A Bug-based Path Planner Guided with Homotopy Classes

Emili Hernández, Marc Carreras and Pere Ridao
Department of Computer Engineering, University of Girona, 17071 Girona, Spain

Keywords: Path Planning, Homotopy Classes, Robotics.

Abstract: This paper proposes a bug-based path planning algorithm guided topologically with homotopy classes. Homotopy classes provide a topological description of how paths avoid obstacles in the workspace. They are generated with a method we developed, which builds a topological environment based on the workspace that allows to compute homotopy classes systematically. The homotopy classes are sorted according to a heuristic estimation of their lower bound. Only those with the smaller lower bound are used to guide the path planner we propose, called *Homotopic Bug* (HBug), which efficiently computes paths in the workspace that accomplish homotopy classes. Results show the feasibility of our method. A comparison with well-known path planners has also been included.

1 INTRODUCTION

The work presented in this paper is focused on the design of a navigation system for a robot that is able to detect the environment, build an internal local map of the explored area according to the information provided by the onboard sensors, compute a safe path through the map and, finally, generate the high-level commands to follow it. The navigation system has to be able to generate an Occupancy Grid Map (OGM) and a path according to the information obtained from the unknown environment in a reduced amount of time. The target application of this research project is autonomous navigation for surveilling and exploration tasks in unstructured environments.

Given an OGM, this paper addresses the design of a path planning algorithm to generate a path in a very short time. Anytime path planners (Ferguson et al., 2005) have been shown suitable to be used with robots that have a limited amount of time to perform path planning. These algorithms compute an initial solution highly suboptimal very fast and improves it until time runs out. Deterministic anytime approaches, like the Anytime Repairing A* (ARA*) (Likhachev et al., 2004), speed up the path generation by inflating the heuristics to force the exploration of those configuration that are closer to the goal according to their heuristics. During successive iterations the inflation factor is decreased in order to generate improved solutions. On the other side, sampling-based anytime algorithms such as Anytime-RRT (ARRT) (Ferguson and Stentz, 2006), generate a series of RRTs where each new tree reuses the cost information from the previous tree to control its growth and thus improve the quality of the resultant path. At each iteration, anytime path planners try to improve the solution by decreasing the heuristics/cost inflation. However, the generation of a new/better path is not ensured (a path with the same cost can be obtained).

Topological approaches are another way to tackle the path planning problem. This kind of solutions work with a graph-based abstraction of the workspace, in which the environment is represented by a reduced number of potential states over the aforementioned strategies (Dudek et al., 1991; Fabrizi and Saffiotti, 2000). Visibility graphs (Latombe, 1991) and voronoi diagrams (Takahashi and Schilling, 1989) constitute well-known approaches in this regard. Other methods use homotopy classes to provide a topological description of how paths avoid obstacles. Two paths that share the start point and the end point belong to the same *homotopy class* if one can be deformed into the other without encroaching any obstacle. Some authors compute the shortest homotopic path of a given input path (Chazelle, 1982; Grigoriev and Slissenko, 1998; Efrat et al., 2002; Bespamyatnikh, 2003). However, in most of robotics applications the path is not known in advance. (Fujita et al., 2003; Bhattacharya et al., 2010) generate an initial path, whose homotopy class is then obtained in order to prevent the algorithm to generate another path with the same topology. This methodology achieves the generation of a path for different homotopy classes by blocking those previously explored, which make them suitable for computing the shortest path, but not for generating a solution with a different homotopy, since it would require to compute several paths to achieve the desired solution. To overcome this problem, some methods (Jenkins, 1991; Schmitzberger et al., 2002) first generate a set of homotopy classes, allowing to look directly for a path with a specific topology. However, they require establishing a set of restriction criteria in order to generate only those homotopy classes that have an interest for the problem we need to solve. A solution to this problem has been has been proposed in (Hernández et al., 2011a; Hernández et al., 2011c).

In this paper we propose a bug-based path planner called *Homotopic Bug* (HBug), which is guided topologically with homotopy classes. Although bug algorithms were initially conceived to achieve reactive online navigation for robots with low computational capabilities in unknown scenarios, it is also possible to use them to perform deliberative path planning on a Configuration Space (C-space) (Antich et al., 2009). The HBug follows the homotopy classes generated with a method we presented in (Hernández et al., 2011a) and improved in (Hernández et al., 2011c). Using the topological information, the path planner does not have to explore the whole space but the space confined in a homotopy class, speeding up the path computation. Moreover, homotopy classes allow the generation of paths that avoid obstacles in different manners, which is interesting for surveilling purposes. Unlike anytime approaches, which start the path search with a highly suboptimal solution, the HBug starts looking for a path in a homotopy class that has a high probability of containing the optimal solution. The method is proved to be complete because in case the goal is not reachable, no homotopy classes will exist and, consequently, no paths will be generated. It is important to note that the homotopy class of the global optimal path is guaranteed to be generated by the algorithm. Results and comparison with other path planners show the efficiency and the performance of our method, achieving near to optimal solutions with respect to the homotopy class in unstructured environments with irregular obstacles, which are the target scenarios of our method.

