Generation of Diverse Paths in 3D Environments

Ana Huamán Quispe Tobias Kunz Mike Stilman

***Abstract*—In this paper we propose a deterministic algorithm to produce a set of *diverse paths* between a given start and goal configuration in 3D environments. These *diverse paths* have the following properties: 1) They are bounded in length and 2) They are non-visibility-deformable into one another. Maintaining multiple path alternatives is important in practical applications such as planning in dynamic environments, in which a path may unexpectedly become infeasible due to unforeseen environmental changes. We present our approach, the distance cost considered (based on the *path deformability concept* previously introduced in [11]) and finally show results of simulated experiments that exemplify the effectiveness of our algorithm.**

I. Introduction

Finding *a single* path between a goal and start configuration is a canonical motion planning problem. However, there are many situations in which more than one path must be considered. For instance, in multi-robot mapping applications, it is reasonable to design different trajectories for each agent such that they cover diverse regions of free space [3]. Other applications include motion planning in dynamic environments, in which considering multiple paths can be useful if one of them becomes infeasible due to unexpected changes in the environment. In general, if a robotic agent is provided with a *diverse* set of paths, it has the flexibility to choose what option is best at any given situation. The question then arises as to how to classify these paths such that they are as different (*diverse*) as possible from one another.

Homotopy is a classical approach to classify paths. Two paths, having the same start and goal configuration, belong to the same homotopy class if they can be continuously deformed into one another. Different homotopy classes may arise due to obstacles in the environment. Fig. 2(a) shows a 2D scenario with an obstacle and two paths. These two paths belong to different homotopic classes, since there is no smooth deformation that can convert one into the other.

Intuitively, we see that 2D obstacles arise different homotopy classes. This, however, is not the general case in 3D environments. Consider Fig.2(b), depicting one obstacle and four different paths. These paths belong to the same homotopy class since there exists a deformation (albeit complex) that can turn any of these paths into another. In practical applications, however, we might want to differentiate between paths that are *easily* deformable [11], for which the homotopy criterion may not be suitable.

The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: ahuaman3@gatech.edu,tobias@gatech.edu, mstilman@cc.gatech.edu

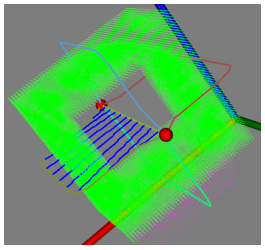


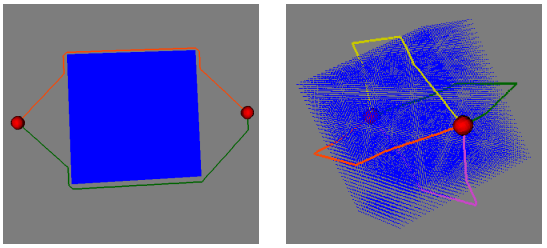
Fig. 1. 4 paths generated for a donut-shaped object

A few authors have proposed to use different criteria to classify paths [11], [14]. In this paper, we propose an algorithm to construct a set of *diverse paths* based on the concept of path visibility deformation [11], [19]. We generate a set of paths that combines two desired properties: diversity and quality. In our approach quality is path length. We ensure path quality by defining a limiting factor by which the length of any path in the set can exceed the optimal path length. Diversity is ensured by generating candidate paths that go through selected via points. From these candidate paths we select paths that are non-visibility-deformable into each other and are far apart from each other.

The rest of this paper is organized as follows. Section II presents previous work relevant to path classification. Section III explains some background concepts that link homotopy with the deformation concept used in this paper. Section IV presents our approach as well as the algorithms implemented. Finally, section V presents experimental results showing diverse sets of paths generated for different test environments.

II. Previous Work

Previous efforts to produce *diverse* paths have largely been focused on homotopy. Early work on homotopy-based path



(a) 2D paths of different homotopic classes (b) 3D paths homotopically equivalent

Fig. 2. Comparison of homotopy in 2D and 3D

classification includes geometric-based approaches such as [8], [9]. and methods based on Probabilistic Roadmaps [13], such as the work of Schmitzberger [18], who proposed a method to build PRMs that encode all the homotopy classes in 2D environments. More recent approaches successfully identify homotopy classes and their corresponding optimal paths in 2D discretized scenarios. Examples of these are [1], which characterize the homotopy classes by using Complex Analysis. A rather simpler approach, proposed by Igarashi in [10] uses overlapped manifolds to represent homotopy classes for cabled robots. Additionally, a few approaches (PRM-based) departed from the strict context of homotopy and instead proposed to include cycles in the map construction step in order to consider more than one possible path between two configurations [6], [16].

