Real-time Motion Planning in Changing Environments Using Topology-based Encoding of Past Knowledge

Richard Fisher[1], Benjamin Rosman[1,2] and Vladimir Ivan[3]

***Abstract*— Trajectory planning and replanning in complex environments often reuses very little information from the previous solutions. This is particularly evident when the motion is repeated multiple times with only a limited amount of variation between each run. To address this issue, we propose the DRM-connect algorithm, a combination of dynamic reachability maps (DRM) with lazy collision checking and a fallback strategy based on the RRT-connect algorithm which is used to repair the roadmap through further exploration. This fallback allows us to use much sparser roadmaps. Furthermore, we investigate using an approximate Reeb graph to capture the topology-persistent features of the past solutions of the problem utilising this sparsity. We evaluate DRM-connect with a Reeb graph on reaching tasks, and we compare it to state-of-the-art methods. We show that the proposed method outperforms both RRT-connect and BKPIECE algorithms in the number of collision checks required and we show that our method has the potential to scale to systems with higher number degrees of freedom.**

I. Introduction

Dexterous and mobile robots are widely used for performing repetitive tasks, such as mapping, inspection, cleaning and pick-and-place. While the environment, or any number of the parameters of the task, may vary between executions, the repetitive nature of these problems suggests that at some (possibly abstract) level, all instances of the problems share a common structure. In this case, we could repeat the task during an offline training process, store the obtained knowledge and then reuse it online, when the robot computes a new motion. There are three key aspects of this problem that we will address: 1) how to represent and efficiently store the prior knowledge so that it generalises across the task domain, 2) how to efficiently retrieve and use this knowledge online, and 3) how to recover and adapt the solution when the prior knowledge conflicts with the state of the environment at runtime. We focus particularly on the problem of knowledge representation and its efficient usage online (problems 1 and 2).

When the task structure is known, we can train models to represent the prior knowledge. For this, some form of a metric is required. However, if we don't have a candidate metric, modelling becomes difficult. Instead, we propose to exploit the topology of the solution space. Trajectories of motion that can successfully complete the task form a manifold in the state space of the robot. This manifold contains voids representing infeasible or suboptimal regions. Tools from topology, such as Reeb graphs, can be used to extract a representation of this manifold that captures the topology of the underlying task. An embedding of this representation in the state space of the robot can then be used to build a sparse graph that can be efficiently searched to perform motion planning (see Fig. 1). Adding a fallback strategy in case of a failure then addresses the 3rd problem.

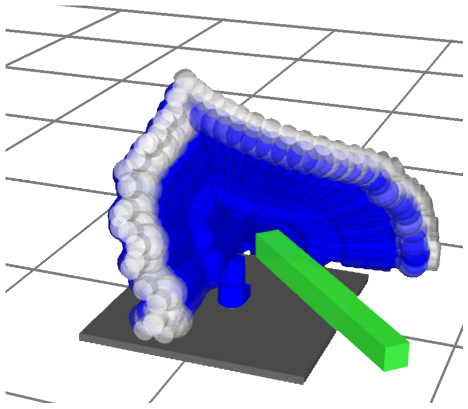
Fig. 1: Visualisation of example trajectory stored using the Reeb graph, executed on a 7DOF KUKA LWR-3 arm avoiding the green bar.

In this paper, we present a scalable probabilistic graph-based motion planning algorithm we call the Dynamic Road Map -connect (DRM-connect). This algorithm is an extension of the Probabilistic Roadmap (PRM) and a variant of the Rapidly-exploring random tree algorithm (RRT-connect). Our main contribution is to construct the roadmap using past solutions of the problem and to reduce its complexity by only storing an embedding of the Reeb graph of the successful solutions. We assume that the demonstrated successful trajectories may be optimal with respect to some arbitrary criteria, however, our method only attempts to compute feasible solutions. Our aim is to efficiently compute collision-free kinematic trajectories and to show that our method scales well to higher dimensional systems. We describe the related methods in the following section, then proceed to defining DRM-connect, and finally, we analyse its performance.