The paper is structured as follows. Section 2 summarizes the method to generate homotopy classes from the workspace. Section 3 describes the topologically-guided path planning algorithm that we propose. Section 4 reports the results, and section 5 exposes the conclusions and future work.

2 HOMOTOPY CLASSES OF THE WORKSPACE

Two paths that share the start point and the end point belong to the same *homotopy class* if one can be deformed into the other without encroaching any obstacle. Therefore, homotopy classes provide topological information of how paths that belong to a class would avoid the obstacles. In (Hernández et al., 2011c) we presented a method to compute all the homotopy classes of any 2D workspace, whose steps are summarized in the following sections.

2.1 Reference Frame

The reference frame determines in the metric space the topological relationships between obstacles and it is used to name the homotopy classes. To build it, every obstacle is represented by a single point b_k. Then, a set of lines join each b_k point and a cental point c placed in the free space. The lines are partitioned into segments according the intersections with the obstacles and labeled α_{k_s} or β_{k_s} depending on the side of the obstacle they rely on, where k is the obstacle index and s is the segment index within the line.

The reference frame allows to represent any path as the sequence of labels of the segments being crossed in order from the starting to the ending point. For instance, Figure 1a depicts a reference frame for a scenario with two obstacles. The path that traverses it is labeled $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$. Notice that this path is topologically equivalent to $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$, which is the *canonical sequence* of the homotopy class, since is the simplest representation of a path without changing its topology (Hernández et al., 2011c).

2.2 Topological Graph

The topological graph, whose construction is based on the reference frame, provides a model to describe the topological relationships between regions of the metric space. The reference frame divides the workspace into several regions. Each region represents a node of the topological graph. The nodes are interconnected according to the number of segments they share in the reference frame. Each edge of the graph is labeled with the same label of the segment that crosses in the reference frame.

In the reference frame, a path is defined according to the segments it crosses whereas in the topological graph it turns into traversing the graph from the start-

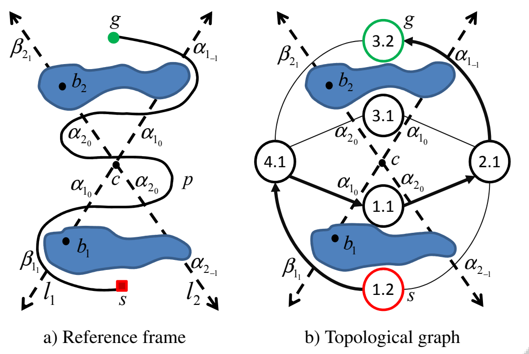


Figure 1: Topological path represented in the reference frame as $p = \beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$ and its canonical sequence ($\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$) in the topological graph.

ing node to the ending node[1]. Figure 1a depicts a path in the reference frame and Figure 1b its equivalent description in the topological graph.

2.3 Generation of Homotopy Classes

Once the topological graph is constructed, it is traversed with a modified version of the Breadth-First Search (BFS) algorithm used in (Hernández et al., 2011a; Hernández et al., 2011c), which incrementally builds the candidate homotopy classes according to the edges traversed during the graph search. The algorithm stops when there are no more homotopy class candidates to explore or the length of the last homotopy class candidate is larger than a given threshold. Notice that those homotopy classes which either self-intersect or whose canonical sequence is duplicated are not considered (Hernández et al., 2011b).

2.4 Lower Bound

Depending on the number of homotopy classes generated by the BFS algorithm, it is not possible to compute all their correspondent paths in the workspace in real-time. Therefore, we have modified *the funnel algorithm* (Chazelle, 1982) to obtain a quantitative measure for each homotopy class estimating their quality. This algorithm computes the shortest path within a *channel*, which is a polygon formed by the vertexes of the segments of the reference frame that are traversed in the topological graph. The modification consists of accumulating the Euclidean distance between the points while they are being added to the shortest path. Hence, the result of the funnel algorithm is a lower bound of the optimal path in the workspace of the selected homotopy class, which is used to set up a pref-

[1]Starting and ending nodes are those regions in the reference frame where the starting and ending points are located.

