A Path Planning Algorithm for an AUV Guided with Homotopy Classes

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

Abstract

The paper proposes a method that uses topological information to guide path planning in any 2D workspace. Our method builds a topological environment based on the workspace to compute homotopy classes, which topologically describe how paths go through the obstacles in the workspace. Then, the homotopy classes are sorted according to an heuristic estimation of their lower bound. Only those with smaller lower bound are used to guide a planner based on the Rapidly-exploring Random Tree (RRT), called *Homotopic RRT* (HRRT), to compute the path in the workspace. Simulated and real results with an Autonomous Underwater Vehicle (AUV) are presented showing the feasibility of the proposal. Comparison with well-known path planning algorithms has also been included.

Introduction

The main goal of this research project is to design a motion system to safely and deliberatively guide and Autonomous Underwater Vehicle (AUV) towards a goal waypoint. The vehicle must be able to sense the environment to build a local map of the robot surroundings to compute a safe path towards the goal. Once a path has been found, it is assumed to be reached by a path following algorithm. To achieve real-time requirements, the motion system has to be able to update the map and the path according to the information obtained from the unknown environment in a reduced amount of time (less than 10-15s). Although path planning for AUVs is naturally formulated in 3D, for certain scenarios of interest the problem can be simplified to 2D. For instance, let us consider a survey and/or search mission where the robot is supposed to fly at a fixed altitude, in bottom-following mode, while acquiring opto-acoustic imagery. Under these conditions, we can consider a 2D map parallel to the seafloor, where any area with a slope greater than a certain threshold behaves as a 2D obstacle. This is the case for applications like benthic habitat mapping, underwater archeology or cable/pipe inspection, being also the target for the system proposed in this paper.

This paper addresses the design of the path planning algorithm to generate a path in the local map in a very short time. Our approach assumes that the local map is constructed just for navigation purposes. This kind of application requires fast path planning. Therefore, anytime planners (Ferguson, Likhachev, and Stentz 2005) are the most suitable ones: they find a first solution, possibly highly suboptimal, very quickly and then they refine it until time runs out. Usually, the anytime path planners require a multiplication factor ϵ to control the cost of the generated solution and a decrement factor to decrease ϵ at each iteration until $\epsilon = 1$ if time does not expire before (Likhachev et al. 2005; Ferguson and Stentz 2007). For example, the Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2004) inflates the heuristic function using the ϵ value to guide the search towards those states which are close to the solution whose final cost is ensured not to be more than ϵ times the cost of the optimal path. Although at each iteration the solution is intended to be improved by decreasing ϵ, the generation of a new/better path is not ensured (the same path as in the previous iteration of the algorithm can be obtained), which means a waste of computation time in a critical context since the available time to perform the path planning is very limited.

Recently, (Ferguson and Stentz 2006) developed a probabilistic anytime algorithm called Anytime-RRT (ARRT). It is a sampling-based planner that works by generating 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. Unlike the ARA*, the ARRT scarcely reutilizes data between iterations because each new tree is almost built from scratch.

Using homotopy constrains is another way to tackle the path planning problem. Two paths that share the start and the end point belong to the same homotopy class if one can be deformed into the other without encroaching any obstacle. There are several approaches that deals with homotopy classes: (Jenkins 1991; Cabello et al. 2002) assume that obstacles are points without area. Grigoriev and Slissenko (Grigoriev and Slissenko 1998) propose a method to construct the shortest path for a given homotopy class in a scenario with semi-algebraic obstacles. Recently, Bhattacharya (Bhattacharya, Kumar, and Likhachev. 2010) proposed a method that assigns a *value* to represent the homotopy class during the computation of shortest path. Then, to obtain a different path, this *value* is used to constrain the search.

However, it is difficult to configure the path planner to follow a specific homotopy class without finding its path and hence, its *value*.

