The Toggle Local Planner for Sampling-Based Motion Planning

Jory Denny and Nancy M. Amato

***Abstract*— Sampling-based solutions to the motion planning problem, such as the probabilistic roadmap method (PRM), have become commonplace in robotics applications. These solutions are the norm as the dimensionality of the planning space grows, i.e., $d > 5$. An important primitive of these methods is the *local planner*, which is used for validation of simple paths between two configurations. The most common is the straight-line local planner which interpolates along the straight line between the two configurations. In this paper, we introduce a new local planner, *Toggle Local Planner* (Toggle LP), which extends local planning to a two-dimensional subspace of the overall planning space. If no path exists between the two configurations in the subspace, then Toggle LP is guaranteed to correctly return $false$. Intuitively, more connections could be found by Toggle LP than by the straight-line planner, resulting in better connected roadmaps. As shown in our results, this is the case, and additionally, the extra cost, in terms of time or storage, for Toggle LP is minimal. Additionally, our experimental analysis of the planner shows the benefit for a wide array of robots, with DOF as high as 70.**

I. Introduction

Motion planning is seen in a wide array of applications including robotics, bioinformatics [24], and gaming/virtual reality [17]. Sampling-based solutions to the motion planning problem are commonly used in these application domains. Many of these methods are based upon the *probabilistic roadmap method* (PRM) [15] that creates a roadmap representing the planning space and extracts paths from it. An important primitive of sampling-based motion planning is *local planning*, which determines and validates simple paths between two points. Methods for local planning should be simple, as they are called many times during planning. While much attention has been given to designing various sampling methods, adaptive strategies, and efficiently extending PRMs to very high dimensions, local planning has not received as much attention. In particular, most algorithms simply use the *straight-line Local Planner* (straight-line LP) which interpolates along the straight line connecting the two configurations at some fixed resolution depending on environment features.

A more successful local planner will increase the connectivity (number of edges) of the roadmap. Additionally, a good local planner should use $O(1)$ storage per edge and avoid storing all connection information for the edge, e.g., all intermediate nodes in the path. To accomplish this, local planners are usually deterministic, meaning the paths that they validate can be easily reconstructed. Lastly, a good local planner should be efficient, as edge validation is the most expensive operation in PRMs [2]. For example, the straight-line local planner is deterministic, efficient, and has no additional storage requirements, but is unable to connect many nodes. More efficient local planners could reduce the need for near optimal placement of nodes to solve the motion planning problem with PRMs [7].

In this paper, we present a new local planning method, *Toggle Local Planner* (Toggle LP), that increases the connectivity of the roadmap by extending simple one-dimensional interpolation methods such as the straight-line LP to a two-dimensional subspace of the total planning space. Each call to Toggle LP requires one new configuration to be generated, thus additional storage is limited to storing one configuration per edge created by the local planner, meaning Toggle LP requires storage space of $O(1)$. Lastly this new method has relatively low overhead for execution when compared to other simple methods.

In *Toggle Local Planner*, a straight-line connection between the configurations s and g is attempted. If this fails, both a witness node is returned from the failed connection attempt (returning witnesses to failures introduced in [6]) and a third node n is generated that defines a triangle $\triangle sgn$ in which a path will be sought. The method of selecting n could be varied, e.g., randomly generated or pushed a distance δ. A recursive search of the triangle is executed in an attempt to find a path. If a path through C_{obst} is found blocking any possible connection between s and g in C_{free} (within the triangle), then a disconnection is found between s and g. The blocking path is related to disconnection proofs for motion planning [3] but is restricted to the triangle $\triangle sgn$.

The main contributions of this work are as follows:

- Introduction of *Toggle Local Planner*, a new simple local planner.
- Extend local planning to a randomly selected two dimensional subspace of the overall problem.
- Theoretically show guarantees of the stopping condition of Toggle LP.
- Experimentally compare the effectiveness of Toggle LP with other local planners in a range of problem types.

II. Preliminaries and Related Work

A. Preliminaries

This paper considers the *motion planning problem*, i.e., finding a valid (e.g., collision-free) path through some environment for a movable object (*robot*). The robot's position

This research supported in part by NSF awards This research supported in part by NSF Grants CRI-0551685, CCF-0833199, CCF-0830753, IIS-096053, IIS-0917266 by THECB NHARP award 000512-0097-2009, by Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

J. Denny and N.M. Amato are with the Parasol Lab., Dept. of Computer Science and Engineering, Texas A&M Univ., College Station, Texas, 77843-3112, USA. {jdenny,amato}@cse.tamu.edu

within the environment is described by d parameters upon which the movement of the robot can change, commonly called *degrees of freedom* (DOF). The set of all possible configurations within the environment is called the configuration space, C_{space} [18], having a dimension equivalent to the DOF of the robot. The set of all valid configurations is called *free space*, C_{free}, while the set of all invalid configurations is called *obstacle space*, C_{obst}. With this abstraction, the motion planning problem becomes that of finding a continuous trajectory in C_{free} connecting the points in C_{free} corresponding to the start and the goal configurations. In general, it is intractable to compute explicit C_{obst} boundaries [22], but we can often determine whether a configuration is feasible or not, e.g., by performing a collision detection (CD) test in the *workspace* – the robot's natural space.