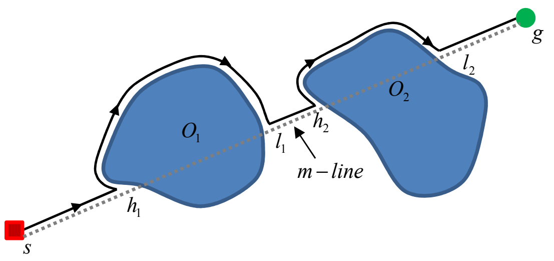


Figure 2: Example of execution of Bug2 algorithm in a simple scenario.

erence order to compute the homotopy classes path in the workspace. Figure 3a depicts an example where the funnel algorithm computes the lower bound for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. The solid lines represents the channel and the dashed line is the path after applying the algorithm. Notice that our variant of the funnel algorithm takes into account that some subsegments may self-intersect when creating the channel (α_{1_0} and α_{2_0} in Figure 3a).

3 BUG-BASED PATH PLANNING

Once the homotopy classes are computed and sorted according to their lower bound, a path planning algorithm has to find a path in the workspace that follows a given homotopy class, which essentially implies turning a topological path into a metric path. The only link between the workspace and the topological space is the reference frame. It allows checking whether a metric path in the workspace is following a topological path by following the intersections –in order– from the initial configuration to the current configuration.

3.1 The Bug2 Algorithm

Our proposal is a path planning algorithm based on the Bug2 (Lumelsky and Stepanov, 1987). The Bug2 starts by setting up an imaginary line, called *m-line*, which connects the start point with the goal point. The robot starts by following the *m-line* until the contact point with an obstacle, which is the first *hit point* h_1. Then, a boundary following behavior surrounds the obstacle until it reaches a point in the *m-line* which is closer to the goal than the previous *hit point*, that is the first *leave point* l_1. The same process is repeated until the goal is found. Figure 2 depicts a path generated by the Bug2 algorithm. Note that the direction to circumnavigate the obstacles can be fixed at the beginning of the execution or chosen randomly.

3.2 Homotopic Bug

As stated before, the path planning algorithm we propose, which is called Homotopic Bug (HBug), is based on the Bug2. Essentially, it tries to follow directly the lower bound path obtained with the modified funnel algorithm, which ensures that the homotopy class is being accomplished. However, the segments of the reference frame constrain the region where the paths can go through, but do not take into account the shape of the obstacles. For this reason, the lower bound path may intersect with the obstacles. In such case, the obstacle boundary is followed in clockwise or counterclockwise direction according to the homotopy class until the lower bound path leaves the obstacle. This process is repeated for all the intersected obstacles by the lower bound path.

The HBug, detailed in Algorithm 1, receives as an input parameters the lower bound path P, a candidate homotopy class to follow H and the reference frame F. Notice that the first and the last elements of P are the start (s) and goal (g) nodes respectively. The algorithm is a three step process. First, the function *BoundaryNodes* checks the intersections of P with the obstacles in the C-space. Every time that P hits or leaves an obstacle, a boundary node is created. Each node contains the contact point c and the obstacle label k, which is the subindex of the point b_k that represents the obstacle in the reference frame. These parameters are accessible through the functions Q and *Obst* respectively. Then, *ObstacleNodes* computes the nodes O based on the boundary nodes N previously computed. Each obstacle node contains the first boundary node that hits the obstacle n_h, the last node that is in its boundary without changing the obstacle n_l, and the direction d to surround the obstacle while following H (line 23). Finally, the function *BuildPath* creates the path P' in the workspace by joining the boundary of each obstacle $o_i \in O$ from n_h to n_l with the direction d.

The direction d to surround an obstacle is set according to the direction of a hit node n_h towards its successor n_{h+1}[2] respect to the point b_k, that represents the obstacle of the workspace in the reference frame. Notice that n_h and n_{h+1} are ensured to belong to the same obstacle since for any point that hits an obstacle there has to be another one that releases it. The perpendicular dot product between vectors $(Q(n_h) - b_k)$ and $(Q(n_{h+1}) - b_k)$ computes the boundary following direction (line 15). If the result is less than 0 the direction from n_h to n_{h+1} is counterclockwise; if it is greater than 0 the direction is clockwise.

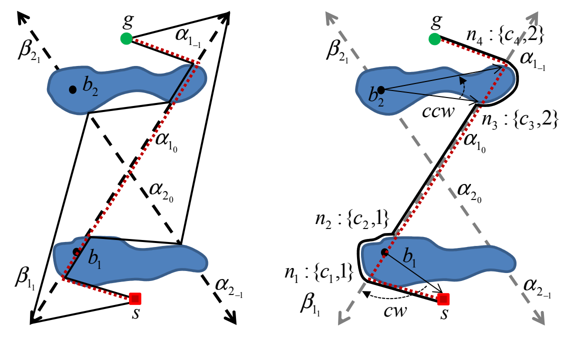
[2]When the lower bound path intersect with obstacle just one time, the n_{h+1} node is also the n_l node.