*This work was supported by EU H2020 project Memory of Motion (MEMMO, project ID: 780684) and by UKIERI project Learning Robotic Motor Skills, Visual Control and Perception for Warehouse Automation (UKIERI-DST 2016-17-0152).

[1] School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, SA `richard.fisher@students.wits.ac.za`

[2] Mobile Intelligent Autonomous Systems, Council for Scientific and Industrial Research, Pretoria, SA `brosman@csir.co.za`

[3] School of Informatics, University of Edinburgh, Edinburgh, UK `v.ivan@ed.ac.uk`

II. Background

Sample-based planning methods probe the free areas of the configuration space without the need for explicit construction of obstacles (which is infeasible for environments with complex obstacles or kinematics). These methods concern themselves only with the random selection of states to query under the assumption that each state can be checked for validity (e.g. collisions). Common sampling-based planners (SBP) either build a tree (online) or a graph (offline) which span the configuration space. These structures are then used to compute feasible or optimal paths [1].

Rapidly-exploring random trees (RRT) are a popular single-query unidirectional path planning algorithm which constructs a search tree over the configuration space by randomly sampling from it [1]. Many variants of RRT attempt to improve performance by biasing the sampled points in a way that leverages past information [2], [3], [4], [5]. Other single-query methods include RRT-connect [6], a bidirectional extension of RRT with specialised heuristics to improve the efficiency of the exploration and BKPIECE [7] which exploits discretisation of low dimensional projections of the state space to drive the exploration.

Asymptotically optimal extensions to RRT and RRT-connect exist as RRT* [8] and RRT*-connect [9], offering this optimality with a constant factor increase in solve time.

In situations where many queries are expected in the same environment, offline pre-computation can be valuable. Algorithms of this type are called multiple-query methods. The Probabilistic Roadmap planner (PRM) [10] is a multiple-query planner which builds a graph of connected regions in the configuration space offline and then performs a graph search to compute the shortest path online. The density of the roadmap can be specified, which affects the robustness and speed of planning, but this roadmap does not adapt to changing environments.

Extensions to both single- and multiple-query methods exist for replanning. RRT has been extended to Execution-extended RRT [2], Dynamic RRT [3], Lazy Reconfiguration Forests [4], Multipartite RRTs [5] and RRTx [11], amongst others. All of these attempt to retain information from either prior tasks or the currently executing path to speed up replanning.

PRM has been extended for dynamic replanning as Dynamic Roadmap Planning [12], [13], iDRM [14] which inverts the frame of reference and computes the location of the floating base and HDRM [15] which exploits hierarchical encoding and symmetries to store a dense roadmap providing resolution completeness guarantees at the cost of allowing only one joint to move at a time. All these methods allow multiple queries when the environment stays static and an efficient way to update the roadmap when it does change. However, the roadmap-based methods all lack efficient encoding of past experience for similar tasks in the same or similar environments.

One way of capturing important information about the structure of the tasks is to consider the topology of the task manifold. In cases where replanning due to dynamic obstacles is required, the appearance of an obstacle could have one of two effects. We would either require minor changes to the planned path (such as a person stepping around an unexpected chair) or major changes to the path (such as a locked door – the person has to find a path through another room). From a topological perspective, this can be treated by reasoning about trajectory equivalence classes. Equivalence classes (or homotopy equivalence classes) are collections of all trajectories, such that no trajectory from one class can be continuously deformed to any trajectory in any other class [16]. The difference, then, between these two scenarios mentioned above is that the closed door invalidates an entire homotopy equivalence class of trajectories, while the chair may split an equivalence class, but (probably) leaves many paths through the current trajectory class still valid.

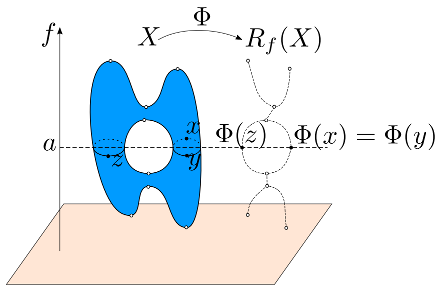

Fig. 2: An example of a Reeb graph (right) of the blue volume (left). The clear and solid circles on the left correspond to critical points and level sets of the continuous function f used to construct the Reeb graph (e.g. the height function).