Sampling-based motion planners explore C_{space} and produce a data structure containing representative valid configurations and information about the connectivity of C_{free}. One such planner, the *Probabilistic Roadmap Method* (PRM) [15], builds a roadmap (graph) in C_{free}. The first phase in this process, *node generation*, is where collision-free configurations are sampled and added as nodes to the roadmap. In the second phase, *node connection*, neighboring nodes are selected by a *distance metric* as potential candidates for connection. Then, simple *local planners* attempt connections between the selected nodes; successful connections are represented as roadmap edges.

In this paper, we are addressing the local planner within these sampling-based motion planners. Simple solutions to the motion planning problem can be referred to as *local planners*. These local planners are used to determine if simple, easily reconstructible, paths can be formed between two nearby configurations s and g. The intuition is that less sophisticated (and therefore less costly) methods can be quite effective in connecting nearby configurations. Thus, these local planners are an important component of sampling-based motion planners.

B. Local Planners

In sampling-based motion planning, how the edges of the roadmap are verified is an important operation. Local plans should minimize extra storage requirements, be easily reconstructible, and be efficient to compute. Some metrics to avoid checking edges have been proposed [21], [4], [16]. In this paper, we focus on the issue of defining and verifying edges. Here we discuss a few of the common methods for completing simple local plans between two configurations.

The first method, *straight-line*, has been used in many motion planning algorithms (e.g., [13]). In straight-line local planning, a simple straight-line interpolation between two configurations is calculated. This line segment is broken up into small segments split by some precomputed distance resolution at "ticks." Each "tick" is checked for validity. Each tick must be valid for the edge to be considered valid. A similar approach is to move along *Manhattan* paths (one DOF at a time). This however is less used. *Manhattan* paths can be improved with *rebound* methods (e.g., [10], [11]) which perform slightly better.

The *Rotate-at-S* local planner [2] was introduced to improve local planning near obstacles such as roadmaps containing OBPRM (Obstacle Based PRM) [1] or Gaussian PRM nodes [5]. *Rotate-at-S* is similar to the straight-line local planner, except motion is split into three distinct sections. First translation occurs for the first s percent of the straight line. Then all rotation is done at s percent along the straight line interpolation. Lastly, the translation is finished off. Verification is done in a similar method to straight-line local planning. This planner performs well when configurations are located near obstacles, such as with OBPRM or Gaussian samples.

Other planners based upon the A^* search have been developed [14], [19]. These are not always suitable for local planning strategies because of how expensive they are to evaluate and compute in both space and time complexity. Other methods use a slide tactic for solving local plans [14], [8], [26]. These methods try to move between two configurations c_1 and c_2 by first moving to a neighbor c' of c_1 that is a step closer to c_2 and has a maximal clearance from obstacles. Planning continues by trying a neighbor c'' of c' which has larger clearance than c' and is closer to c_2. These planners are typically expensive and non-deterministic which makes them difficult to recompute without storing significant data. Stopping criterion such as maximum number of tries is implemented to ensure stopping in reasonable time. Other metrics such as distance can be used instead of clearance. A^* planners were previously compared to Rotate-at-S and straight-line local planning in [2]. This work found it is beneficial to use complex planners, i.e., A^*, in a hierarchy of local planners and to use them only when simple local planners, e.g., straight-line and Rotate-at-S, fail.

C. Utilizing C_{obst} and Disconnection Proofs

In our method, we utilize important information from both C_{free} and C_{obst}. There are many PRM variants which bias sampling using collision information such as obstacle-based sampling [1], Gaussian sampling [5], and bridge-test sampling [12]. All these methods use simple tests or movements of configurations to bias sampling towards difficult areas such as narrow passages or obstacle surfaces. Additionally, many methods have been proposed that use collision information to identify important regions of C_{space} which might need additional sampling [20], [23], [27].

Other work exploits specific tests to discover full disconnections in C_{free} upon which the robot cannot pass [3]. However, this work is limited by robot and environment types. A recent approach, Toggle PRM [6] introduces the concept of simultaneously mapping C_{obst} and C_{free}. In Toggle PRM, two roadmaps are retained, one representing the connectivity of C_{obst} and one representing the connectivity of C_{free}. Additionally, when connections are attempted between two configurations, witness configurations to local plan failures are returned and added to the opposite map. With this addition it was shown that Toggle PRM can more

efficiently sample within narrow passages. The *Toggle Local Planner*, introduced in this paper, employs the general ideas of Toggle PRM to plan in both C_{free} and C_{obst}, as well as returning witnesses to failed connection attempts.