Algorithm 1: Homotopic Bug.

```
BoundaryNodes($P$)
 1: $N \leftarrow \emptyset$
 2: for $i \leftarrow 1$ to $|P| - 1, p_i \in P$ do
 3:     $C \leftarrow$ ContourPoints($p_i$, $p_{i+1}$)
 4:     for all $j \leftarrow 1$ to $|C|, c_j \in C$ do
 5:         $k \leftarrow$ Label($c_j$)
 6:         $N \leftarrow N \cup \{c_j, k\}$
 7:     end for
 8: end for
 9: return $N$
ObstacleNodes($N, H, F$)
10: $O \leftarrow \emptyset$
11: $h \leftarrow 1$
12: while $n_h \in N / h < |N| - 1$ do
13:     $n_l \leftarrow$ last $n_j \in N / j > h$ without changing $Obst(n_h)$
14:     $b_k \leftarrow$ point of $Obst(n_h)$ in $F$
15:     $d \leftarrow (Q(n_h) - b_k)^{\perp} \cdot (Q(n_{h+1}) - b_k)$
16:     if $d = 0$ then {parallel}
17:         $d \leftarrow (s - b_k)^{\perp} \cdot$ (point of $1^{st}$ $\chi_k \in H - b_k$)
18:         $i_k \leftarrow$ index of $\chi_k \in H$ where $n_{h+1}$ relies on
19:         if $|\chi_k| \in H_{1..i_k}$ is even then
20:             switch $d$
21:         end if
22:     end if
23:     $O \leftarrow O \cup \{n_h, n_l, d\}$
24:     $h \leftarrow l + 1$
25: end while
26: return $O$
BuildPath($O$)
27: $P' \leftarrow \emptyset$
28: for $i \leftarrow 1$ to $|O|, o_i \in O$ do
29:     $P' \leftarrow P' \cup$ Boundary($o_i$)
30: end for
31: return $P'$
HBug($P, H, F$)
32: $N \leftarrow$ BoundaryNodes($P$)
33: $O \leftarrow$ ObstacleNodes($N$, $H$, $F$)
34: $P' \leftarrow$ BuildPath($O$)
```

The result of the perpendicular dot product can be 0 if the vectors $(Q(n_h) - b_k)$ and $(Q(n_{h+1}) - b_k)$ are parallel, which means that n_h, n_{h+1} and b_k belong to the same l_k in the reference frame (line 16). In such case, d is obtained according two conditions: the initial direction selected to cross l_k from the start point, and the number of times that l_k is crossed until the α_k or β_k –denoted by χ_k– of the homotopy class, where n_{h+1} relies on, is reached. The initial direction is obtained with the dot product form the start s to the first χ_k with the same subindex than l_k[3] (line 17). The number of times that l_k is crossed depends on the number of χ_k found in the homotopy class from the beginning to the index i_k, which indicates the position

[3]Notice that the start point cannot be in a line l_k of the reference frame since the perpendicular dot product would be also 0.



a) Channel and lower bound path b) Path computed with the HBug

Figure 3: Generation of the lower bound path and the HBug path for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$.

of the χ_k that contains n_{h+1} (line 19).

Figure 3 depicts an example scenario where the HBug is applied. The homotopy class to follow is $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. The dashed line represents its lower bound path, which intersects with the first obstacle generating two boundary nodes n_1 and n_2, both of them located on the line l_1 of the reference frame. The point that represents the obstacle is b_1, also on l_1, which makes perpendicular dot product between $(Q(n_1) - b_1)$ and $(Q(n_2) - b_1)$ unable to set the direction ($d = 0$). Therefore, using the start point s and a point of the edge β_{1_1}, the initial direction is set clockwise (*cw*). The last edge involved in this situation is α_{1_0}, which is located in the second position in the homotopy class. The number of edges with subindex 1 till this position is 2. Thus, the direction is not changed. Then, the lower bound path intersects with the second obstacle in n_3 and n_4. Using the base point b_2, the perpendicular dot product sets the direction as counterclockwise (*ccw*). Finally, the path is composed from s to g with the boundaries of the obstacle 1 (from n_1 to n_2) and the obstacle 2 (from n_3 to n_4) joint by straight lines.

