use Gaussian sampling, and possibly subsets of configurations as seeds for sampling the manifold.

Algorithm 1: SDCL

```
Input: G; // Planning Graph
Input: Q; // Set of configurations
Output: G; // Planning Graph
1 repeat
2     P_rest, P_goal ← Acquire-Input-Data(G);
3     f ← train-SVM(P_rest, P_goal);
      #pragma parallel for
4     foreach q ∈ Q do
5         q_m ← sample-manifold(q, f);
6         if q_m ∈ C_free then
7             add-sample(G, q_m);
8 until Timeout or Found path;
```

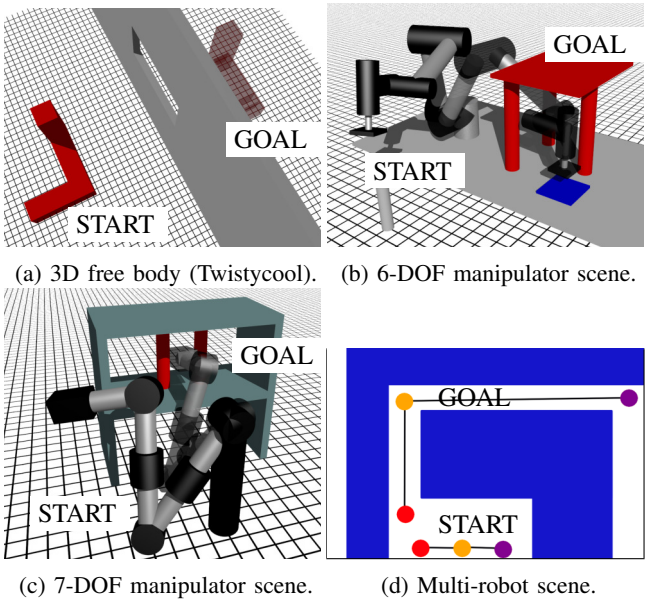
Although we use a PRM as the base planner for this section's description and our experimental evaluation, SDCL is not limited to PRMs. SDCL could be applied to other bi-directional or multi-directional planners, as long as we know which samples of the planning graph or tree are in the goal and non-goal components to learn the manifold. For example, with RRT-connect [5], we can use the start tree nodes as one class and the goal tree nodes as the other class. In general, SDCL can serve as an additional sampler that produces constructive samples for difficult-to-reach areas. One topic for future work is to evaluate which base algorithms benefit from SDCL and provide the fastest planning results.

V. Experiments

We run experiments in four scenes with narrow passages: one 3D rigid body scene, two serial manipulator scenes, and one multi-robot navigation scene. We implement SDCL with PRM (SDCL-PRM), and compare with earlier planners to demonstrate its effectiveness in reducing total runtime. We profile the different procedures in SDCL to identify and discuss future improvements. To demonstrate the potential overheads introduced while using SDCL, we also do experiments in simple planning scenes. All code is available[1].

We train the RBF-kernel SVM using ThunderSVM [17], which accelerates learning using GPUs. GPU acceleration is crucial because we train online while the planner is running. With GPU SVM training, the learning step only takes a couple of seconds. To evaluate the increase in speed from GPUs and to parallelize manifold sampling, we run our experiments on a server machine with NVIDIA TU102 GPU and a dual CPU AMD EPYC 7402 with 24 cores per CPU. We adapt PRM [16] in OMPL [6] to run in parallel with SDCL. We solve the nonlinear optimization

[1] http://sdcl.dyalab.org



(a) 3D free body (Twistycool). (b) 6-DOF manipulator scene.

(c) 7-DOF manipulator scene. (d) Multi-robot scene.

Fig. 4: Experiment setup.

problem for sampling the manifold using sequential least-squares quadratic programming (SLSQP) [34], [35], [36]. We check collisions and penetration depth using the Flexible Collision Library [37], and we model robot kinematics using Amino [38]. All robots use primitive shape collision geometry models for faster collision checking.