III. Toggle Local Planner

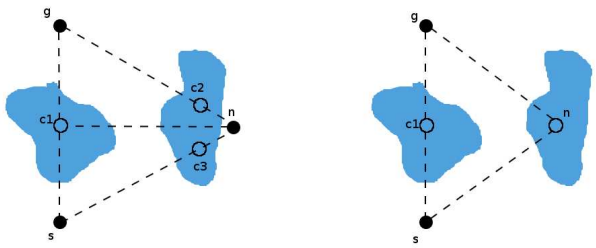
In *Toggle Local Planner* (Toggle LP), a straight-line connection between the configurations s and g is attempted. If this fails, not only is a witness node returned from the failed connection attempt (introduced in [6]), but also a third node n is generated that defines a triangle $\triangle sgn$ which will be searched for a path. After some preprocessing, a recursive analysis of the triangle is performed trying to find a path in C_{obst} that disconnects s and g in $\triangle sgn$. If in the process a valid path is found, then it will be returned. When a disconnection path through C_{obst} exists within $\triangle sgn$, then the algorithm is guaranteed to find it. The full algorithm is described in Section III-A. An example of the algorithm can be found in Section III-B. The disconnection proof associated with the algorithm is given in Section III-C along with a discussion of degenerate cases.

A. Algorithm

Toggle LP uses one important primitive operation called $Connect$. $Connect$ is a straight-line local planner which records witnesses to failed connection attempts. $Connect$ attempts connection between its first two parameters, sets its third parameter to the witness, and returns a boolean of success/failure.

Toggle LP is broken into two phases, setup and recursive analysis, called *Toggle Connect*. The setup phase is shown in Algorithm 1, with examples shown in Figure 1. The first step of setup is to analyze the straight line connection between s and g (Line 1). If this succeeds, then it returns true. Otherwise, c_1 is set as the witness to failure, and n is generated (Line 3). n can be generated in a number of ways, for example a random configuration might be chosen uniformly at random from C_{space} or the midpoint of $\overline{sg}$ is perturbed a distance δ along a random ray through C_{space} – these are the two methods used in this paper, see Section IV. s, g, and n define a triangle $\triangle sgn$ that is a planar subspace of C_{space}. If n is invalid (Figure 1(b)), then *Toggle Connect* is directly called with c_1, n, s, and g (Line 13). If n is valid (Figure 1(a)), then extra pre-processing takes place to initially split the triangle defined by s, g, and n. In this pre-processing, straight-line connections are attempted between s and n (Line 5) and g and n (Line 6). Collision nodes c_2 and c_3 might be returned if connections are invalid. From here, *Toggle Connect* is possibly called on each half of the triangle, returning boolean values b_1 and b_2 respectively representing the connectivity of g and n within triangle $\triangle gc_1n$ (Line 8) and of n and s within triangle $\triangle sc_1n$ (Line 10). Finally, an $\wedge$ operator is applied on the values b_1 and b_2 to yield the final connectivity of s and g through $\triangle sgn$.

The second phase of Toggle LP is a recursive analysis of the triangle $\triangle sgn$, called *Toggle Connect* (see Algorithm 2). In *Toggle Connect*, $toggle$ defines in which space validity



(a) n is valid (b) n is invalid

Fig. 1: Setup of $\triangle sgn$

Algorithm 1 Toggle Local Planner - Setup

Input: Configurations s, g
Output: boolean $connectible$

1: **if** $Connect(s, g, c_1)$ **then**
2: **Return** $true$
3: $n \leftarrow ChooseCfg()$
4: **if** $isValid(n)$ **then**
5: boolean $b_1 \leftarrow Connect(s, n, c_2)$
6: boolean $b_2 \leftarrow Connect(g, n, c_3)$
7: **if** $\neg b_1$ **then**
8: $b_1 \leftarrow ToggleConnect(c_1, c_2, s, n, false)$
9: **if** $\neg b_2$ **then**
10: $b_2 \leftarrow ToggleConnect(c_1, c_3, g, n, false)$
11: **Return** $b_1 \wedge b_2$
12: **else**
13: **Return** $ToggleConnect(c_1, n, s, g, false)$

is checked, i.e., $false$ means s and g lie in C_{obst} while $true$ implies s and g lie in C_{free}. n_1 and n_2 lie in the opposite space of s and g. $Connect$ is called between s and g (Line 1). If they are connectible, then $toggle$ is returned, i.e., when $toggle$ is $false$ a connection through C_{obst} is achieved and $false$ is returned. If no connection succeeds, then c is the witness of failure. This witness in turn splits the original triangle $\triangle sgn$ into triangles $\triangle sgn_1$ and $\triangle sgn_2$. The operator combining the triangles depends on $toggle$. If $toggle$ is $false$, then in order for a successful