4 RESULTS

The feasibility the topological path search with the HBug has been tested in different bitmap scenarios which have been used as C-spaces, where the robot is represented as a single point. To identify the obstacles of the scenarios we have adapted a Component-Labeling (CL) algorithm that efficiently labels connected cells and their contours in greyscale images at the same time (Chang et al., 2004). The contours of the obstacles provided by this algorithm are used by the HBug to avoid recomputing them at each new path search. For the construction of the reference frame, the c point has been set at a fixed position in

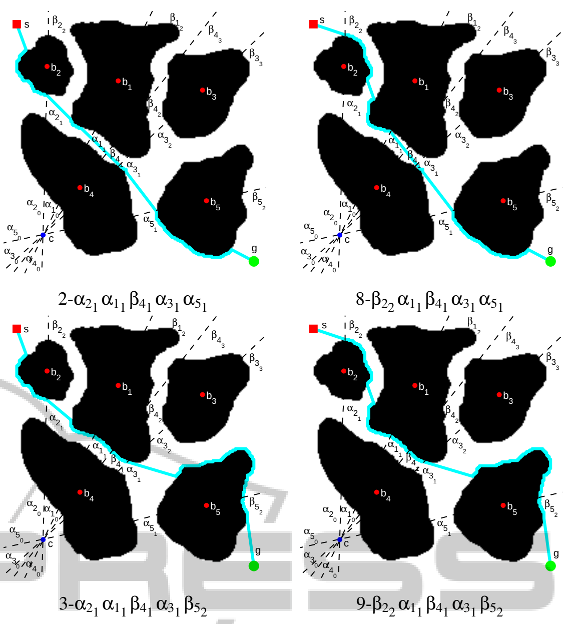

2-$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$
8-$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$
3-$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$
9-$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$

Figure 4: Paths for the four homotopy classes with the smaller lower bound in Table 1.

order to ensure the same topological graph construction –and homotopy classes generation– through different executions. The homotopy classes have been set at a maximum of 20 characters length. In order to show all the possible results, no time restrictions have been taken into consideration. The computations have been carried out with a laptop equipped with an Intel® Core^{TM} Duo@1.83GHz processor and 2GB of RAM.

4.1 Cluttered Environment

Our method has been applied in a cluttered environment using a 200x200 pixels bitmap. Figure 4 depicts the scenario with the paths of the best homotopy classes, according to their lower bound. The construction of the reference frame, the topological graph and the generation of the homotopy classes with their lower bound computation took 7.9ms. Table 1 shows the homotopy classes sorted by their lower bound with the path cost and the accumulated computation time, which takes into account the homotopy classes computation and the path generation. The lower bound and the path cost have been normalized with the cost of the optimal path computed with the A* algorithm. It can be appreciated that using the HBug the computation time for the paths of the 13 homotopy classes is almost negligible (0.374ms) compared with the 7.9ms that takes the rest of the process.

When operating under time restrictions, it is possible to stop the path search when the lower bound

Table 1: Homotopy classes of Figure 4 environment sorted by their lower bound.

Idx	Homotopy class	Lower bound	Cost	Cumulated time (*ms*)
2	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.93	1.16	7.927
8	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.94	1.20	7.950
3	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	0.99	1.43	7.981
9	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	1.00	1.45	8.007
1	$\alpha_{5_0}\alpha_{3_0}\alpha_{4_0}\alpha_{1_0}\alpha_{2_0}$	1.04	1.48	8.035
11	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	1.08	1.53	8.063
10	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.13	1.51	8.084
13	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.18	1.31	8.106
5	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	1.26	2.02	8.142
4	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.32	1.99	8.174
7	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.36	1.80	8.205
12	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.40	1.77	8.236
6	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.58	2.26	8.274

Table 2: The five homotopy classes of the large environment with the smaller lower bound and their generation index.