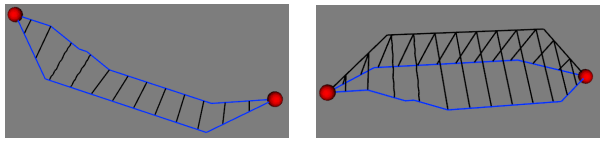
On the side of 3D path classification, Battacharya presented in [2] a planner that identify homotopy classes and produces optimal paths for each of them. However, as it was noted in [16], finding paths in different homotopy classes may fail to capture the variety of existing representative paths since they might belong to the same homotopy class but still be hard to deform into one another: A 3D scene with multiple finite obstacles can easily have only one homotopic family. In general, homotopy classes in three dimensions can be induced only by objects with infinite dimensions in two directions or by objects with holes on them [2].

Alternative approaches not based on homotopy have been proposed to generate diverse paths. In [11], Jaillet presented a PRM-based approach to produce diverse paths based on *path deformability*, which consider less strong (easier) deformations than homotopy. This method is applicable for higher dimensions (such as 3D) but since it is a probabilistic approach, it can neither guarantee that the paths produced will be bounded in length nor that they will be the same each time they are generated.

The term *path diversity* has also been used by [7] to define path sets and extensions of this work applied to offline generation of local paths are presented in [14]. Under this approach, a set of paths is called *diverse* if it minimizes the *dispersion metric* between them. Our approach is similar to [7] in the sense that our algorithm produces a *sequence* of paths that attempts to maximize the diversity between them. In our context, *diversity* is quantified by the accumulated distance between a path and the rest of paths in the set, not by dispersion (although these metrics are related). Another important difference is that [7] generates an initial set of paths by combining any set of discrete inputs in a given model. While we also generate a initial set of candidate paths, our approach is different: We discretize our 3D environment into a 26-connected graph and then generate a set of paths bounded in length by using a graph-search based method (as it will be explained later in Section IV).

III. Background

We will briefly review the concept of homotopy [15] and path deformability [11].



(a) 1st order deformation (b) 2nd order Deformation

Fig. 3. Deformation examples

A. Homotopy

Definition 1: Let P_0 and P_1 be two paths in a topological space $\mathcal{X}$ with the same initial point $p_0 = P_0(0) = P_1(0)$ and terminal point $p_1 = P_0(1) = P_1(1)$. P_0 is said to be *homotopic* to P_1 if there is a continuous function $H : I^2 \to \mathcal{X}$ such that

$$\begin{aligned} H(0,t) &= p_0 \qquad 0 \le t \le 1 \\ H(1,t) &= p_1 \qquad 0 \le t \le 1 \\ H(x,0) &= P_0(x) \qquad 0 \le x \le 1 \\ H(x,1) &= P_1(x) \qquad 0 \le x \le 1 \end{aligned} \tag{1}$$

where the function H is called a *homotopy* connecting P_0 to P_1.

Notice that H represents *any* continuous function, which in turn may produce very complex deformations. For practical uses, however, we want to consider only *simpler* deformations, which are more likely to occur. Jaillet et al introduced the concept of *K-order deformations* [11], which we borrow here and explain below:

B. K-order Deformations

Definition 2: A K-order deformation is a particular homotopic deformation such that each curve transforming a point from P_0 into a point of P_1 is an angle line of K segments

Figures 3(a) and 3(b) show a 1st and 2nd order deformations respectively. In general, a K-order deformation can be obtained from the concatenation of K ruled surfaces.

In this paper we consider 1st order deformations (also called *visibility deformations*). Among other reasons, these deformations are important since:

- They capture the visibility between two paths: If a path P_0 can be deformed into a path P_1 with a first-order deformation, then the corresponding points between both paths are *visible*, meaning that a straight line can join them without crossing any obstacle.
- Given two simple paths in a discretized environment, we can determine if they are first-order deformable by simply applying collision-testing to the lines that join their corresponding points.

IV. Algorithm

The algorithm works by pruning and selecting paths from an initial set of candidate paths. Section IV-A describes the generation of the set of candidate paths. Section IV-B filters the set of candidate paths to only include high-quality paths of bounded length. Section IV-C describes how we select a diverse set of paths from the set of bounded paths.

