Abstract—Motivated by what is required for real-time path planning, the paper starts out by presenting sRMPD, a new recursive “local” planner founded on the key notion that, unless made necessary by an obstacle, there must be no deviation from the shortest path between any two points, which would normally be a straight line path in the configuration space. Subsequently, we increase the power of sRMPD by using it as a “connect” subroutine call in a higher-level sampling-based algorithm mRMPD that is inspired by multi-RRT. As a consequence, mRMPD spawns a larger number of space-exploring trees in regions of the configuration space that are characterized by a higher density of obstacles. The overall effect is a hybrid tree growing strategy with a trade-off between random exploration as made possible by multi-RRT based logic and immediate exploitation of opportunities to connect two trees as made possible by sRMPD. The mRMPD planner can be biased with regard to this trade-off for solving different kinds of planning problems efficiently. Based on the test cases we have run, our experiments show that mRMPD can reduce planning time by up to 80% compared to basic RRT.

I. INTRODUCTION

Path planning has been an important area of research in robotics over the last several decades. Path planning algorithms have important uses in robotic assembly [1], autonomous driving [2], kinematic and dynamic control for robots, etc.

In traditional approaches to path planning, one first overlays a grid of points on the configuration space and then develops an obstacle-free path incrementally from the start configuration to the goal configuration, going from one grid point to the next in the process. Subsequently, a search is carried out over the paths thus discovered to find the optimal path that connects the goal configuration with the start configuration. These algorithms are known to work well in low-dimensional configuration spaces. However, as the dimensionality of the space increases, they extract a large performance penalty.

More recently, sampling based algorithms have gained considerable prominence. The basic idea of such algorithms is to sample the configuration space at randomly selected points until a connection of the local pathways thus constructed can lead one from the start configuration to the goal configuration. These are best exemplified by the Probabilistic Roadmap Method (PRM), the Randomized Potential Fields, and the Rapidly-exploring Random Tree (RRT). For some of the more important factors that account for their popularity: (1) They can be used with greater ease in high dimensional configuration spaces compared to the traditional algorithms; (2) The algorithms based on PRM and RRT can be shown to be probabilistically complete; (3) The ease in merging partially developed solutions in order to respond simultaneously to multiple path planning queries; etc.

In our laboratory we have been exploring the use of RRT and RRT-Connect algorithms for real-time motion planning as needed for automatic pruning of dormant trees — a crucial step in growing healthy apple orchards. Our main concern has been the extent to which a straightforward application of RRT-Connect, as originally formulated by its authors, results in unnecessary exploration of large segments of the configuration space while slowly advancing in obstacle dense regions.

To address this shortcoming of the randomized algorithms, we propose a new approach here. The first part of our approach is based on a novel “local” planner that aims to overcome immediate obstacles. This local planner is based on the key notion of divide and conquer driven by the intuition that, unless made necessary by an obstacle, there must be no deviation from the shortest path between any two points, which would normally be a straight line path in the configuration space. We refer to this basic path planner as the sRMPD algorithm for “Single-path Recursive Mid-Point Displacement” algorithm. Its basic idea is recursive: Check each sampling point on the path connecting two end-points for being collision free. If that condition is not satisfied at any sampling point, move the mid-point to a collision-free location to create a detour, divide the resulting path into sub-paths and attempt to find a collision-free detour for each such sub-path. If the detour is again in collision, further divide the sub-path, and so on.
Subsequently, we increase the power of sRMPD by using it as a “connect” subroutine call in a higher-level randomized algorithm (mRMPD) that is inspired by RRT-Connect [3] and multi-RRT [4]. As a consequence, mRMPD has the ability to spawn a large number of space exploring trees in those regions of the configuration space that are characterized by a large density of obstacles. Incorporating both multi-tree and sRMPD logic in mRMPD gives the planner the ability to make a trade-off between exploration through growing additional trees and exploitation of opportunities for connecting trees directly. It would be correct to say that mRMPD uses a hybrid tree growing strategy that invokes a combination of multi-RRT type logic and sRMPD for the exploration behavior, while, at the same time, mRMPD uses just sRMPD for the behavior required for making direct connections. mRMPD can be appropriately biased with respect to this trade-off for solving different kinds of planning problems.