Idx	Homotopy class
25	$\alpha_{15_{-1}}\alpha_{9_0}\alpha_{12_0}\alpha_{1_0}\alpha_{4_0}\alpha_{6_0}\alpha_{5_0}\alpha_{8_1}\alpha_{3_{-1}}\alpha_{11_2}\alpha_{13_2}\beta_{7_2}\alpha_{2_{-2}}\alpha_{14_2}\beta_{10_2}$
26	$\alpha_{15_{-1}}\alpha_{9_0}\alpha_{12_0}\alpha_{1_0}\alpha_{4_0}\alpha_{6_0}\alpha_{5_0}\alpha_{8_1}\alpha_{3_{-1}}\alpha_{11_2}\alpha_{13_2}\beta_{7_2}\alpha_{2_{-2}}\beta_{14_3}\beta_{10_2}$
5	$\alpha_{10_{-1}}\alpha_{14_{-1}}\beta_{2_1}\alpha_{7_{-1}}\alpha_{13_{-1}}\alpha_{11_{-1}}\alpha_{3_0}\alpha_{8_0}\beta_{5_1}\alpha_{6_1}\alpha_{4_{-1}}\alpha_{1_{-2}}\alpha_{12_2}\dots$ $\dots\beta_{9_2}\alpha_{15_2}$
1	$\alpha_{10_{-1}}\alpha_{14_{-1}}\beta_{2_1}\alpha_{7_{-1}}\alpha_{13_{-1}}\alpha_{11_{-1}}\alpha_{3_0}\alpha_{8_0}\alpha_{5_0}\alpha_{6_0}\alpha_{4_0}\alpha_{1_{-2}}\alpha_{12_2}\dots$ $\dots\beta_{9_2}\alpha_{15_2}$
41	$\alpha_{15_{-1}}\alpha_{9_0}\alpha_{12_0}\alpha_{1_0}\beta_{4_1}\alpha_{6_{-1}}\alpha_{5_{-1}}\alpha_{8_1}\alpha_{3_{-1}}\alpha_{11_2}\alpha_{13_2}\beta_{7_2}\alpha_{2_{-2}}\dots$ $\dots\alpha_{14_2}\beta_{10_2}$

of the next homotopy class –whose path is going to be computed– is higher than the minimum cost of the paths already computed. In such case, it is ensured that the best path with the HBug has been already computed because it is not possible to obtain a path with a lower cost than the lower bound. For instance, in Table 1 the HBug would stop before computing path for homotopy class with index 13 since its lower bound (1.18) is higher than the path length obtained with index 2 class (1.16).

4.2 Scalability

The scalability of the method has been tested in a bitmap of 1000x1000 pixels with 15 irregular obstacles (Figure 5). The construction of the reference frame, the topological graph and the generation of 112 homotopy classes with their lower bound computation took 0.304*s*. The HBug has been used to compute the path for all homotopy classes of the scenario. The homotopy classes with the smaller lower bound, shown in Table 2, and their paths computed with the HBug are depicted in Figure 5. Figure 6 shows the cost and computation time for each homotopy class path computed with the HBug. The homotopy classes have been sorted according to their lower bound. The path cost and the lower bound have been normalized

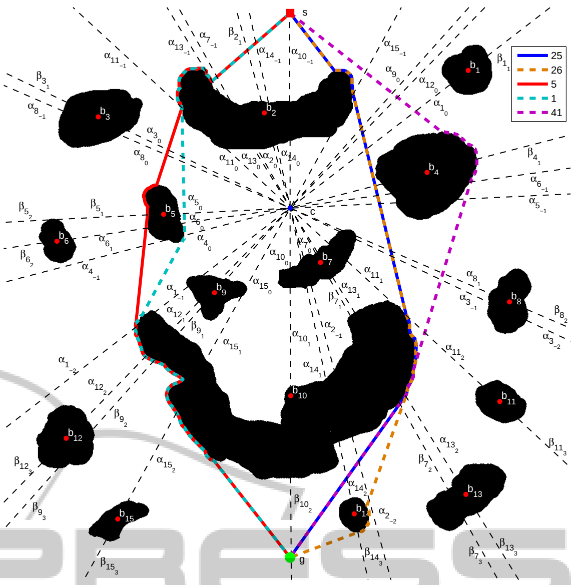


Figure 5: Paths of the five homotopy classes with the smaller lower bound computed with the HBug. The class associated to the index can be found in Table 2.

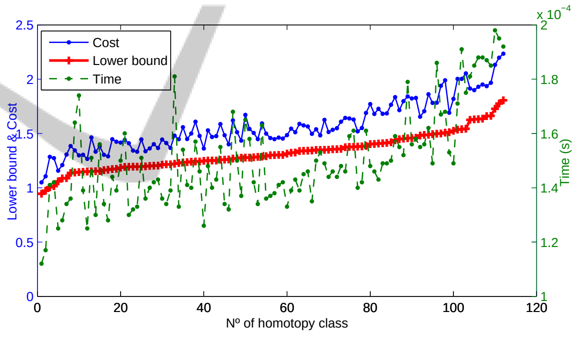


Figure 6: Normalized cost, normalized lower bound and computation time for paths generated with the HBug for each homotopy class.

with respect to the A* path cost. If time restrictions were applied, the HBug would compute only till the fourth path.

The fastest path was generated in $1.12\text{x}10^{-4}s$. It corresponds to the first class (index 25) with a path cost only 1.05 times the optimal solution. The homotopy class 110 (index 102) was the one that took more time to be computed ($1.98\text{x}10^{-4}s$) with a cost 2.24 times the global optimal solution. The mean computation per path was only ($1.50\text{x}10^{-4}s$). In this environment, the paths for the whole set of homotopy classes were computed in 16.8*ms*, which is almost negligible when compared with the 304*ms* spent in the generation of the reference frame, topological graph and the generation of the homotopy classes with their lower bound.