Pokorny et al. [16] propose an approach to recover these trajectory classes for path replanning and trajectory optimisation. Samples from the free space are used to estimate the path-connectedness of the configuration space using filtrations of simplicial complexes[1]. Persistent homology is used to find an optimal scale factor ϵ for the filtrations.

Reeb graphs are another representation of topological connectedness, but describe volume connectedness rather than path connectedness. A Reeb graph is an abstract graph which describes the evolution of level sets of a continuous function f defined on some volume [18]. This is the 1-skeleton of the volume (and can be thought of as it's contraction onto a set of arcs through the "centre" of it). An example of a volume and its corresponding Reeb graph is shown in Figure 2. The function Φ maps the volume X to the Reeb graph $R_f(X)$. This abstract graph could then be embedded back into the volume to create a compact description of it.

Constructing Reeb graphs from successful trajectories of a robot is of particular interest. In their query phases, algorithms such as DRM and PRM can use any kind of embedded graph, such as an embedded Reeb graph.

III. Graph-based Replanning

[1]For a full treatment of computational topology in path planning, see Carlsson [17], Edelsbrunner [18] and LaValle [1].

Let $\mathcal{C} \in \mathbb{R}^N$ be the configuration space of a N-DoF robot and $\mathbf{q} \in \mathcal{C}$ be a state in configuration space. Let $\mathcal{C}_{obs}$ represent the obstacles and $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ the collision-free region. A classical probabilistic roadmap (PRM) contains a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \in \mathcal{C}_{free}$ are the vertices (robot configurations) and $\mathcal{E} \subset \mathcal{C}_{free}$ are the edges (trajectory segments) that connect two neighboring vertices. These vertices and edges are generated during an offline pre-processing phase. During the online planning phase, given start and goal states q_s, q_g, we first find two vertices $\mathcal{V}_s$ and $\mathcal{V}_g$ that are closest to the start and goal states respectively. Then, a graph search algorithm, such as A* [19], is deployed to find a path in the roadmap connecting $\mathcal{V}_s$ and $\mathcal{V}_g$. However, the pre-generated vertices and edges may not be valid in an unknown and non-static environment. The validity of the stored vertices and edges must be checked, and in many cases we need to sample new collision-free configurations during the on-line phase.

To address this, PRM was extended for dynamic replanning scenarios by maintaining and repairing the roadmap [12], [13] in what is termed dynamic roadmap planning (DRM). Similar to PRM, the first step of DRM is a pre-planning stage where a probabilistic roadmap is built, assuming no obstacles. A rectangular cell decomposition of the workspace is then computed, and each cell is mapped to the nodes and edges of the roadmap which correspond to collisions in these decomposed cells. When obstacles in the workspace appear, move or disappear, they may intersect with different parts of the discretised workspce. The mapping to the edges and vertices of the roadmap makes it efficient to infer which edges and nodes are affected by the change in the environment. Online, when responding to planning queries, nodes and edges are temporarily invalidated when obstacles move through their mapped cells. Queries are then carried out as A*-style graph searches. The graph has to densely cover the configuration space for this approach to be able to solve queries with any reasonable success rate. However, sparsity in the graph decreases the computational cost as a smaller graph has to be maintained.

A. Planning with sparse roadmaps

In a sparse graph, some parts of the state space may become entirely unreachable, especially after invalidating parts of the graph due to collisions. We propose to modify DRM to repair the graph if no path through the graph is found. Our algorithm, called DRM-connect (Alg. 1) uses an approach similar to RRT-connect to repair and expand the roadmap if no path exists through it due to obstructions.

The task is to find a path between a start and goal (q_s and q_g respectively) using a graph $\mathcal{G}$. If either the start or end point do not exist in $\mathcal{G}$ (e.g. if the start or end point has moved due to sensor errors, replanning, or if this is a new task), then these are added as unconnected nodes to $\mathcal{G}$ (lines 1 to 6). Then, a search is conducted on the graph for the shortest path (line 7), checking collisions lazily, using the DRM-lazy algorithm (discussed below). If this fails to return a valid path, the graph is repaired (line 9) using RRT-connect by adding the tree nodes to the graph, after which DRM-lazy is called again (line 10). This is repeated until a valid path is found.