This paper proposes the use of homotopy classes to guide topologically a path planning algorithm. Using the topological information, the planner does not have to explore the whole space but the space confined in a homotopy class. Then, using a lower bound criterion the homotopy classes that most-probably contain the lower cost solutions are known. Thus, the algorithm can generate some good solutions very fast and, therefore, act as an anytime algorithm. Moreover, the homotopy classes allow us to reach the goal position by avoiding the obstacles in different ways, which has interest when the robot is surveilling a particular area.

The paper presents an extension of the method proposed by (Jenkins 1991) to generate homotopy classes that can be followed in any 2D workspace. Then, the homotopy classes are sorted according their quality given by a lower bound estimator. In order to maximize the number of homotopy classes that can be explored, the proposed planner, called *Homotopic RRT* (HRRT), is based on the RRT algorithm which has been shown to be very efficient in time, even in complex workspaces (Lavalle S. M. 1999; Kim and Ostrowski 2003). The HRRT algorithm does not require the setting of the ϵ nor decrement value. Instead of starting the search with a highly suboptimal path that is improved over time, it starts looking for a path in the 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. On the other hand, it is important to note that the homotopy class of the global optimal path is guaranteed to be generated by the algorithm. Finally, in order to evaluate the feasibility of the work in a motion system of an AUV, the paper presents the results obtained with an underwater robot. An underwater environment was built to allow the testing of the path planning algorithm in a water tank. The robot moved through the environment, perceiving the obstacles and generating an Occupancy Grid Map (OGM). Then, the HRRT was applied generating a path for each homotopy class on the map.

The Jenkins Method

Given a workspace with obstacles, (Jenkins 1991) proposes a method to generate a topological representation of the environment which is used to extract all the different *homotopic* paths from the starting to the ending point. Two paths are homotopic if they go trough the obstacles in the same manner.

Formally, given a region R defined by the workspace, let P be the set of all continuous, obstacle-avoiding paths from a starting point to an ending point. Then, two paths $p_i, p_j \in P$ are *homotopic* if they can be mapped to each other through a continuous function without encroaching any obstacle. The homotopy relation is reflexive, symmetric, and transitive, and forms an equivalence relation. Thus, the homotopic functions in P form equivalence classes, and these are called *homotopy classes* (Jenkins 1991).

Reference Frame

The reference frame determines, in the metric space, the topological relationships between obstacles and it is used to name the homotopy classes. The construction method proposed by (Jenkins 1991) in a scenario with n obstacles starts by choosing a random point for each obstacle, which is labeled as b_k, where $k = 1..n$. Then, a central point c is randomly selected. This point cannot be inside an obstacle nor be inside the $n(n-1)/2$ lines determined by the pairwise choices of distinct b_k. Finally, n lines l_k joining c with each b_k are constructed. Each line is partitioned into two directed semifinite rays: the ray emanating from b_k and away from c is labeled β_k and the ray from b_k that contains c is labeled α_k. Figure 1a shows an example of a reference frame in a scenario with two obstacles.

Using the reference frame, any path p can be defined by the sequence of labels of the rays being crossed in order from the starting to the ending point. For instance, the path in Figure 1a is labeled $\alpha_2\beta_1\alpha_2\alpha_2\alpha_2$. Nevertheless, there are two special cases: when p crosses no rays then $p = \emptyset$ and when p crosses through c meaning that all the α's are simultaneously crossed. In such a latter case, all α_k are added in subindex order to the sequence.

Two paths are homotopic if they have the same *canonical sequence*, which is the simplest representation of a path without changing its topology. Using Jenkins notation, it is computed by sorting the α's substrings of the path in non-decreasing order of subindex and then removing all the elements of the sequence by pairs that have the same character. This process is repeated until no changes are made to the sequence. In Figure 1 the canonical sequence of the path $\alpha_2\beta_1\alpha_2\alpha_2\alpha_2$ is $\alpha_2\beta_1\alpha_2$.