Spawning randomized space-exploring trees in a manner similar to multi-RRT makes mRMPD probabilistically complete. And basing its basic path-construction operation toward sRMPD significantly enhances both the rate of exploration and the merging of the trees. In particular, mRMPD significantly reduces unnecessary exploring in open regions since nodes in such regions can be connected efficiently by sRMPD. What contributes further to the efficiency with which narrow passages are treated is the behavior of sRMPD to seek detours through such passages.

In the experimental section, we demonstrate our algorithm using both 2D bitmap and 3D environments with narrow passages that must be traversed by a path from the start state to the goal state. Based on the test cases we have run, we observe mRMPD reduces the average number of states and planning time consistently. Furthermore, mRMPD achieves up to 80% reduction in planning time for some of test cases vis-a-vis the other state-of-the-art algorithms.

In the rest of the paper, Section II presents the related work that is most relevant to mRMPD. In particular, this section reviews the RRT algorithm along with related past research that has focused on addressing the shortcomings of RRT in exploring through narrow passages. Subsequently, Section III presents our RMPD algorithm. Section IV presents the experimental results that demonstrate the efficiency of the RMPD algorithm vis-a-vis its main competition. Finally, we conclude in Section V.

II. RELATED WORK

We start with an overview of the sampling based approaches to path planning. A large promise of sampling based methods is the theoretical guarantee of probabilistic completeness. If there exists a solution for a motion planning problem, the sampling based algorithms will eventually find it provided the samples are sufficiently dense in the configuration space. Such algorithms are generally either graph-based or tree-based. Graph based motion planning algorithms, with Probabilistic Roadmap Method [5] being the most representative, are widely used in multi-query motion planning scenarios. Given a static workspace, the “roadmap” constructed once is able to accommodate multiple queries; and, when necessary, the roadmap may be expanded. On the other hand, tree based algorithms, such as Rapidly-exploring Random Tree (RRT) [6], build space filling trees with biased growth toward large unexplored regions. Algorithm 1 presents a brief overview of RRT.

Algorithm 1 RRT
1. Initiate \( \tau \)
2. for \( i \leftarrow 1 \ldots N_{\text{iterations}} \) do
3. Uniformly sample a collision-free point \( p_{\text{random}} \)
4. Select the nearest-neighbor \( p_{\text{near}} \) in \( \tau \) for \( p_{\text{random}} \)
5. Extend \( p_{\text{near}} \) toward \( p_{\text{random}} \) to obtain \( p_{\text{new}} \)
6. if \( p_{\text{new}} \) and \( p_{\text{goal}} \) are close enough
7. return path in \( \tau \)
8. return empty path

From the standpoint of efficiency, a problem with randomized sampling based algorithms, such as RRT, is that they tend to carry out excessive unnecessary sampling. Only a tiny fraction of what is explored in the configuration space is eventually used for the construction of the solution path. In other words, due to the indiscriminate nature of uniform sampling, wide-open regions are likely to become over-sampled before the growth of RRT is sufficient in narrow regions. Many approaches have been proposed to address this problem, either by directly biasing the sampling to favor obstacle-dense spaces [7], [8], [9], [10] or by actively reducing the sampling in wide-open free spaces [7], [10], [3], [4]. In the rest of this section, we will refer to the former as “Narrow-passages Favored Sampling” and the latter as “Open-Space Selective Sampling.”

Narrow-passages Favored Sampling: Approaches in this category aim explicitly to sample more densely in potentially narrow regions of the configuration space. In order to identify such regions, Hsu et al. has proposed a bridge test in which a bridge is defined as a line segment consisting of two in-collision endpoints and one free middle point [8]. Hsu et al. showed that short bridges are easily built across narrow passages. As a result, narrow passage samples can be directly obtained by selecting the bridge test survivors. Additionally, Zhang et al. has proposed a retraction-based modification to RRT, which generates samples near obstacles more densely at the cost of a higher computational overhead [9]. Every new in-collision random sample is retracted iteratively to its closest point in the contact space. The contribution reported in [7] attempts to integrate both the bridge test and the retraction — retraction is carried out on the samples that pass the bridge test.