Algorithm 1 DRM-CONNECT $(\mathcal{G}, q_s, q_g)$

```
1:  if $q_s \notin V(\mathcal{G})$ then
2:      $\mathcal{G}$.add_node($q_s$)
3:  end if
4:  if $q_g \notin V(\mathcal{G})$ then
5:      $\mathcal{G}$.add_node($q_g$)
6:  end if
7:  $returnFlag, p \leftarrow$ DRM-LAZY();
8:  while not $returnFlag$ do
9:      $\mathcal{G} \leftarrow$ RRT-CONNECT();
10:     $p \leftarrow$ DRM-LAZY();
11: end while
12: return $p$
```

RRT-connect in line 9 is modified slightly to operate on disconnected graphs. It maintains 3 graphs (containing q_s, q_g and all other disconnected points respectively) rather than two trees. The two terminal graphs are grown towards random points with the same greedy heuristic present in standard RRT-connect, except that the graphs are also biased to grow towards the graph of disconnected points.

DRM-connect generalises both RRT-connect and DRM. If no prior knowledge is used ($\mathcal{G}$ is empty), then DRM-connect simplifies to RRT-connect, where the start and end points are used as the roots for the trees which grow to connect to one another. Assuming then that the knowledge provided is useful in solving the task, the performance of DRM-connect should be at worst the same as RRT-connect. If the path repair algorithm (DRM-lazy) is removed or there exists a collision-free path, then DRM-connect is equivalent to DRM. Thus (ignoring the time for collision checking), DRM-connect will match the performance of DRM in cases where DRM would succeed – namely when the start and end points lie on $\mathcal{G}$ and there are no collisions on the shortest path.

DRM-connect differs from DRM in that if a path is not found through the graph, a repair process is carried out to join the subgraphs, rather than using RRT-connect from scratch. It is expected that this (rather than an RRT-connect fallback) should reduce the replanning time if obstacles appear which split the graph into disjoint subgraphs, since the motions encoded by these subgraphs can be reused.

DRM-lazy (line 7) attempts a graph search on the supplied graph, with lazy collision checking, removing invalid edges and repeating until either a valid path is found or q_s and q_g lie in disconnected subgraphs.

Currently, no workspace-configuration space mapping is used to invalidate edges, but rather lazy collision checking. Since collision checking is typically the largest contributor to query time, the mapping (which allows collisions to be checked using a lookup) can significantly improve replanning times, especially when the number of edges to check is large.

However, the graphs tested here are fairly sparse. For larger graphs, the mapping would confer greater benefits.

The completeness of DRM-connect can be inferred from the following scenarios. If a path exists through the graph $\mathcal{G}$, DRM-connect finds a solution and returns it in finite time, which satisfies the requirements for completeness. If a path doesn't exist through the graph, RRT-connect is called. Since RRT-connect is probabilistically complete, DRM-connect is expected also to be probabilistically complete.

The DRM-connect algorithm can be used with any kind of roadmap. However, the fallback strategy of RRT-connect allows us to utilise sparse roadmaps which may better capture task-specific prior knowledge.

IV. Motion Representation using Embedded Reeb Graphs

We now explore using an embedded Reeb graph as a representation of the set of trajectories which form the prior experience of a task or environment.

Given a height function f, a Reeb graph describes the 1-connectivity of a space. From samples along successful trajectories, and ignoring spurious loops, this is the graph on which the contraction of all trajectory equivalence classes must lie. Thus the Reeb graph can be interpreted as a compact representation of all trajectory classes [18].

Our approach to creating a Reeb graph is based on the following procedure:

1) The manifold is swept in order of increasing function value $f(u_i)$, where u_i is the ith vertex of the triangulation of the manifold.
2) Whenever an appearing, disappearing, bifurcating or merging volume is found, the Reeb graph being constructed is updated accordingly, by creating a node and joining it to other present nodes, according to the type of critical point encountered.