Algorithm 2 Toggle Local Planner - Toggle Connect

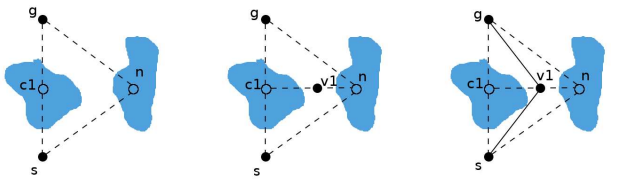
Input: Configurations s, g, n_1, n_2, boolean $toggle$
Output: boolean $connectible$

1: **if** $Connect(s, g, c)$ **then**
2: **Return** $toggle$
3: **else if** $\overrightarrow{(s, n_1)} \cdot \overrightarrow{(s, n_2)} = 0$ **then**
4: **Return** $false$
5: **if** $toggle$ **then**
6: **Return** $ToggleConnect(n_1, c, s, g, \neg toggle)$
$\vee\ ToggleConnect(n_2, c, s, g, \neg toggle)$
7: **else**
8: **Return** $ToggleConnect(n_1, c, s, g, \neg toggle)$
$\wedge\ ToggleConnect(n_2, c, s, g, \neg toggle)$

connection to occur both sub-triangles must contain a path, so $\wedge$ is considered (Line 8). When $toggle$ is true, a successful connection must occur in one or the other sub-triangle, so $\vee$ is the operator (Line 6). A brief example of the algorithm is discussed in Section III-B, and correctness is discussed in III-C.

B. Examples

In this section we discuss two simple examples. The first example shown in Figure 2 examines a simple case where a path is found between an s and g pair. Fig. 2(a) shows the failure of a straight-line connection between s and g. n is generated. In this case, n is invalid, and c_1 is the witness to the failed straight-line connection. In Fig. 2(b), *Toggle Connect* is called, with $toggle$ set to $false$. No straight-line connection through C_{obst} connects c_1 and n, so v_1 is saved as a witness. *Toggle Connect* is called recursively with the $\wedge$ operator on $\triangle sc_1n$ and $\triangle gc_1n$. In Fig. 2(c), successful connections through C_{free} are found – individually those *Toggle Connect* calls return $true$, and the $\wedge$ operator returns $true$.



(a) $\triangle sgn$ setup (b) *Toggle Connect* iteration 1 (c) $\wedge$ operator returns $true$

Fig. 2: Example of valid path

In the second example, shown in Figure 3, a disconnection is found within $\triangle sgn$. Fig. 3(a) shows the straight-line connection between s and g failing and the setup for *Toggle Connect* (setup of $\triangle sc_1n$ and $\triangle gc_1n$). c_2 and c_3 are found in this step. Fig. 3(b) shows the recursive analysis in *Toggle Connect* and its evaluation on both sides to false. Both halves found a disconnection in the first iteration. The $\wedge$ operation will return $false$ as well.

(a) $\triangle sgn$ setup (b) *Toggle Connect* recursively called, $\wedge$ operator returns $false$

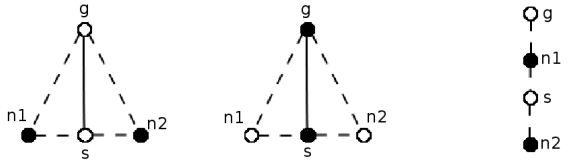
Fig. 3: Example of disconnection

C. Correctness Discussion

There are a few benefits of *Toggle Local Planner*. Firstly, as compared to the traditional straight-line LP which only considers a straight-line in C_{space}, Toggle LP extends local planning to a 2D subspace of C_{space}. Thus, allowing for Toggle LP to be as good as straight-line LP in the worst case, but, as the experiments will show, this worst case does not usually happen. Secondly, as the recursive analysis of $\triangle sgn$ is deterministic, storage needed to recreate an edge is limited to $O(1)$ configurations per edge, namely n. Lastly, we show in Proposition 1 that Toggle LP correctly terminates when no path between s and g exists in the triangle $\triangle sgn$, i.e., proof of the disconnection argument for the local planner. The proposition basically guarantees that when no path exists the planner will determine as much. However, we note that not all valid paths are found. Discovering disconnections is important in showing that likely no path exists. Toggle LP converges quickly on the disconnection within a triangle which is important for efficiency reasons. After the proof, we discuss degeneracies and ways to avoid them in implementation.

Proposition 1: When no valid path exists between s and g within $\triangle sgn$, *Toggle Local Planner* detects this and terminates.

Proof: This proposition can be proved by induction. First, we define the base cases, $m = 0$, where 2^m is the number of decomposed triangles of the recursive algorithm *Toggle Connect*, shown in Figure 4. At $m = 0$, a total of $2^m = 1$ triangles are present. Fig. 4(a) shows a successful connection through C_{obst} and $false$ is returned. Fig. 4(b) shows a successful connection through C_{free} and $true$ is returned. Fig. 4(c) is the case where the triangle has become flat, thus no path exists within the triangle – $false$ is returned. If no valid path exists, $m = 0$ holds.