A. Candidate Paths

A possible set of candidate paths would be the set of all possible simple paths (i. e. paths without loops) between the given start and goal. However, calculating the number of paths between two given vertices in a graph is #P-complete [17], so it would be computationally expensive to enumerate all these paths. We could also attempt to frame the generation of the candidate paths as a *K shortest simple paths* problem. However, besides being also computationally intensive [12], we do not know how many paths will be in the set beforehand.

Instead, we calculate the shortest paths from v_{start} to v_{goal} through one waypoint v for all vertices $v \in \mathcal{V}$. Since the number of paths generated this way is still big, we further restrict the selection of v to vertices that are roughly middle points in the paths to-be-generated (line 2 of Alg.1). We could have also considered shortest paths through two or more waypoints. If the number of waypoints was unlimited, we would actually find all possible simple paths (and also some non-simple ones). By choosing the number of waypoints we trade-off complexity and completeness. We choose to consider only a single waypoint because that makes it especially simple to calculate all candidate paths. We only need to execute two Dijkstra searches [5] rooted at v_{start} and v_{goal}. This gives us the distances and shortest paths from every vertex v to v_{start} and v_{goal}. Concatenating the two yields the shortest paths from v_{start} to v_{goal} through a vertex v.

B. Bounded Paths

In order to ensure the quality of the paths in the set, we filter out all paths in the candidate set that exceed the length of the optimal path by a certain factor. We only retain paths that satisfy

$$||P_i|| \leq \alpha_B P_1, \forall i \in [1, n] \tag{2}$$

The set of candidate paths as described in Section IV-A is actually never explicitly calculated. Instead, the candidate paths are filtered from the beginning to only include paths that satisfy Eq. 2. Algorithm 1 combines the two steps of generating the candidate paths and filtering out paths that do not satisfy the bound on path length. D_{start} and D_{goal} are data structures representing the result of the Dijkstra searches rooted at the v_{start} and v_{goal} respectively. $D_{\text{start}}(v)$ is the distance of v from v_{start}. P is the shortest path from v_{start} to v_{goal} and α_B is the maximum factor by which the paths are allowed to exceed the optimal path length.

An example of a set of bounded paths generated using this approach is shown in Fig. 4.

C. Diverse Paths

From the set of bounded paths we iteratively build the set of diverse paths by choosing one path P_i from the set of bounded paths and add it to the set of diverse paths $\mathcal{P}$ until we reached the desired number of paths. In each step we pick the one path that

- maximizes the distance to all the other paths $P_1, P_2, P_3, ..., P_{i-1}$ already in the set of diverse paths and that

Algorithm 1: Get Bounded Paths

Input: D_{start}, D_{goal}, P, α_B
Output: $\mathcal{P}_B$

```
1 forall the v ∈ V do
2     if D_start(v) ≈ D_goal(v) then
3         if D_start(v) + D_goal(v) ≤ α_B · P.size then
4             P_sv ← getShortestPath(v, D_start) ;
5             P_vg ← getShortestPath(v, D_goal) ;
6             P ← concatenate(P_sv, P_vg) ;
7             P_B.add(P) ;
8 return P_B ;
```

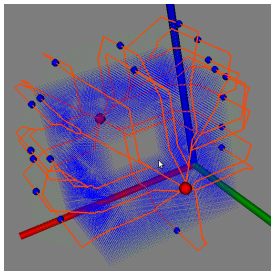


Fig. 4. Sample set of paths in $\mathcal{P}_B$

- is not visibility-deformable into any of the other paths that are already in the set of diverse paths.

Definition 3: Given two paths P_A and P_B and a given number of waypoints cp, we define the distance δ between these two paths as:

$$\delta(P_A, P_B) = \sum_{i=1}^{cp} d(p_A^i, p_B^i) \tag{3}$$

Algorithm 2 shows the main high-level routine to produce diverse paths. It receives as inputs the start and goal positions (v_{start}, v_{goal}), the graph representation of the 3D space ($\mathcal{G}$) and the number of paths desired (the algorithm stops if it cannot find more non-visibility-deformable paths, so this parameter is a maximum).

Our algorithm first precomputes the distances and shortest paths from every vertex to the start and goal vertices using Dijkstra's algorithm. We then calculate the shortest path between v_{start} and v_{goal}, which is added as the first path (P_1) in our path set $\mathcal{P}$ and defines the upper bound on the length of all considered paths. The generation of bounded paths is performed in line 4 of Alg. 2.