Our approach follows the technique proposed in [20] to embed the Reeb graph into the configuration space. The nodes of each of the set of training paths p are extracted and the pairwise distance between these points is calculated. A graph ($\mathcal{G}_{adjacency}$) is constructed by connecting all points with a pairwise distance less than or equal to a parameter ϵ. Height values are then assigned to each node in the graph, using GET-HEIGHT-MAP (Alg. 2). The graph is then contracted into an embedded Reeb graph by BUILD-REEB-GRAPH.

The requirements we impose on GET-HEIGHT-MAP are simply that the heights along the shortest path from any point to q_g are increasing. Additionally, an advantage would be to have these node heights fairly uniformly spread, so that the sweep of heights during BUILD-REEB-GRAPH (see below) encounters the nodes spaced out evenly across the range of height values. GET-HEIGHT-MAP satisfies this criteria by finding the shortest path between the start and goal, spreading height values along this path, and repeating this for the nodes with minimum and maximum height values, removing nodes with known heights which are not adjacent to nodes with unknown heights.

Algorithm 2 GET-HEIGHT-MAP($\mathcal{G}, q_s, q_g$)

```
1: $h(q_s) \leftarrow 0$; $h(q_g) \leftarrow 1$; $\mathcal{G}_{conn} \leftarrow \mathcal{G}$
2: while (any $h$ unassigned) do
3:     for all $\mathcal{G}_{sub}$ in SUBGRAPH($\mathcal{G}_{conn}$) do
4:         $q_0, h_0 \leftarrow$ MIN($h(\mathcal{G}_{sub})$)
5:         $q_1, h_1 \leftarrow$ MAX($h(\mathcal{G}_{sub})$)
6:         $p \leftarrow$ SHORTEST-PATH($\mathcal{G}_{sub}, q_0, q_1$)
7:         $h(q_0 : q_1) \leftarrow$ ASSIGN-HEIGHTS($p, h_0, h_1$)
8:         REMOVE-EDGES($\mathcal{G}_{conn}, p$)
9:     end for
10: end while
11: return $h$
```

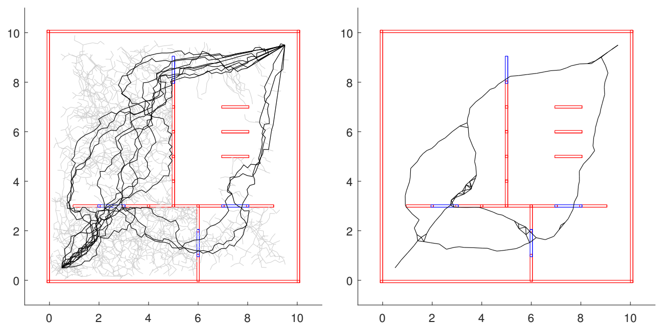
(a) Input trajectories. Grey lines show the full RRTs, and black lines the solution paths. (b) Embedded Reeb graph. Note the representation of distinct trajectory classes.

Fig. 3: Reeb graph generation in the 2-D Doors domain.

BUILD-REEB-GRAPH then generates an approximate embedded Reeb graph ($\mathcal{G}_{embedded\_reeb}$) by segmenting the nodes in $\mathcal{G}$ into bands of uniform width, according to the height values h (7 bands are used here). The vertex closest to the geometric midpoint of each connected subgraph in each band is assigned as a seed-node. These are connected together (where possible) by finding the shortest path through $\mathcal{G}$. The number of bands can be varied depending on the volume of data points in the adjacency graph and the computational requirements. The approach used here guarantees that the embedded Reeb graph is a subgraph of $\mathcal{G}_{adjacency}$ above.

Embedded Reeb graphs (with the right selection of ϵ) are sparse representations of data volumes, and we use them as roadmaps for DRM-connect.