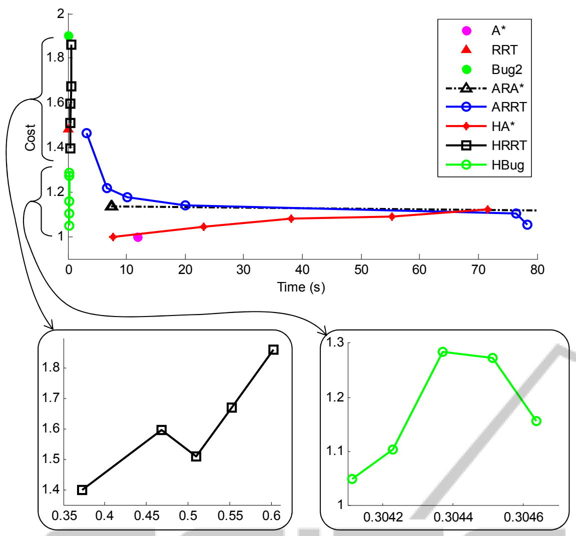


Figure 7: Comparison of the HA*, HRRT and HBug paths of the five homotopy classes with the smaller lower bound vs A*, RRT, ARA*, ARRT and Bug2 algorithms.

4.3 Path Planners Comparison

The HBug has been compared against the A*, the RRT, their respective anytime versions (ARA* and ARRT) and the Bug2 algorithm. The comparison also includes the results obtained with the Homotopic A* (HA*) (Hernández et al., 2011b) and the Homotopic RRT (HRRT) (Hernández et al., 2011a), two path planners that constrain the path search according to the homotopy classes previously generated. The A*, the RRT and the Bug2 algorithms are designed to compute only one path towards the goal; ARA* and ARRT compute several paths but without taking into account their homotopy. Because of that, the comparison with these well-known path planners are against the five homotopy classes with the smaller lower bound (see Table 2). Figure 7 depicts the comparison. In order to ensure the stabilization of the results of the probabilistic path planners, the data obtained with the RRT and the ARRT are the average of 100 executions. Notice that all the time values of the HA*, HRRT and HBug include the computation time of the reference frame and topological graph construction, and the generation of the homotopy classes with their lower bound.

The A* returned the optimal path in 11.90s. The ARA* generated the first solution in 7.40s and found the optimal solution after 301s. The RRT algorithm took 0.012s to compute a path with a cost 1.48 times the global optimal solution. The ARRT took 3.13s to obtain the first solution and 78.30s to compute all of them, ensuring that any new generated solution is closer to the optimal one. The Bug2 algorithm computed the path in 0.044s with a cost 1.90 times the optimal solution. In order to obtain the best possible path with this algorithm, we have chosen manually the directions to surround the obstacles: the m-line, which connects the start with the goal, intersects with the obstacles labeled b_1, b_7 and b_{10}; the directions are clockwise for b_1, counter-clockwise for b_7 and clockwise for b_{10}.

The HA* computed the optimal path (index 25) in 7.79s and obtained the path for the five selected homotopy classes in 71.61s. The HRRT, the best solution (index 25) was computed in 0.373s with a cost 1.40 times the optimal one, and obtained the path for the five homotopy classes with the smaller lower bound in 0.603s. Using the HBug, our path planning method computed the best solution (index 25) in 0.304s with a cost 1.05 times the optimal one, and obtained the path for the whole set of homotopy classes in 0.321s. The computation of the paths with the HBug for each homotopy class offers a very good performance. Only the RRT and Bug2 had lower computation times at expenses of finding higher cost solutions. Notice that most of the time was spent in the computation of the reference frame, the topological graph and the generation of the homotopy classes with their lower bound. The HBug shows a very good performance because part of its solution is implicitly generated on the lower bound computation. When the lower bound path intersects with an obstacle of the scenario, its boundary is followed in clockwise or counterclockwise direction according to the homotopy class until the lower bound path leaves the obstacle. Notice that the whole boundary of each obstacle is already computed by the CL algorithm.

The HA* computes the shortest path for each homotopy class, which is the optimal solution according to the topological constraints. Because of that, Figure 8 depicts a comparison of the solutions generated with the HRRT and the HBug against the HA* cost for each specific homotopy class. The HRRT generates solutions between 1.35 (class 15, index 73) and 1.82 (class 83, index 61), with a mean of 1.57 times with respect the optimal path cost for the specific homotopy class generated with the HA*. Finally, the HBug generates solutions between 1.03 (class 5, index 41) and 1.19 (class 100, index 101), with a mean of 1.1 times with respect the optimal path cost computed with the HA*.