Once we have the initial set of bounded paths $\mathcal{P}_B$, we sequentially produce paths P_i that are *non-visibility-deformable* with respect to the existing paths $P_1, P_2, P_3, ..., P_{i-1}$. To make sure that each new path generated will be non-visibility-deformable with respect to the previous paths, we prune the set of bounded paths at each iteration(line 7 of Alg. 2) such that the bounded paths considered at each step i are non-visible w.r.t P_{i-1}.

Algorithm 2: Generate Diverse Paths

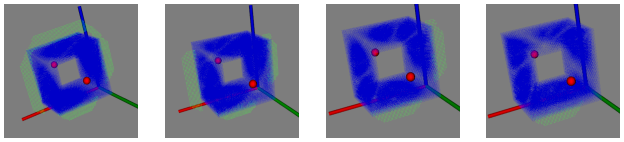
Input: v_{start}, v_{goal}, $\mathcal{G}$, n
Output: $\mathcal{P}$ `//Set of paths non-visibility-deformable`

1 $D_{\text{start}} \leftarrow$ `Dijkstra(`v_{start}, $\mathcal{G}$`);`
2 $D_{\text{goal}} \leftarrow$ `Dijkstra(`v_{goal}, $\mathcal{G}$`);`
3 $P_1 \leftarrow$ `getShortestPath(`v_{start}, D_{goal}`);`
4 $\mathcal{P}$`.add(`P_1`);`
5 $\mathcal{P}_B \leftarrow$ `getBoundedPaths(`D_{start}, D_{goal}, P_1, α_B`);`
6 **for** $i \leftarrow 2$ **to** n **do**
7 $\mathcal{P}_B \leftarrow$ `getNoDeformPaths(`$\mathcal{P}_B$, P_{i-1}, cp`);`
8 **if** $\mathcal{P}_B \neq \emptyset$ **then**
9 $P_i \leftarrow$ `getMaxDistancePath(`$\mathcal{P}_B$`);`
10 $\mathcal{P}$`.add(`P_i`);`
11 **else**
12 `break;`
13 **return** $\mathcal{P}$;

Algorithm 3 shows the process that prune the existing $\mathcal{P}_B$ with respect to P_i. The algorithm tests each path $P \in \mathcal{P}_B$ against P_i and prune it if it they are visible w.r.t. each other. To determine visibility our algorithm builds a discretized ruled-surface between paths P and P_i and test if there is any obstacle between them. If there is an obstacle, then the path is preserved (since it is not visible). If there is not, then the path is pruned. Figure 5 shows the evolution of the pruning process: The green dots represents the middle points of the paths in $\mathcal{P}_B$ after each iteration. As it can be seen, the number of paths considered decreases at each step since a new path is added and hence the visible paths w.r.t. it must be pruned.

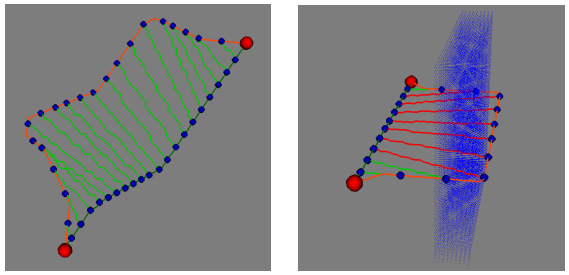
The visibility test is shown in line 5 of Algorithm 3. The `FreeLine` function receives as input a pair of corresponding points of paths P and P_i, which are generated by equally dividing each of these paths in cp segments (lines 1 and 3 of Alg. 3). If any of the points in the line joining these corresponding points is not free then the whole line is deemed non-visible (a visualization of this process is shown in Fig. 6).

Finally, once the test of visibility is done, a new set of non-visibility-deformable paths has been generated. In order to select one of them to be the next P_i to be added to $\mathcal{P}$, we choose the path $P \in \mathcal{P}_B$ that have the maximum added distance (δ) with respect to P_{i-1} (line 9 of Alg. 2).