V. Evaluation

A. Reeb graph evaluation

We validate the use of a Reeb graph as a representation of the prior knowledge in the 2-D Doors domain (shown in Fig. 3). The red lines are fixed obstacles, while the blue lines are transient obstacles (which appear and disappear between tasks). We generate 12 paths using RRT (Fig. 3a), for the task with $q_s = (0.5, 0.5)$ and $q_g = (9.5, 9.5)$. These 12 trajectories are contracted into an approximate Reeb graph, with $\epsilon = 0.19$ (Fig. 3b).

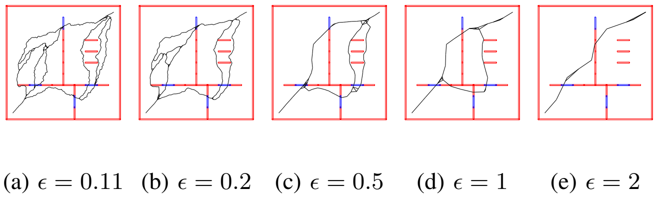
(a) $\epsilon = 0.11$ (b) $\epsilon = 0.2$ (c) $\epsilon = 0.5$ (d) $\epsilon = 1$ (e) $\epsilon = 2$

Fig. 4: Effect of the ϵ parameter on the graph complexity.

The effect of varying ϵ in the DOORS domain is shown in Fig. 4. As the value of ϵ increases, the Reeb graph becomes more compact, with increasingly fewer loops. Additionally, some of the edges in the Reeb graph begin to be in collision, since ϵ is larger than the minimum size of the obstacles. For smaller values of ϵ the Reeb graph successfully captures the task manifold in 2-D. As this value increases, the graph may become too coarse and oversimplify the embedding. Further exploration of selecting a value for ϵ is left for future work.

B. Benchmarking on reaching task

This section presents experimental results in three 7-D domains, shown in Fig. 5. Here, the domain complexity increases from left to right. In each environment, the robot arm has to move from the start point q_s to the end point q_g while avoiding obstacles.

We test the performance of DRM-connect against RRT-connect and BKPIECE using the number of collisions checked as the performance metric. We have used the implementation in the EXOTica library [21]. As the implementations may vary, this metric serves as a measure of computation time irrespective of the computer or collision checking algorithm used.

Five unique start poses are uniformly sampled from the configuration space of each domain. The end point is always $q_g = [\frac{\pi}{2}, \frac{\pi}{2}, 0, 0, 0, 0, 0]$. For each of these, the start point, end point and obstacles are perturbed 10 times by adding noise with $\sigma = \frac{\pi}{64}$ to each configuration and noise with $\sigma = 5mm$ to each obstacle. We build a Reeb graph for each of these, using 12 trajectories from RRT-connect. We use DRM-connect to solve each of these 50 tasks. If any of these are unsolvable (using RRT-connect) these are discarded and replaced.

During the execution, we interrupt motion at intervals of 10% of the distance between the start and end configuration d defined as:

$$d = (\mathbf{p_c} - \mathbf{p_s}) \cdot \frac{\mathbf{p_g} - \mathbf{p_s}}{||\mathbf{p_g} - \mathbf{p_s}||^2}, \qquad (1)$$

where $\mathbf{p_s}$, $\mathbf{p_c}$ and $\mathbf{p_g}$ are the coordinates of the start, current and goal points respectively. This is just the projection of the current path extent onto the vector from the start point to the end point. The path is interrupted with a ball of radius $0.1m$ (approximately the size of the end effector).

Each of the tested algorithms is tasked with replanning from the last valid state to q_g.

Figs. 6, 7 and 8 show the performance of DRM-connect compared to the other benchmark algorithms in the BAR,

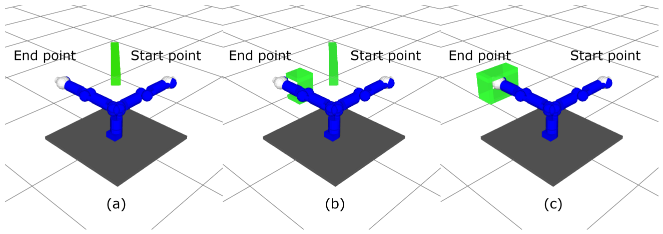

Fig. 5: The three test environments (domains) used in the evaluation (a) BAR domain – a single bar shaped obstace is present, (b) CUP domain – the goal is placed inside a C-shaped obstacle, and (c) BOX domain – the goal is inside a box-shaped obstacle. The 7DOF KUKA LWR3 arm has to reach from the start to the goal configuration.