(a) Connection through C_{obst}, return false (b) Connection through C_{free}, return true (c) Line, return false

Fig. 4: Base case: smallest triangles

At $m = 1$ we can combine triangles to form up to $2^m = 2^1 = 2$ total triangles. The possible triangles formed are shown in Figure 5. Fig. 5(a) shows a combination of two triangles with successful paths, by which the $\wedge$ operator will return $true$. Fig. 5(b) shows a combination of two triangles with no successful paths, by which the $\vee$ operator will return $false$. If a flat triangle were to become combined with another triangle, it is easy to see that when no true path exists, then *Toggle Connect* will return $false$ because flattened triangles show a disconnection. Additionally, the case of two lines being combined returns false, as it is still a line. Again when no path exists, $m = 1$ holds.

Assume that for $m = k$, no path exists within the possible 2^k triangles composed, then by the inductive hypothesis we will show this holds for $m = k + 1$. Figure 6 shows the various combinations of up to 2^k additional triangles. Recall

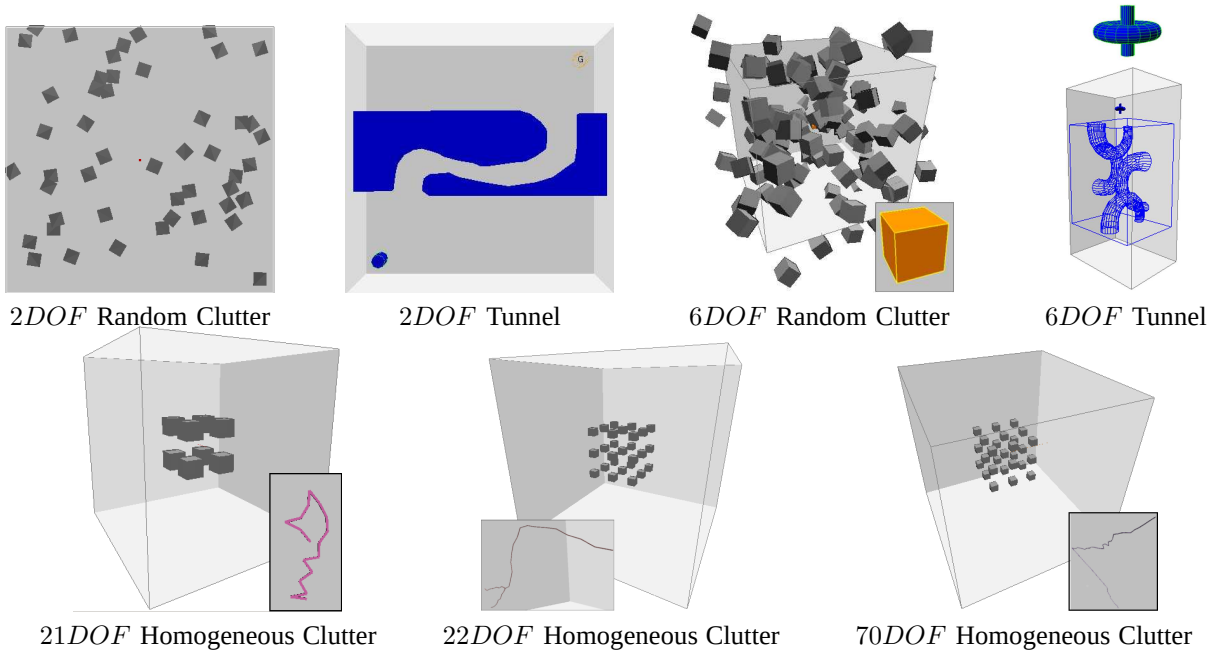
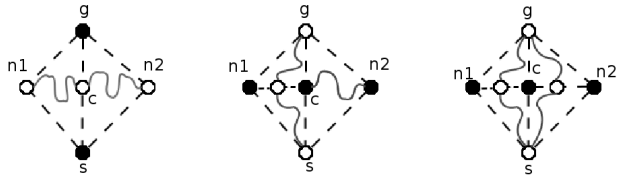


Fig. 7: Environments with varying complexities of robots.

(a) Connection through C_{obst}, return false

(b) Connection through C_{free}, return true

Fig. 5: Case $m = 1$: two triangles



(a) $\vee$ returns false (b) $\wedge$ returns false (c) $\wedge$ returns false

Fig. 6: Base case: smallest triangles

$2^{k+1} = 2^k + 2^k$ and that this is an upper bound on the number of smaller triangles composing the overall triangle. Fig. 6(a) shows the case where one triangle is composed with another. The $\vee$ operator will return $false$ because the original assumption is that no path exists. If a path existed then one of the halves could return $true$, thus returning $true$ for the entire local planning attempt. Fig. 6(b) shows a blocking case with the $\wedge$ operator. No matter the outcome of the other triangle, no path exists and $\wedge$ operator returns $false$. We see that at $m = k + 1$ a false path will still be returned. ■