Open-space Selective Sampling: The main idea here is to implicitly bias the sampling toward narrower regions in the configuration space by biasing it against wide-open regions. While it is difficult to characterize the degree of openness of a region deterministically, an approximate approach for doing so is proposed in [7] in which a free hypersphere is associated with every non-contact RRT node. Subsequently,
samples falling within any of the free hyperspheres are discarded. Another way to create biased sampling is to simply apply a goal bias to the sampling distribution. Bi-directional RRT, namely RRT-Connect, proposed in [3] also improves the rate of convergence of the trees using a greedy extend function. Moreover, along the same lines as RRT-Connect, an arbitrary number of RRTs can be grown simultaneously to increase the pace of exploration in narrow regions, as described in [4].

At a high level of comparison, our algorithm incorporates both philosophies mentioned above in an intuitive manner. Aggressively connecting samples with straight lines, whenever that is possible, significantly diminishes the over exploration of RRT. At the same time, our approach explores more densely in the vicinity of narrow regions through the detouring logic triggered by in-collision samples.

In Section [IV] we compare the performance of our proposed method with the other algorithms mentioned previously. The input in all cases consists of the start state and the goal state of a robot in the configuration space, while the output is a collision-free path from the start to the goal state when such a path exists.

III. RMPD – Recursive Mid-Point Displacement Algorithm

In this section, we propose a novel yet simple path planning algorithm — Recursive Mid-Point Displacement (RMPD) algorithm — that uses the simplest of the intuitions for constructing the paths at their most elemental level and that, in addition, calls on randomized sampling based strategies to ensure probabilistic completeness. At its most elemental level, the path construction logic is as simple as it can be: Don’t deviate from a straight-line unless absolutely necessary on account of the obstacles.

We first present a single-path version of the algorithm (sRMPD), which focuses on overcoming simple local obstacles without much computational overhead. In order to find a path, sRMPD connects any two nodes with a straight line, which is checked for in-collision property using a local planner. If this property is found to be true at any sampling point on the straight-line path, tangential detours around the obstacle are made recursively by replacing the line’s middle point. Subsequently we embed sRMPD as a subroutine call in the higher-level algorithm, mRMPD, that may be considered to be a multi-path variant of the basic algorithm. mRMPD leverages the robustness and efficiency of the single-path sRMPD as a “connect” routine and further incorporates a random sampling based scheme to improve the success rate for difficult planning problems.

A. Single-path RMPD (sRMPD)

In contrast to the retraction method described in [9], [7], our basic algorithm, sRMPD, navigates its way around obstacles with inexpensive resampling, while leveraging the idea of divide and conquer. As the recursion goes deeper, sRMPD focuses on finding a warped collision-free replacement for a shorter path segment within its local area.

The steps of sRMPD are presented in Algorithm 2. Using the notation shown in Figure [1], given the start state $s$ and the goal state $g$ in a configuration space, the algorithm first validates $s$ and $g$ and returns with failure if either $s$ or $g$ is in collision. If both states are valid, sRMPD invokes the recursive function Detour to populate an exploration tree $\tau_s$ rooted at $s$ with intermediate waypoints. If the condition returned by Detour is true, we have found a valid path residing in $\tau_s$.

Algorithm 2 sRMPD($\tau_s, s, g$)

1. if $s$ or $g$ is in collision
2. return false
3. return Detour($\tau_s, s, g$)

Algorithm 3 Detour($\tau_s, p_s, p_g$)

1. static $i_{recur} \Leftarrow 0$
2. if ($i_{recur} \Leftarrow i_{recur} + 1 > N_{max}$
3. return false
4. if ($\langle p_s, p_g \rangle$ is in collision
5. $p_m \Leftarrow (p_s + p_g)/2$
6. if $p_m$ is in collision
7. $d \Leftarrow Distance(p_m, p_g)$
8. $p_f \Leftarrow NearbyRandomFreeState(p_m, d)$
9. else
10. $p_f \Leftarrow p_m$
11. return Detour($\tau_s, p_s, p_f$) and Detour($\tau_s, p_f, p_g$)
12. else
13. $\tau_s$.add($p_g$)
14. return true

As a key step in sRMPD, Detour($\tau_s, p_s, p_g$) recursively breaks a path segment into two potentially collision-free sub-paths and appends the middle state to the final path, as shown in Algorithm 3. The function first attempts to connect $p_s$ and $p_g$ directly with a straight line ($\langle p_s, p_g \rangle$) using a bidirectional local planner. If no obstacle is encountered, the function simply adds $p_g$ to $\tau_s$ as a waypoint in the final path. On the other hand, if a path segment ($\langle p_s, p_g \rangle$) encounters an obstacle at any sampling point, Detour tries to make a tangential detour around the obstacle by breaking ($\langle p_s, p_g \rangle$) into two halves.