We run 30 trials for each experiment scene using our planner and baseline planners in OMPL. There are many planners in OMPL, and not all focus on quickly finding feasible plans, e.g., some focus instead on asymptotic optimally [4], [39] at the cost of typically slower runtime. For a more interpretable and even evaluation, we compare against feasible planners that are either (1) widely used or (2) offer competitive runtimes with our approach. We include SDCL results with the base PRM using Gaussian sampling and uniform sampling, and we use samples in both $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ as seeds to sample the manifold. We set timeout limits for the scenes. When planners timeout, they usually report a runtime that is slightly over the timeout limit. We use the reported runtime to calculate the mean and standard deviation, even in timeout cases.

A. Experiment Scenes and Results

1) Single rigid body: We use a classic free body narrow passage problem, the "Twistycool" problem [6], [40]. As shown in Figure 4a, the goal is to move a twisty 3D object from one side to the other through a small open window. The object must rotate in accordance with its shape to prevent colliding with the window because it is less in size than its bounding box. We use a 300 seconds timeout limit in all the trials. SDCL-PRM with uniform sampling solves all 30 trials and is the fastest among all planners. The second fastest planner is TRRT [41], which solves 28 out of 30 trials.

2) 6-DOF serial manipulator: In this scene, we use a manipulator structure similar to the Universal UR5 robot [42]. The goal is to reach the blue spot on the table, which is covered by a shelf that makes tight spaces. Figure 4b shows the start and goal configurations. The timeout is 200 seconds. Table I shows the experimental results. SDCL with Gaussian sampling solves all trials, with an average runtime of 4.53 seconds. The closest baseline planner is BiTRRT [43], which solves 27 of 30 trials with 10 times higher average runtime.

3) 7-DOF serial manipulator: We use a manipulator that is similar in structure to the Schunk LWA4D [44] in these experiments. The goal in the scene is to reach in between the two red obstacles on the shelf from outside of the shelf. Figure 4c shows the start and goal configurations. The timeout is 300 seconds. Table I shows the results. SDCL solves all the trials and is about 8 times faster than the closest baseline.

4) Communication-constrained Multi-Robot Navigation: In this experiment, we consider a multi-robot scene where the line of sight (i.e., for communication [45]) must be maintained between pairs of robots while exploring a narrow passage. In Figure 4d, the goal is to traverse and cover an "L" shape tunnel with three point robots while maintaining line of sight (the figure is modified for clarity; actual setup has a much narrower "L" passage). SDCL with Gaussian sampling is slightly slower than the best baseline planner by a few seconds.

5) Easy scenario to evaluate overhead: We setup an easy tabletop planning problem with the same 6-DOF manipulator in subsubsection V-A.2 to evaluate potential overheads of SDCL. The scene is similar to Figure 4b, but using an empty table. The result is in Table I. SDCL introduces overhead when compared with PRM. This overhead mainly comes from starting the training process. Note the unit in this column is milliseconds, so the average overhead introduced is a fraction of a second.

B. Experiments with Varying Difficulty

Besides the above experiments, we also test the change in runtime for varying levels of difficulty in the 7-DOF scene. We make the scene easier or harder by changing the distance between the two red cylinder obstacles, producing a corresponding change in the configuration space narrow passage. The easy case has an obstacle distance (surface to surface) of 1.66 times the end-effector size (a square box). The medium case is the original scene in Figure 4c. The hard case has an obstacle distance of 1.16 times the end-effector size. In this comparison, we only include the three other better performing planners, that is, BiTRRT [43], LBKPiece (Lazy Bi-directional KPIECE) [22] and SBL [46].

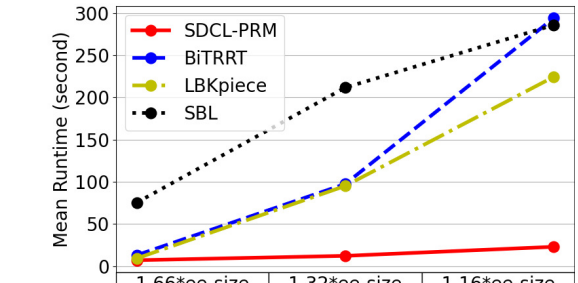
Figure 5 shows the increase in runtime from the easy case to the hard case. SDCL-PRM uses Gaussian sampling. As we can see, SDCL-PRM, BiTRRT, and LBKpiece have about the same average runtime in the easy case, and SBL is worse than these three. In the medium case, BiTRRT and LBKpiece's average runtime both increase by about 10 times, while SDCL-PRM increases by only a few seconds. In the hard case, the three baseline planners timeout in more than half of the trials, while SDCL-PRM runs all trials successfully with an average runtime of 23 seconds. These varying difficulty experiments indicate that sampling with SDCL is robust in revealing narrow passages in the configuration space even when the problem becomes more difficult.