5 CONCLUSIONS

This paper proposes the HBug path planning algorithm to compute efficiently paths that accomplish ho-

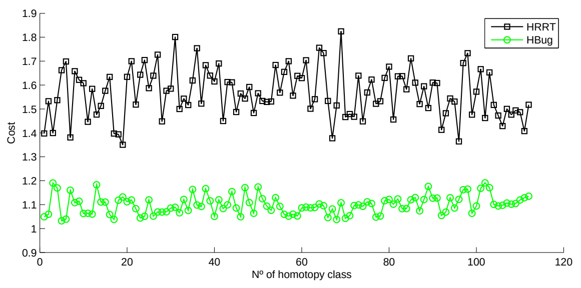


Figure 8: HRRT and HBug paths cost with respect to the HA* cost for each homotopy class.

motopy classes in any 2D workspace. Given a map with obstacles, we use a method we developed to generate systematically the homotopy classes of the environment. After sorting them according to a lower bound, the HBug algorithm generates paths in the C-space following the homotopy classes previously found. The path planner offers very good performance since the path search for a homotopy class is guided by its lower bound, making the path planning computation time almost negligible when compared with the time used to generate the homotopy classes. Results obtained with the HBug, have shown up that it is a homotopic path planner suitable for robots with very limited computational capabilities or applications in which the time to perform path planning is highly constrained.

Future work will consists in applying our method into one of the vehicles of our lab. The HBug will be improved by taking into account the robot's kinodynamic constraints during the path generation. These paths will be used to guide the robot autonomously.

ACKNOWLEDGEMENTS

The authors of this paper gratefully acknowledge the support from Spanish government under the grant DPI2011-27977-C03-02 and the TRIDENT EU FP7-Project under the grant agreement No: ICT-248497.

REFERENCES

Antich, J., Ortiz, A., and Minguez, J. (2009). A bug-inspired algorithm for efficient anytime path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5407 –5413.

Bespamyatnikh, S. (2003). Computing homotopic shortest paths in the plane. *Journal of Algorithms*, 49:284–303.

Bhattacharya, S., Kumar, V., and Likhachev., M. (2010). Search-based path planning with homotopy class constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1230–1237, Atlanta, Georgia, USA.

Chang, F., jen Chen, C., and Lu, C.-J. (2004). A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220.

Chazelle, B. (1982). A theorem on polygon cutting with applications. In *23rd Annual Symposium on Foundations of Computer Science (SFCS)*, pages 339 –349.

Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1991). Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865.

Efrat, A., Kobourov, S., and Lubiw, A. (2002). Computing homotopic shortest paths efficiently. In Möhring, R. and Raman, R., editors, *Algorithms – ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 277–288. Springer Berlin / Heidelberg.

Fabrizi, E. and Saffiotti, A. (2000). Extracting topology-based maps from gridmaps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2972–2978.

Ferguson, D., Likhachev, M., and Stentz, A. (2005). A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*.

Ferguson, D. and Stentz, A. (2006). Anytime RRTs. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5369 – 5375.

Fujita, Y., Nakamura, Y., and Shiller, Z. (2003). Dual dijkstra search for paths with different topologies. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3359–3364.

Grigoriev, D. and Slissenko, A. (1998). Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 17–24, New York, NY, USA. ACM.

Hernández, E., Carreras, M., Antich, J., Ridao, P., and A.Ortiz (2011a). A topologically guided path planner for an AUV using homotopy classes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2337–2343, Shanghai, China.

Hernández, E., Carreras, M., Antich, J., Ridao, P., and Ortiz, A. (2011b). A search-based path planning algorithm with topological constraints. Application to an AUV. In *Proceedings of the 18th IFAC World Congress*, Milan, Italy.

Hernández, E., Carreras, M., and Ridao, P. (2011c). A path planning algorithm for an AUV guided with homotopy classes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany.

Jenkins, K. D. (1991). The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes.

Master's thesis, Naval Postgraduate School, Monterey, California.

Latombe, J. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA.

Likhachev, M., Gordon, G., and Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Advances in Neural Inforamtion Processing Systems 16 (NIPS)*. MIT Press.

Lumelsky, V. and Stepanov, A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430.

Schmitzberger, E., Bouchet, J., Dufaut, M., Wolf, D., and Husson, R. (2002). Capture of homotopy classes with probabilistic road map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2317–2322.

Takahashi, O. and Schilling, R. (1989). Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143 – 150.