(a) $i = 1$ (b) $i = 2$ (c) $i = 3$ (d) $i = 4$

Fig. 5. Pruning visualization: After a path P_i is added to $\mathcal{P}$ the paths in $\mathcal{P}_B$ is decreased. Notice how the surviving paths agglomerate to the sides of the obstacle that are opposed to P_i (not shown)



(a) No obstacles between paths (b) Obstacle between paths

Fig. 6. Example of visibility test procedure: Green lines represent free visibility lines and red lines are not visible due to the presence of obstacles. Blue markers represents the checkpoints ($C_p[i]$ and $C_{ref}[i]$) of each path tested)

Algorithm 3: Get Non-Deformable Paths

Input: $\mathcal{P}_B$, P_{ref}, cp
Output: $\mathcal{P}_D$; `// Non-Deformable paths`

1 $C_{ref} \leftarrow$ `getCheckpoints(`P_{ref}, cp`)` ;
2 **forall the** $P \in \mathcal{P}_B$ **do**
3 $C_p \leftarrow$ `getCheckpoints(`P, cp`)` ;
4 **for** $i \leftarrow 1$ **to** cp **do**
5 **if** `isFreeLine(`$C_{ref}[i]$, $C_p[i]$`)` *is false* **then**
6 $\mathcal{P}_D$`.add(`P`)` ;
7 `break` ;

V. Experiments

To assess the variety of paths produced by our algorithm we produced a set of tests with different obstacle configurations.

In all the experiments the graph $\mathcal{G}$ that represents the 3D environment was discretized to voxels of 1.25 cm×1.25 cm×1.25 cm in a volume of 1 m×1 m×1 m (512000 voxels).

The line evaluation of the visibility test(line 5 of Alg.3) was implemented using the Bresenham's algorithm [4] which is a simple yet effective method to calculate which points in the discretized space are part of a line. We preferred this method over other popular, more exact methods (such as Wu's Algorithm [20]) since it is simpler to implement and has higher speed.

We present the results of 5 experiments in which we used box-shaped objects as obstacles. Since these objects are finite and have neither holes nor handles, they do not arise different homotopy classes, so all paths presented are homotopically equivalent (with the exception of test 1 which has an obstacle with a hole, hence it presents two homotopy classes). The paths obtained are shown in Fig. 7, 8, 9, 10 and 11

Comparison of path lengths is presented in Table I. Due to the bound in length, only paths short enough are considered. Notice that for Experiments 3 and 7 we only present three paths that are non-visible with respect to each other (due to the particular geometry of the tested environment). For all experiments we used a bound factor $\alpha_B = 2.0$ and 10 checkpoints (cp) to measure line collisions.

TABLE I
PATH LENGTHS

Test	Length(cm)				
	P_1	P_2	P_3	P_4	P_5
Test 1	80.6	108.6	113.2	122.6	124.9
Test 2	64.0	89.8	102.8	104.8	105.6
Test 3	64.0	111.6	113.6	–	–
Test 7	82.3	86.8	89.0	–	–
Test 8	60.8	61.4	70.7	74.4	–

Table II shows the size of the $\mathcal{P}_B$ set at each iteration i. It is important to observe that the paths are calculated only once (at the start of the algorithm). The pruning then is only important to limit the number of visibility checks to generate the new path P_i.

TABLE II
PATH PRUNING

Test	Size of (Pruned) $\mathcal{P}_B$ at each iteration i				
	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
Test 1	8544	7441	3963	901	462
Test 2	7650	3130	1731	1185	611
Test 3	8633	1041	628	–	–
Test 7	15335	4362	261	–	–
Test 8	21038	20288	17514	12236	–

Table III shows the time distribution for the test cases. Notice that the times could have been further reduced if we required less diverse paths (in the test cases we found all the possible paths that were non-visible with respect to each other). Further, if time constraints are present, our algorithm can be stopped at any time, having produced as many paths as it was feasible.

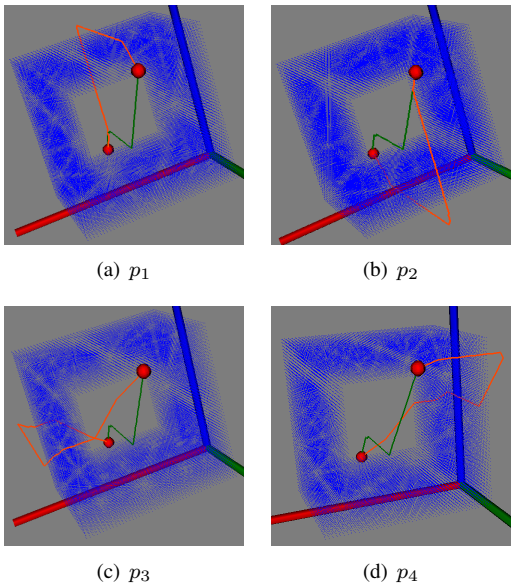
TABLE III
TIME MEASUREMENTS

Test	Time Distribution(seconds)				
	Dij. Start	***Dij. Goal***	***Initial Path Set***	***Iter.***	***Total***
Test 1	0.64	0.65	1.35	2.77	5.62
Test 2	0.73	0.74	1.31	2.06	4.86
Test 3	0.70	0.71	1.61	1.67	4.70
Test 7	0.86	0.85	2.8	3.42	7.97
Test 8	0.80	0.80	3.81	7.39	12.87