	3D free body		6-DOF manipulator		7-DOF manipulator		Multi-Robot		Easy Scene
	Mean (s) ±STD	*Solved*	*Mean (s) ±STD*	*Solved*	*Mean (s) ±STD*	*Solved*	*Mean (s) ±STD*	*Solved*	*Mean (ms) ±STD*
SDCL-PRM-U	78.99 ±51.03	30/30	4.94 ±2.37	30/30	12.32±5.90	30/30	77.08 ±38.29	30/30	164.33 ±153.62
SDCL-PRM-G	109.32 ±69.38	30/30	4.53 ±2.38	30/30	12.36 ±13.67	30/30	41.64 ±14.86	30/30	164.05 ±154.16
KPIECE1	301.04 ±0.17	0/30	200.06 ±0.03	0/30	300.05 ±0.03	0/30	99.03 ±98.01	19/30	8.92 ±7.53
LBKPIECE1	300.97 ±0.19	0/30	180.36 ±55.07	4/30	95.18 ±114.05	25/30	119.12 ±87.09	16/30	19.68 ±9.88
BKPIECE1	300.81 ±0.14	0/30	179.69 ±42.59	7/30	216.01 ±109.31	14/30	37.02 ±20.22	30/30	12.75 ±5.72
BFMT	290.10 ±36.66	4/30	172.44 ±54.37	7/30	271.94 ±73.40	4/30	195.15 ±18.92	2/30	194.18 ±8.01
RRT	282.51 ±49.53	5/30	200.07 ±0.03	0/30	300.06 ±0.03	0/30	200.05 ±0.03	0/30	5.07 ±3.41
RRTConnect	299.85 ±1.74	1/30	200.06 ±0.03	0/30	298.53 ±8.17	1/30	200.06 ±0.03	0/30	2.72 ±0.27
LazyRRT	301.18 ±1.70	0/30	252.95 ±93.73	0/30	325.86 ±43.07	0/30	201.64 ±5.34	0/30	1.69 ±0.04
TRRT	106.55 ±82.15	28/30	200.06 ±0.03	0/30	300.05 ±0.03	0/30	111.06 ±87.55	16/30	4.73 ±2.32
BiTRRT	131.44 ±112.04	27/30	49.07 ±60.11	27/30	97.50 ±62.80	30/30	55.63 ±63.47	27/30	1.75 ±0.06
EST	300.07 ±0.03	0/30	200.06 ±0.03	0/30	300.06 ±0.02	0/30	200.05 ±0.03	0/30	2.02 ±0.84
SBL	301.34 ±0.66	0/30	195.77 ±17.27	2/30	211.71 ±100.50	16/30	168.37 ±53.52	9/30	12.01 ±6.33
PRM	189.06 ±97.45	23/30	200.23 ±0.03	0/30	300.24 ±0.04	0/30	202.04 ±0.03	0/30	5.76 ±9.52
LazyPRM	300.11 ±0.07	0/30	200.07 ±0.05	0/30	300.06 ±0.03	0/30	201.31 ±6.50	0/30	1.73 ±0.04

TABLE I: Runtime comparison of all the scenes, 30 trials for each planner, SDCL-PRM-G is SDCL with PRM and Gaussian sampling, SDCL-PRM-U is SDCL with PRM and uniform sampling.