First, $p_m$, the middle state of $p_s$ and $p_g$, is checked for collision. In case of collision, $p_m$ is replaced by a nearby collision-free point $p_f$, returned by NearbyRandomFreeState($p_m, d$). More specifically, NearbyRandomFreeState function simply samples uniformly within a hypersphere around $p_m$ until a collision free state is found. The diameter of the hypersphere is
determined by the distance between $p_s$ and $p_g$, which ensures locality of the search for a free middle point. Finally, Detour recursively invokes itself with both sub-paths, ($\langle p_s, p_f \rangle$)

A collision free state is guaranteed since both $p_s$ and $p_g$ are collision free and reside on the search hypersphere.
and \((p_f, p_g)\). Recursion terminates automatically when the waypoint states in \(\tau_s\) connect \(p_s\) to \(p_g\) successfully with a collision-free path.

The implementation of sRMPD, as shown in Algorithm\(^2\), is iterative and based on a stack data structure. We specify a max number of iterations for sRMPD, \(N_{\text{max}}\). If \(s\) and \(g\) are still not connected after \(N_{\text{max}}\) iterations, we deem that sRMPD invocation to have failed to find a solution.

**B. Multi-path RMPD (mRMPD)**

mRMPD incorporates sRMPD as a “connect” procedure and leverages its effectiveness in finding paths that must squeeze through narrow passages\(^3\). But, as intuition would suggest, in terms of completeness, sRMPD by itself would be severely limited in its ability to find paths in the presence of complex-shaped obstacles in the configuration space. That is where mRMPD comes to our rescue.

mRMPD tries to grow and simultaneously merge space-exploring trees. The idea of mRMPD is partially inspired by multiple RRTs, as described in [4], where multiple RRTs are shown to outperform RRT-Connect in terms of the convergence rate in configuration spaces with high obstacle densities. Since essentially each exploration tree in mRMPD can be treated as an RRT but with a greedy extend function, mRMPD is probabilistically complete owing to the probabilistic completeness of RRT. What that implies is that a feasible path is guaranteed to be found if it exists. mRMPD couples that guarantee with the efficiency made possible by its ability to call on sRMPD for exploring direct interconnects. This gives mRMPD an overall hybrid tree growing strategy in which trees can either extend toward a common new state or attempt to interconnect directly.

As shown in Algorithm\(^4\) mRMPD uses RRT-like trees to organize the sub-paths discovered. Only two trees are initialized, \(\tau_s\) and \(\tau_g\), using the start state and the goal state, respectively. At the beginning of each iteration, mRMPD chooses either to explore or to inter-connect randomly according to the exploration bias, \(\sigma_{\text{expl}}\).

On the one hand, the interconnection procedure in mRMPD, as shown in Algorithm\(^4\) Line 6 to 10, first samples a random configuration (which could be in-collision) as a pivot. And, then, sRMPD is invoked to connect two nearest neighbor nodes of the pivot on two randomly chosen trees with the hope of speeding up convergence.

On the other hand, the exploration procedure encourages multiple trees to connect to a common free point, \(p_{\text{curr}}\). For each successful connection, the corresponding tree of the nearest neighbor point and the tree of the current point are merged into one. In the very first iteration, the current point is the goal state of the query, \(g\). Afterwards, as a feasible path is still not found between \(s\) and \(g\), a new tree is spawned using a randomly sampled free state. The current state then is assigned to be the newly sampled state. Although the number of concurrent trees scales with the obstacle density and eventually converges to the number of partitions of the open space, we only select up to \(K\) nearest trees for exploration in the actual implementation. Eventually, the iterations terminate when \(\tau_s\) and \(\tau_g\) are connected and smoothing functions can then be applied to produce the final output path.