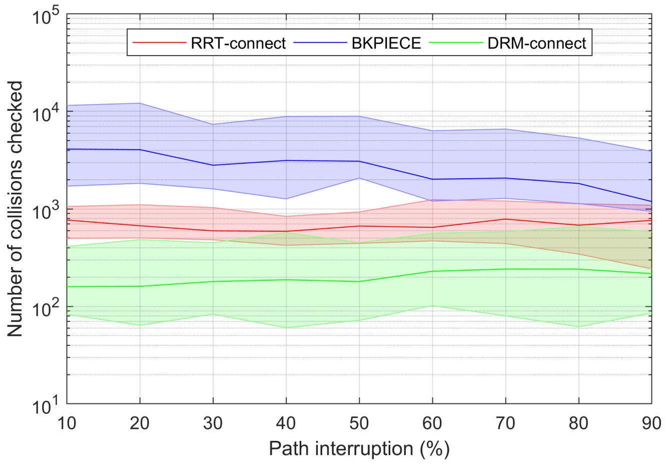

Fig. 6: No. of collision checks for DRM-connect, RRT-connect and BKPIECE in the BAR domain.

CUP and BOX domains respectively. In each figure the x-axis shows how far into the execution of the path the robot is obstructed (see Eq. 1). The results shown are the median values over 50 runs, with the shaded regions representing the interquartile range (50 sets of example paths, Reeb graphs and obstructions). Note that the y-axes are plotted using a log-scale.

Using DRM-connect with a Reeb graph provides a significant runtime benefit over RRT-connect and BKPIECE in all domains, due to collisions checked along the Reeb graph, a sparse representation of prior experience on the task manifold. In the BAR domain, the advantage is limited (approximately 5 times fewer collisions than RRT-connect and 10 times fewer than BKPIECE on average), since the domain itself has many redundancies. As the task complexity increases (the CUP and BOX domains), the improvement of DRM-connect becomes more pronounced (around 10-100 times fewer collisions than RRT-connect/BKPIECE). For these domains, when the path is interrupted near the goal, this typically increases the task complexity further, since the inserted obstacle will further constrain access to the goal. In these cases there is a trade-off between BKPIECE/RRT-connect and DRM-connect, yet DRM-connect still outperforms both.

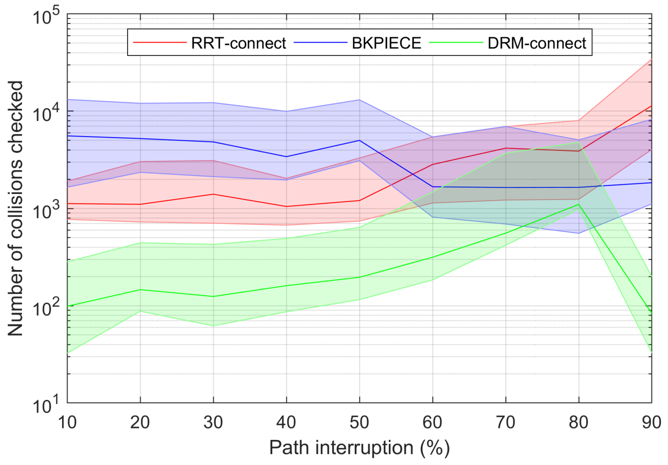

Fig. 7: No. of collision checks for DRM-connect, RRT-connect and BKPIECE in the CUP domain.