VI. CONCLUSION AND FUTURE WORK

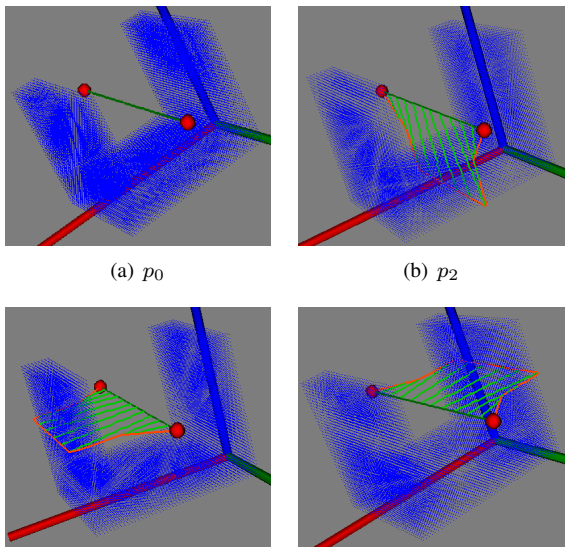
We have presented an algorithm to generate diverse paths in 3D environments. Our approach finds paths that are bounded in length and cannot be deformed by means of visibility-deformations from one to another. We also present a simple yet effective distance cost to quantify the aggregated distance between paths.

As future work, we would like to experiment with second-order deformations and make a qualitative comparison with the paths obtained with the method presented. Also we are currently attempting to find better ways to generate the initial set of paths $\mathcal{X}$ with the least possible number of paths that are yet representative of the whole set of paths between v_{start} and v_{goal}.



(a) p_1 (b) p_2 (c) p_3 (d) p_4

Fig. 7. Results of test 1: Paths around an object with a hole (2 homotopy classes)



(a) p_0 (b) p_2 (c) p_3 (d) p_4

Fig. 8. Results of test 2: U-shaped obstacle (1 homotopy class)

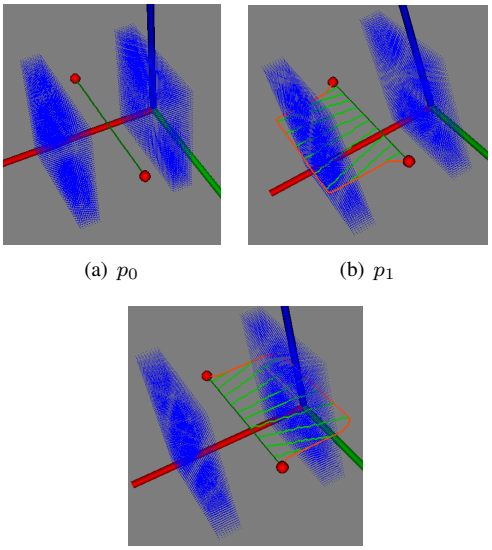
REFERENCES

[1] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Search-Based Path Planning with Homotopy Class Constraints. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[2] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Identification and Representation of Homotopy Classes of Trajectories for Search-based Path Planning in 3D. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[3] Frederic Bourgault, Alexei A. Makarenko, Stefan B. Williams, Ben Grocholsky, and Hugh F. Durrant-Whyte. Information Based Adaptive Robotic Exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*, pages 540–545, 2002.