**Algorithm 4 mRMPD\((s, g)\)**

```plaintext
1. \(\tau_s, \text{init}(s), \tau_g, \text{init}(g)\)
2. \(\tau_{\text{all}} \leftarrow \{\tau_s, \tau_g\}\)
3. \(\tau_{\text{curr}} \leftarrow \tau_g, p_{\text{curr}} \leftarrow g\)
4. while not IsConnected\((s, g)\) do
5.   if RandomDecimal\((0, 1) > \sigma_{\text{expl}}\) then
6.     \(\tau_1, \tau_2 \leftarrow \text{TwoRandomTrees}(\tau_{\text{all}})\)
7.     \(p \leftarrow \text{RandomState}()\)
8.     \(p_1 \leftarrow \tau_1.\text{nearest}(p), p_2 \leftarrow \tau_2.\text{nearest}(p)\)
9.     if sRMPD\((\tau_1, p_1, p_2)\) then
10.    \(\text{MergeTrees}(\tau_1, \tau_2)\)
11.  end if
12. for each \(\tau\) in \(\{\text{K nearest trees of } p_{\text{curr}}\}\) do
13.   \(p \leftarrow \tau.\text{nearest}(p_{\text{curr}})\)
14.   if sRMPD\((\tau, p, p_{\text{curr}})\) then
15.     \(\text{MergeTrees}(\tau, \tau_{\text{curr}})\)
16.   end if
17. end for
18. \(p_{\text{new}} \leftarrow \text{RandomFreeState}()\)
19. \(\tau_{\text{new}}, \text{init}(p_{\text{new}})\)
20. \(\tau_{\text{curr}} \leftarrow \tau_{\text{new}}, p_{\text{curr}} \leftarrow p_{\text{new}}\)
21. \(\tau_{\text{all}} \leftarrow \tau_{\text{all}} \cup \tau_{\text{new}}\)
22. end while
23. return \(p_{\text{path}}\) in \(\tau_s\)
```

**IV. EXPERIMENTS**

In this section, we compare the performance of mRMPD against RRT (with 0.05 goal bias) [6] and its two other state-of-the-art variants: RRT-Connect [3] and multi-RRT [4]. We show that mRMPD outperforms its competitors in terms of planning time and number of states without sacrificing the rate of success.

**A. Experimental Setup**

Both sRMPD and mRMPD are implemented in C++ within the OMPL [11] framework. OMPL also provides optimized implementations for the competing algorithms — this ensures fairness when comparing experimental results. All experiments were performed 30 times on a 2.30 GHz Intel i7 processor with 8 GB of RAM.

Our experiments apply all the algorithms to two types of path planning problems: a) A 2D bitmap with a 2 DoF point robot; and b) A 6 DoF rigid body robot in an obstacle rich 3D environment.

The bitmap case we studied, as shown in Figure\(^2\a\), includes a Zigzag pattern with its narrow passages to test the planners’ ability of navigating through them. In order to demonstrate that mRMPD is better able to avoid over-exploration in open spaces, we deliberately included open spaces at the two sides, as shown in Figure\(^2\b\).

To demonstrate the performance of our algorithm in higher dimensional configuration space, we tested it on two 6DOF
rigid bodies in a 3D space, as shown in Figure 4a and 5a. In the first case, the Twistcool problem, a 5-link rigid-body robot must navigate through a small hole in a barrier in the middle while there exist large open spaces on the two sides of the barrier. In the second case, an example of the real-world Piano Mover’s Problem, a piano must be maneuvered through the living room and the narrow corridor at top right of Figure 5a. In order to demonstrate the robustness of the planner, we empirically set $N_{\text{max}} = 10$, $K = 2$ and $\sigma_{\text{expl}} = 0.1$ for all the experiments described in the next subsection.

B. Experimental Results

We will first show the experimental results qualitatively and then quantitatively. The point of illustrating the qualitative results is to impress upon the reader the fact that, in comparison with the other randomized sampling algorithms, mRMPD carries out far less exploration of the open spaces, as seen in Figure 2d. By contrast, both the original goal-biased RRT and RRT-Connect show poor performance at navigating through narrow passages; for these two algorithms, excessive number of states accumulate in wide-open spaces before a solution is found (Figure 2b and 2c).

For quantitative results, we start with the Zigzag problem. The results in terms of the number of states and planning times for this problem are shown in Figure 3. Not surprisingly, mRMPD outperforms its competitors with far fewer states and much less time, which is in accord with the visualization of the states in Figure 2d.