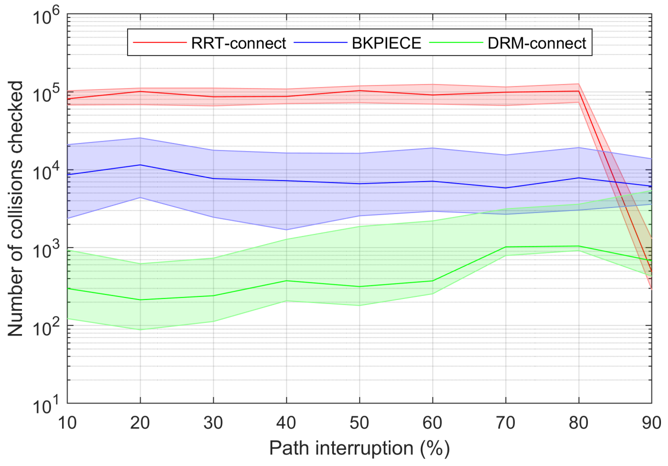

Fig. 8: No. of collision checks for DRM-connect, RRT-connect and BKPIECE in the BOX domain.

VI. CONCLUSION

We have presented a graph-based motion planning method that exploits the previous solutions of the problem. To achieve this, we combined the DRM method with a fallback strategy using the RRT-connect algorithm. The resulting DRM-connect method is capable of solving motion planning queries efficiently while using very sparse, task specific roadmaps. We then constructed such roadmaps using an embedded Reeb graph. Our evaluation of the Reeb graph showed that the representation is suitable and that the open parameters can be tuned to achieve optimal roadmap complexity. In our future work, we will explore how to automate the tuning of the complexity using tools from persistent homology to compute an optimal value of the ϵ parameter robustly. This will be of particular interest when the state space dimensionality increases. While our experiments have shown promising results on 2-D (navigation) and 7-D (fixed base robot reaching motion) problems, a sparse Reeb graph may be able to scale to high dimensional spaces, such as humanoid whole body motion planning ($\sim$30-D state space). The effect of the curse of dimensionality and high degree of redundancy on the utility of the Reeb graph representation will also be the focus of our future work. Finally, we construct the Reeb graph using all the sample data available offline. A life-long-learning approach to storing the prior knowledge by recomputing the Reeb graph may be possible.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[2] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2383–2388.

[3] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation*, 2006, pp. 1243–1248.

[4] R. Gayle, K. R. Klinger, and P. G. Xavier, "Lazy reconfiguration forest: An approach for planning with multiple tasks in dynamic environments," *IEEE Transactions on Robotics and Automation*, pp. 1316–1323, 2007.

[5] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1603–1609.

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

[7] I. A. Şucan and L. E. Kavraki, *Kinodynamic motion planning by interior-exterior cell exploration*. Springer, 2009.

[8] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, p. 2, 2010.

[9] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.

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

[11] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, pp. 797–822, 2016.

[12] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments," in *International Workshop on the Algorithmic Foundations of Robotics*, no. 9, 2000, pp. 363–376.

[13] L. Jaillet and T. Siméon, "A PRM-based motion planner for dynamically changing environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 2004, pp. 1606–1611.

[14] Y. Yang, V. Ivan, Z. Li, M. Fallon, and S. Vijayakumar, "iDRM: Humanoid motion planning with realtime end-pose selection in complex environments," in *2016 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 271–278.

[15] Y. Yang, W. Merkt, V. Ivan, Z. Li, and S. Vijayakumar, "HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 551–558, 2018.

[16] F. T. Pokorny, M. Hawasly, and S. Ramamoorthy, "Topological trajectory classification with filtrations of simplicial complexes and persistent homology," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 204–223, 2016.

[17] G. Carlsson, "Topology and data," *Bulletin of the American Mathematical Society*, vol. 46, no. 2, pp. 255–308, 2009.

[18] H. Edelsbrunner and J. Harer, *Computational topology: an introduction*. American Mathematical Society, 2010.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[20] X. Ge, I. Safa, M. Belkin, and Y. Wang, "Data skeletonization via Reeb graphs," *Advances in Neural Information Processing Systems*, pp. 837 – 845, 2011.

[21] V. Ivan, Y. Yang, W. Merkt, M. P. Camilleri, and S. Vijayakumar, *EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control*. Springer International Publishing, 2019, pp. 211–240.