D. Degeneracies

As with most geometric algorithms, there are degeneracies. In Toggle LP, degeneracies are caused when a cycle of nodes is being returned as witnesses from successive straight-line connections. The simplest case of this is shown in Figure 4(c). In this case, when a connection between s and g is attempted, n_1 is returned. Then in the next iteration of *Toggle Connect*, when connection between n_1 and n_2 is attempted, s is returned. This can cause infinite recursion. To avoid this one could implement a history of witnesses returned and stop when a cycle is detected, or more simply a limit to the recursive depth could be enforced. In this paper, we implemented a maximal recursive depth of 5. We experimentally found this number to capture most successful cases. We, however, do not show the full analysis here as that is not the focus of the paper. It should be noted that degeneracies can exist with more than a 1 node cycle. In fact we believe many cycles are possible but have only found up to a 12 node cycle in high DOF cases. These cycles are found because of the binary analysis of the local planning edges, along with local plans evaluated only up to a certain resolution.

IV. Experimental Analysis

A. Experimental Setup

Toggle Local Planner (Toggle LP) was implemented using the C++ motion planning library developed by the Parasol Lab at Texas A&M University. RAPID [9] is used for collision detection computations. A wide range of problems with varying DOFs were tested. The scenarios are shown in Figure 7. In the robots with $DOF \leq 21$, uniform sampling and obstacle-based sampling [1] occurred followed by con-

nections in C_{space}. In the other cases, sampling and connection occurred in Reachable Distance space (RD_{space}) [25] to more efficiently map high dimensional C_{space}. Connections are attempted to a configuration's k-closest nodes based upon a brute force neighborhood search using a scaled Euclidean distance metric. The Toggle LP is compared to the straight-line local planner. For two of the cases ($DOF = 6$ and $DOF = 21$) Rotate-At-S[2] is also compared ($s = 0.5$ and $s = 0.0$), and in two other cases ($DOF = 2$ and $DOF = 6$) an A^*-like distance-based search [2] is compared. For the Toggle LP, recursive analysis of triangles is allowed to go to a recursive depth of 5. In $DOF \leq 21$, Toggle LP choses n by finding the midpoint of $\overline{sg}$ and pushing this node along a random directional ray at a distance $\delta = distance(s, g)/2$. In $DOF \geq 22$, we choose n uniformly at random from RD_{space}.

The environments and robot types explored are shown in Figure 7 and described as:

- *2DOF Random Clutter* - Randomly scattered squares around the workspace. Robot is near point sized.
- *2DOF Tunnel* - A simple s-like tunnel in 2-dimensions. Used in comparing query results in which a solution path must traverse the tunnel. Robot is a circular robot.
- *6DOF Random Clutter* - Randomly scattered cubes around the workspace. Robot is a cube.
- *6DOF Tunnel* - A simple s-like tunnel in 6-dimensions. Used in comparing query results in which a solution path must traverse the tunnel. Robot is in the shape of a spinning top.
- *21DOF Homogeneous Clutter* - 8 centrally located cubes surrounded by free space. Robot is a 16 link free base articulated linkage.
- *22DOF Homogeneous Clutter* - 27 centrally located cubes surrounded by free space. Robot is a free base fork-like linkage. 9 links construct the arm from which two 4 link linkages extend.
- *70DOF Homogeneous Clutter* - 27 centrally located cubes surrounded by free space. Robot is a 65 link free base articulated linkage.

The metrics being analyzed are the number of edges/nodes within the final roadmap, the total number of connected components of the final roadmap, roadmap connectivity, and the number of CD calls (used as a cost metric). Roadmap connectivity is defined as the ratio of the size of the largest connected component to the number of nodes in the roadmap.

The experiments are broken down into three sections. Firstly, Section IV-B compares the number of edges produced with a fixed number of nodes and connections. Secondly, Section IV-C compares the number of nodes required to reach a fixed number of edges is compared. Lastly, Section IV-D compares the efficiency of a simple PRM using different local planners to solve example queries.

B. Comparing Number of Edges Generated

In this first experiment, we compare the number of edges successfully created in our roadmap when we fix the total number of nodes and connection attempts. For $DOF = 2$ and $DOF = 6$, 100 nodes were generated and all possible edge pairs were attempted. For $DOF = 21$ and $DOF = 22$, 500 nodes were generated and the $k = 16$-closest connections were attempted for each node. In the $DOF = 70$ case, 50 nodes were generated and the $k = 4$-closest connections were attempted for each node. Results are averaged over 10 random seeds and shown in Table I. In environments labeled Mix, half of the nodes generated were from uniform random sampling and half of the nodes were generated from obstacle-based sampling.