	3D free body		6-DOF manipulator		7-DOF manipulator		Multi-Robot	
	Uniform	**Gaussian**	**Uniform**	**Gaussian**	**Uniform**	**Gaussian**	**Uniform**	**Gaussian**
Total (s)	78.76 ±50.92	108.99 ±69.24	4.94 ±2.37	4.53 ±2.38	12.32 ±5.90	12.36 ±13.67	77.08 ±38.29	41.64 ±14.86
Training itr.	11.83 ±1.09	8.97 ±0.81	81.00 ±16.56	46.17 ±14.35	195.97 ±53.90	73.37 ± 20.50	31.73 ±7.69	44.63 ±9.78
Training (s)	14.09 ±6.75	13.43 ±7.60	1.62 ±0.81	1.51 ±0.75	3.65 ±1.52	2.25 ±0.97	7.46 ±3.95	5.68 ±1.99
Sampling (s)	64.50 ±44.45	95.45 ±61.95	2.98 ±1.49	2.83 ±1.56	7.70 ±4.08	9.75 ±13.41	67.24 ±34.36	33.55 ±12.00

TABLE II: Runtime statistics (mean ±STD) for SDCL-PRM in all the scenes.



	1.66*ee-size	1.32*ee-size	1.16*ee-size
SDCL-PRM	7.20 | 30/30	12.36 | 30/30	23.07 | 30/30
BiTRRT	13.18 | 30/30	97.50 | 30/30	293.93 | 02/30
LBKpiece	9.39 | 30/30	95.18 | 25/30	224.35 | 12/30
SBL	75.51 | 29/30	211.71 | 16/30	285.79 | 02/30

Fig. 5: Compare 7-DOF scenes with different hardness levels. Table is showing "average runtime (s) | solved cases". 30 trials for each planner and each difficulty level.

C. Discussion of Experiments and Future Work

Table II shows the runtime statistics. SDCL with Gaussian sampling usually takes fewer learning and sampling iterations than Uniform sampling because Gaussian sampling samples close to $\mathcal{C}_{\text{obs}}$, which forces the manifold into $\mathcal{C}_{\text{obs}}$ and narrow passages faster [31]. In the multi-robot scene, using Gaussian sampling has significant advantages. However, Gaussian sampling does not always provide the best sample distribution for learning (the free body scene). Future investigation is required to decide the base planners' sampling strategies.

In all scenes, the sampling time is longer than the training time, especially in the multi-robot scene and the free body scene, where the average runtime is also much higher because of the long sampling time. Reducing sampling time is one direction of future work. Relating to the discussion in section IV on how the number of seeds influences sampling time and overall results, we can apply more filters to the seeds to reduce the number of unnecessary projection calculations that would produce $\mathcal{C}_{\text{obs}}$ samples.

SDCL is especially effective in the manipulator scenes, producing one order of magnitude improvement, and it is competitive in the free body scene and the multi-robot scene. However, there are limitations. Currently, SDCL cannot handle kinodynamic motion planning problems, which planners like KPIECE [22] support, because the learned manifold needs to exist in a Euclidean space. Additionally, scaling SDCL to very high dimensions, as handled by planners like QRRT [25], may pose challenges because learning and sampling the manifold would be more time-consuming. In general, SDCL excels in tightly coupled configuration spaces with extremely narrow passages connecting two parts of $\mathcal{C}_{\text{free}}$. For example, we could apply SDCL to the quotient spaces if the full configuration space is more decoupled.

Another direction of future work is to generalize SDCL as a sampler to support other planners, i.e., planners with a data structure suitable for learning a classifier, and compare the effect of SDCL in RRT-type and PRM-type algorithms.

VI. Conclusion

We presented a configuration space learning algorithm, coupled with a PRM, to improve kinematic motion planning in narrow passages and evaluated performance on a free 3D body, serial manipulators, and multi-robot navigation to show the efficiency of our algorithm. The algorithm is effective for solving the manipulator narrow passage problems with an order of magnitude speedup, despite the small overhead.

Acknowledgments

We thank Dr. Mehmet Belviranli for providing access to the multi-core server used to evaluate the presented algorithm.

REFERENCES

[1] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics*, vol. 14, no. 1, pp. 166–171, 1998.