For the case of 6 DoF motions by a robot, we start with the Twistcool problem, as shown in Figure 4a. Similar to the
Zigzag problem, the broad open space is divided by a narrow passage, except this time in 6 dimensions. The quantitative results for the Twisctcool problem are shown in Figures 4c and 4d. mRMPD is the only algorithm that consistently solves the problem within 200 seconds, achieving at least a factor of 6 speedup in terms of the planning time as compared to the other planners.

Finally, we demonstrate the quantitative results on a more realistic benchmark, the Piano Mover’s problem, as shown in Figure 5a. Note that, compared to the previous benchmarks, collision checking is more expensive now on account of the complexity of the piano and the environment meshes. Furthermore, the open space is now partitioned, as illustrated by the two isolated (blue and yellow) trees in Figure 5b. Therefore, the ability of a multi-tree planner to efficiently navigate through narrow passages becomes critical, since over-exploration is penalized by costly collision checking and disconnected open spaces. The performance comparison for the Piano Mover’s problem is in Figures 5c and 5d. As a result of the computational expense of collision checking and the difficulties posed by partitioned open spaces, the reduction in planning time for mRMPD is not as significant as in the other two benchmarks. Nonetheless, mRMPD still achieves a factor of 7 reduction in the number of states and a factor of 2 reduction in the planning time when compared to RRT-Connect.

C. Limitations

Despite out-performing the other well-known planners in our comparative study, mRMPD has some limitations of its own. First, the performance of mRMPD degrades when open space is strongly partitioned, since extra exploration effort will be spent in vain in disconnected regions. Although this can be partially compensated for by setting $\sigma_{\text{expl}} = 0$, that causes mRMPD to lose its balance between exploration and direct inter-connection, which results in the planner performing poorly. Secondly, the run-time efficiency of our planner depends inversely on the cost of collision checking. Intuitively, if the local planner finds that a direct line path is in collision (Algorithm 2, Line 4), much of the collision checking effort is wasted, since partially valid segments are not memorized.

V. Conclusion

Best known randomized sampling based algorithms, while possessing the highly desirable property of probabilistic completeness, unfortunately tend to carry out unnecessary randomized explorations in open spaces and expand slowly in narrow passages. Our “local” planner, sRMPD, has the ability to efficiently bypasses local obstacles using inexpensive resampling, which accelerates excursions into narrow passages through a divide-and-conquer strategy. Additionally, being simple in its implementation, sRMPD can easily be plugged into a wide range of existing planners as a “connect” routine. Our top-level planner, mRMPD, invokes sRMPD as a subroutine and takes advantage of multiple exploration trees for solving complex planning problems.

mRMPD uses a hybrid growing strategy that establishes a trade off between random sampling for space exploration and an attempt at direct interconnections. This trade-off can be exploited with an appropriate bias toward direct inter-connects for solving difficult path planning problems. This makes RMPD a powerful new approach to path planning.

REFERENCES

[1] S. Y. Nof, Handbook of industrial robotics. John Wiley & Sons, 1999, vol. 1.
[2] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, “Real-time motion planning with applications to autonomous urban driving,” IEEE Transactions on Control Systems Technology, vol. 17, no. 5, pp. 1105–1118, 2009.
[3] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on, vol. 2. IEEE, 2000, pp. 995–1001.
[4] M. Clifton, G. Paul, N. Kwok, D. Liu, and D.-L. Wang, “Evaluating performance of multiple rrt’s,” in Mechntronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on. IEEE, 2008, pp. 564–569.
[5] 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 and Automation, vol. 12, no. 4, pp. 566–580, 1996.
[6] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
[7] J. Lee, O. Kwon, L. Zhang, and S.-e. Yoon, “Sr-rrt: Selective retraction-based rrt planner,” in Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012, pp. 2543–2550.
[8] D. Hui, T. Jiang, J. Reif, and Z. Sun, “The bridge test for sampling narrow passages with probabilistic roadmap planners,” in Robotics and Automation, 2003. Proceedings. ICRA’03. IEEE International Conference on, vol. 3. IEEE, 2003, pp. 4420–4426.
[9] L. Zhang and D. Manocha, “An efficient retraction-based rrt planner,” in Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on. IEEE, 2008, pp. 3743–3750.
[10] C. Urmson and R. Simmons, “Approaches for heuristically biasing rrt growth,” in Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, vol. 2. IEEE, 2003, pp. 1178–1183.
[11] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72–82, 2012.