Local Planner	Edges Created	Number of CCs	Roadmap Conn.	CD Calls
$2DOF$				
Toggle	3924	1.1	0.999	4.32e6
SL	2338	1.2	0.998	7.59e5
ASD	5081	1.1	0.999	1.90e7
$2DOFMix$				
Toggle	3722	1.7	0.993	4.07e6
SL	2045	2.0	0.990	6.51e5
ASD	4907	1.4	0.996	1.78e7
$6DOF$				
Toggle	4317	3.6	0.974	1.88e6
SL	2848	3.7	0.973	5.88e5
R0.5	2841	3.7	0.973	1.53e6
R0.0	2805	3.8	0.972	1.24e6
ASD	6720	3.6	0.974	2.16e7
$6DOFMix$				
Toggle	2919	6.2	0.946	1.59e6
SL	1745	6.4	0.944	4.73e5
R0.5	1753	6.4	0.944	8.01e5
R0.0	1027	13.8	0.871	6.12e5
ASD	5504	5.6	0.952	2.35e6
$21DOF$				
Toggle	9035	1.7	0.999	5.63e6
SL	8679	1.9	0.998	5.01e6
R0.5	8711	1.8	0.998	5.70e6
R0.0	8860	1.8	0.998	5.58e6
$21DOFMix$				
Toggle	8626	3.2	0.996	5.79e6
SL	8172	3.3	0.995	4.98e6
R0.5	8247	3.3	0.995	5.92e6
R0.0	8389	3.3	0.995	5.61e6
$22DOF$				
Toggle	1170	115.6	0.755	1.95e6
SL	752	190	0.572	1.00e5
$70DOF$				
Toggle	46	27.5	0.24	3.39e5
SL	38.4	31.4	0.22	3.96e4

TABLE I: Results for various clutter environments. Mix implies mixed sample types. SL is the straight-line local planner. R0.5 is the Rotate-At-S planner with $s = 0.5$. R0.0 is the Rotate-At-S planner with $s = 0$. ASD is an A^*-like distance-based search.

In these experiments, we can clearly see that in all cases Toggle LP makes a significant number of additional connections as compared to the less powerful local planners, straight-line and Rotate-at-S, and almost as many connections as the very powerful A^*-like local planner. From this observation, Toggle LP approaches the capabilities of very powerful local planners while its cost in terms of CD calls is only marginally larger than less powerful local planners. The second item to notice is the scalability of the method for high DOF cases. In the higher dimensions, using Toggle LP successfully reduces the number of CCs in the roadmap,

Local Planner	Nodes Required	Number of CCs	Roadmap Conn.	CD Calls
$2DOF$				
Toggle	26.3	1.3	0.989	1.56e5
SL	34.3	1.7	0.955	5.10e4
ASD	24.0	1.1	0.996	5.77e5
$6DOF$				
Toggle	25.1	1.4	0.986	7.26e4
SL	31.5	2.3	0.960	3.53e4
R0.5	31.6	2.3	0.958	7.08e4
R0.0	32.0	2.7	0.949	6.16e4
ASD	20.1	1.5	0.977	5.59e5
$21DOF$				
Toggle	18.9	1.0	1.000	2.23e5
SL	19.7	1.0	1.000	1.86e5
R0.5	19.6	1.0	1.000	2.42e5
R0.0	19.0	1.0	1.000	2.33e5
$22DOF$				
Toggle	124.6	41.2	0.539	3.88e5
SL	186.5	96.0	0.267	3.23e4
$70DOF$				
Toggle	45.1	22.2	0.393	5.03e5
SL	57.4	33.9	0.267	6.00e4

TABLE II: Results for various clutter environments. Maps were constructed until a set number of edges were reached. SL is the straight-line local planner. R0.5 is the Rotate-At-S planner with $s = 0.5$. R0.0 is the Rotate-At-S planner with $s = 0$. ASD is an A^*-like distance-based search.

increasing roadmap connectivity. This implies that more important connections are made in these high DOF cases as compared to straight-line local planning. Thirdly, we know that the Toggle LP is versatile to various clearances of nodes ranging from close to obstacles in the case of obstacle-based samples to higher clearances with uniform random samples.

C. Generating Fixed Number of Edges

In this section, we explore the efficiency of Toggle LP by keeping the number of edges constant and seeing how many nodes a planner needs to reach the edge limit. In $DOFs \leq 22$, after a node is generated, connections are attempted to its $k = 10$ closest neighbors. In $DOF = 70$, after a node is generated, connections are attempted to its $k = 4$-closest neighbors. Results over 10 random seeds in the cluttered environments are shown in Table II.

As seen in the table, Toggle LP reduces the number of nodes needed in the roadmap, as well as the number of CCs in the roadmap. Secondly, the CD calls are more comparable to the less expensive and less powerful local planners than to the more expensive A^*-like planner. Again, the Toggle LP shows results of this manner in high DOF cases. In terms of efficiency, we notice not only an increased connectivity, but reduced map-size reduces the memory footprint of the roadmap, while keeping the cost-to-connect relatively cheap. As we have seen, Toggle LP increases connectivity and reduces the map size, so we would expect that in example query problems the Toggle LP should outperform the other local planners.