[2] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[3] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Deptartment, Iowa State University, Tech. Rep. TR-98-11, October 1998.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[5] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[7] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *arXiv preprint arXiv:1109.3145*, 2011.

[8] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *2005 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2005, pp. 3856–3861.

[9] L. Jaillet, A. Yershova, S. M. La Valle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 2851–2856.

[10] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *2007 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 3307–3312.

[11] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, vol. 5, no. 48, p. eabd7710, 2020.

[12] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9563–9569.

[13] M. Pándy, D. Lenton, and R. Clark, "Unsupervised path regression networks," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1413–1420.

[14] A. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics & Automation Magazine*, vol. 4, no. 2, pp. 1255–1262, 2019.

[15] S. Li and N. T. Dantam, "Learning proofs of motion planning infeasibility," in *Proceedings of Robotics: Science and Systems*, 2021.

[16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics*, vol. 12, no. 4, pp. 566–580, 1996.

[17] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "ThunderSVM: A fast SVM library on GPUs and CPUs," *Journal of Machine Learning Research*, vol. 19, pp. 797–801, 2018.

[18] V. Boor, M. H. Overmars, and A. F. Van Der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *1999 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2. IEEE, 1999, pp. 1018–1023.

[19] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *2003 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2003, pp. 4420–4426.

[20] A. Upadhyay, B. Goldfarb, W. Wang, and C. Ekenna, "A new application of discrete morse theory to optimizing safe motion planning paths," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2022.

[21] S. Ruan, K. L. Poblete, H. Wu, Q. Ma, and G. S. Chirikjian, "Efficient path planning in narrow passages for robots with ellipsoidal components," *IEEE Transactions on Robotics*, 2022.

[22] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.

[23] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2000, pp. 521–528.

[24] A. Orthey and M. Toussaint, "Rapidly-exploring quotient-space trees: Motion planning using sequential simplifications," in *International Symposium on Robotics Research*. Springer, 2019, pp. 52–68.

[25] A. Orthey, A. Escande, and E. Yoshida, "Quotient-space motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8089–8096.

[26] A. Orthey and M. Toussaint, "Section patterns: Efficiently solving narrow passage problems in multilevel motion planning," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1891–1905, 2021.

[27] B. Burns and O. Brock, "Sampling-based motion planning using predictive models," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 3120–3125.

[28] S. Dalibard and J.-P. Laumond, "Linear dimensionality reduction in random motion planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1461–1476, 2011.

[29] W. Wang, L. Zuo, and X. Xu, "A learning-based multi-RRT approach for robot path planning in narrow passages," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 1, pp. 81–100, 2018.

[30] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[31] S. Li and N. T. Dantam, "Exponential convergence of infeasibility proofs for kinematic motion planning," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2022.

[32] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[33] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual review of control, robotics, and autonomous systems*, vol. 1, no. 1, pp. 159–185, 2018.

[34] D. Kraft, "A software package for sequential quadratic programming," Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Tech. Rep. DFVLR-FB 88-28, July 1988.

[35] ——, "Algorithm 733: TOMP–fortran modules for optimal control calculations," *Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[36] S. G. Johnson, "The NLopt nonlinear-optimization package," 2022, http://github.com/stevengj/nlopt.

[37] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3859–3866.

[38] N. T. Dantam, "Robust and efficient forward, differential, and inverse kinematics using dual quaternions," *The International Journal of Robotics Research*, 2020.

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

[40] M. Moll, I. A. Şucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, September 2015.

[41] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 2145–2150.

[42] UniversalRobots, "UR5 collaborative robot arm: Flexible and lightweight cobot." [Online]. Available: https://www.universal-robots.com/products/ur5-robot/

[43] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based RRT to deal with complex cost spaces," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 4120–4125.

[44] "Industrial robots," Oct 2021. [Online]. Available: https://schunk.com/us_en/solutions/industry-solutions/list/industrial-robots/

[45] M. A. Schack, J. G. Rogers, Q. Han, and N. T. Dantam, "Optimizing non-Markovian information gain under physics-based communication constraints," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4813–4819, 2021.

[46] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics research*. Springer, 2003, pp. 403–417.