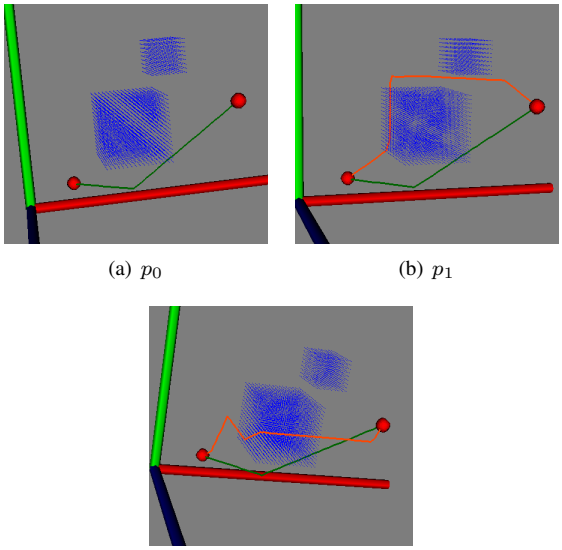
[4] Jack Elton Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.



(a) p_0 (b) p_1

(c) p_2

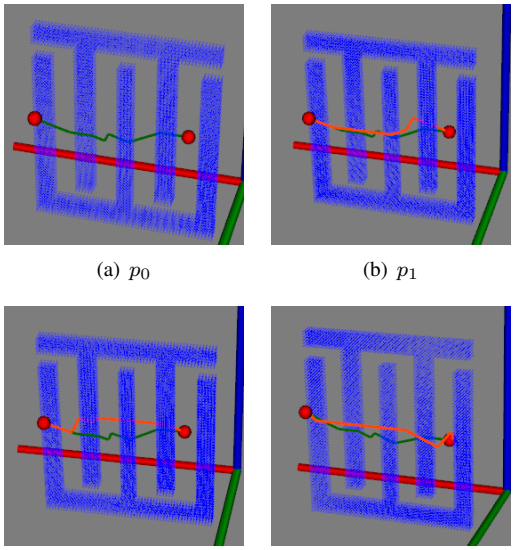
Fig. 9. Results of test 3: Two parallel objects (1 homotopy class)



(a) p_0 (b) p_1

(c) p_2

Fig. 10. Results of test 7: Two obstacles (1 homotopy class)



(a) p_0 (b) p_1

(c) p_2 (d) p_3

Fig. 11. Results of test 8: Complex obstacle with 1 homotopy class

[5] Edsger Wybe Dijkstra. A Note on Two Problems in Connexion with Graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[6] Roland Geraerts and Mark H. Overmars. Creating High-quality Paths for Motion Planning. *International Journal of Robotics Research*, 26(8):845–863, August 2007.

[7] Colin Green and Alonzo Kelly. Toward Optimal Sampling in the Space of Paths. In *13th International Symposium of Robotics Research*, November 2007.

[8] Dima Grigoriev and Anatol Slissenko. Polytime Algorithm for the Shortest Path in a Homotopy Class Amidst Semi-Algebraic Obstacles in the Plane. In *The International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 17–24, 1998.

[9] John Hershberger and Jack Snoeyink. Computing Minimum Length Paths of a Given Homotopy Class. *Computational Geometry: Theory and Applications*, 4:331–342, 1991.

[10] Takeo Igarashi and Mike Stilman. Homotopic Path Planning on Manifolds for Cabled Mobile Robots. In *Proceedings of the The Ninth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2010.

[11] Leonard Jaillet and Thierry Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotic Research*, 27(11-12):1175–1188, 2008.

[12] N. Katoh, T Ibaraki, , and H. Mine. An efficient algorithm for K shortest simple paths. volume 12, pages 411–427, 1982.

[13] Lydia E. Kavraki, Petr Svestka, Jean-claude Latombe, and Mark H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[14] Ross Alan Knepper, Siddhartha Srinivasa, and Matthew T. Mason. Toward a deeper understanding of motion alternatives via an equivalence relation on local paths. *International Journal of Robotics Research*, 31(2):168–187, February 2012.

[15] Bert Mendelson. *Introduction to Topology*. Dover, third edition, 1968.

[16] Dennis Nieuwenhuisen and Mark H. Overmars. Useful Cycles in Probabilistic Roadmap Graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 446–452, 2004.

[17] Ben Roberts and Dirk P. Kroese. Estimating the number of s-t paths in a graph. *Journal of Graph Algorithms and Applications*, 11(1):195–214, 2007.

[18] E. Schmitzberger, J.L. Bouchet, M. Dufaut, M. Wolf, and R. Husson. Capture of Homotopy Classes with Probabilistic Roadmap. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[19] Thierry Siméon, Jean-Paul Laumond, and Carole Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[20] Xiaolin Wu. An Efficient Antialiasing Technique. *SIGGRAPH Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 25(4):143–152, July 1991.