Local Planner	Nodes Required	Edges Required	Number of CCs	CD Calls
$2DOF$				
Toggle	40.4	413	1.1	2.55e4
SL	138.8	2039	1.1	3.41e4
ASD	30.8	421	1.0	9.96e5
$6DOF$				
Toggle	511	3395	20.8	6.40e5
SL	751	5347	31.1	6.36e5
R0.5	858	5061	41.1	7.65e5
R0.0	1020	9920	51.9	1.04e6
ASD	57.3	483	1.4	1.03e6

TABLE III: Results for tunnel environments. Maps were constructed until example query was solved. SL is the straight-line local planner. R0.5 is the Rotate-At-S planner with $s = 0.5$. R0.0 is the Rotate-At-S planner with $s = 0$. ASD is an A^*-like distance-based search.

D. Example Queries

In this section, we compare Toggle LP to other local planners for an example query. In both examples, the robot must traverse a narrow passage – a tunnel in the work space. Roadmaps are iteratively constructed one node at a time until the query is solved. Connections are attempted to a node's $k = 10$-nearest neighbors. Results are averaged over 10 random seeds and shown in Table III.

Again we can see the benefit of Toggle LP. In the $DOF = 2$ case, not only does Toggle LP reduce the overall size of the roadmap, but it is also cheaper than solving the problem with straight-line local planning. In the $DOF = 6$ case, the Toggle LP greatly reduces the size of the overall roadmap while keeping the cost comparable to straight-line planning. As we can see from these results, Toggle LP shows real benefit for solving queries in the presence of narrow passages.

V. Conclusion

In this paper, *Toggle Local Planner* (Toggle LP) is introduced as a new simple method for defining and validating simple paths between two configurations. The main idea of the Toggle LP is to extend local planning to a 2 dimensional plane of C_{space}. Additionally, it is proven for the Toggle LP that when no valid path exists in this subspace between two configurations then a proof of disconnection is found. The Toggle LP is an efficient local planner requiring little extra storage per edge verified which operates in both C_{space} and RD_{space}. Experimental results showed the benefits of this method for a broad range of DOFs, up to $DOF = 70$, and a range of environments and problems including example queries on a roadmap.

In the future, exploration into better methods for choosing the third node n could be explored, as well as extending this idea to a 3 (or more) dimensional subspace of C_{space}. A different evaluation strategy of $\triangle sgn$ can also be taken to better guarantee valid paths to be found.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Robotics:*

The Algorithmic Perspective, pages 155–168, Natick, MA, 1998. A.K. Peters. Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. Automat.*, 16(4):442–447, August 2000.

[3] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen. Disconnection proofs for motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1765–1772, 2001.

[4] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 521–528, 2000.

[5] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1018–1023, May 1999.

[6] J. Denny and N. M. Amato. Toggle PRM: Simultaneous mapping of C-free and C-obstacle - a study in 2D -. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2011.

[7] R. Geraerts and M. Overmars. Reachability-based analysis for probabilistic roadmap planners. *In Journal of Robotics and Autonomous Systems*, 55:824–836, 2007.

[8] B. Glavina. Solving findpath by combination of directed and randomized search. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1718–1723, 1990.

[9] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Comput. Graph.*, 30:171–180, 1996. Proc. SIGGRAPH '96.

[10] K. K. Gupta and Z. Guo. Motion planning for many degrees of freedom: Sequential search with backtracking. *IEEE Trans. Robot. Automat.*, 11(6):897–906, 1995.

[11] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom – random reflections at c-space obstacles. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3318–3323, 1994.

[12] D. Hsu, T. Jiang, J. Reif, and Z. Sun. Bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4420–4426, 2003.

[13] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[14] Y. K. Hwang and P. C. Chen. A heuristic and complete planner for the classical mover's problem. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 729–736, 1995.

[15] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 473–479, 1999.

[17] J.-M. Lien, O. B. Bayazit, R.-T. Sowell, S. Rodriguez, and N. M. Amato. Shepherding behaviors. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4159–4164, April 2004.

[18] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[19] C. Mirolo and E. Pagello. A practical motion planning strategy based on plane-sweep approach. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2705–2712, 1997.

[20] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin/Heidelberg, 2005. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Utrecht/Zeist, The Netherlands, 2004.

[21] C. L. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. Technical Report TR2000-365, Computer Science, Rice University, Houston, TX, 2000.

[22] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, pages 421–427, San Juan, Puerto Rico, October 1979.

[23] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. (RESAMPL): A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[24] A. P. Singh, J.-C. Latombe, and D. L. Brutlag. A motion planning approach to flexible ligand binding. In *Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pages 252–261, 1999.

[25] X. Tang, S. Thomas, P. Coleman, and N. M. Amato. Reachable distance space: Efficient sampling-based planning for spacially constrained systems. *Int. J. Robot. Res.*, 29(7):916–934, 2010.

[26] P. Watterberg, P. Xavier, and Y. Hwang. Path planning for everyday robotics with sandros. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1170–1175, 1997.

[27] S. W. H. Wong and M. Jenkin. Exploiting collision information in probabilistic roadmap planning. In *Int. Conf. on Mechatronics*, 2009.