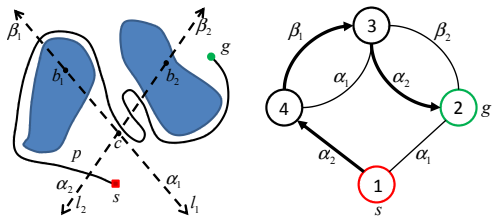
Topological Graph

The topological graph G, 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 metric space into *wedges* which are represented as nodes of G. Each node is connected with its neighbors with one or two edges depending on the segments traversed between wedges. These edges are labeled according to the segment crossed in the reference frame.

In the reference frame, a path is defined according to the segments it crosses whereas in G it turns into traversing the graph from the starting 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.

Once the topological graph is constructed, Jenkins proposes to traverse it using a modified version of the Breadth-First Search algorithm (BFS) to generate all the homotopy classes that are not self-crossing nor duplicated. For example, in Figure 1 the homotopy classes would be: α_1, $\alpha_1\alpha_2\beta_2$, $\alpha_1\beta_2\alpha_2$, $\alpha_2\beta_1\alpha_2$ (depicted), $\alpha_2\beta_1\beta_2$ and $\alpha_1\beta_2\alpha_1\beta_1\beta_2$.

[1]Starting and ending nodes are those *wedges* in the reference frame -nodes in G- where the starting and ending points are located.



a) Reference frame b) Topological graph

Figure 1: Topological path represented in the reference frame as $p = \alpha_2\beta_1\alpha_2\alpha_2\alpha_2$ and its canonical sequence $(\alpha_2\beta_1\alpha_2)$ in the topological graph.

Applicability to the Path Planning Problem

We propose to use the Jenkins method to guide a path planning algorithm following a topological path. Thus, the topological information of the homotopy classes has to be turned into metric paths in the workspace by using the reference frame as a link between the topological graph and the workspace. Therefore, in order to find a path in the workspace that follows a specific homotopy class, it is required for a path planning algorithm to be modified to look for the intersections with the desired segments of the reference frame.

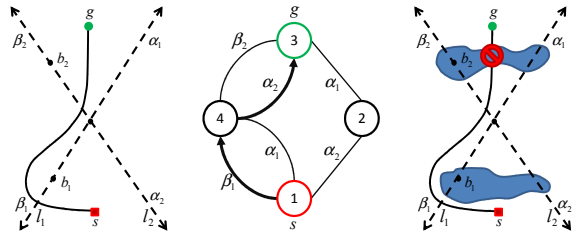
During the reference frame construction, each obstacle of the map is represented by a b_k point without area in order to ensure that each line of the reference frame only crosses one obstacle; and the topological graph is built under this assumption. However, depending on the reference frame construction and the particular shape of the obstacles it is possible that a line of the reference frame intersects with more than one obstacle. In some cases, there will be homotopy classes that cannot be followed in the workspace. Figure 2 depicts this problem: the homotopy class to follow is $\beta_1\alpha_2$, which is shown as a path in Figure 2a in the reference frame with its equivalence in the topological graph (Figure 2b). However, in Figure 2c the metric path cannot be followed because obstacle 2 is crossing l_2 and l_1. Notice that this problem would not arise if points c and b_1 were more carefully selected to avoid the intersection of the obstacle 2 with l_1. However this solution cannot be applied in complex scenarios with more obstacles.

Extension of the Jenkins Method

In order to avoid the problem found in the previous section, this paper proposes an extension of the Jenkins method which takes into account the shape of the workspace obstacles during the construction of the reference frame and the creation of the topological graph. The homotopy classes would be computed according to the intrinsic restrictions of the scenario keeping the coherence between the topological path and its metric representation in the workspace.

Reference Frame

As in Jenkins method, each obstacle k has to be represented as a single point b_k. However, it is necessary to distinguish different segments when other obstacles intersect line l_k.



a) Reference frame b) Topological graph c) Workspace

Figure 2: Example of a valid homotopy class $(\beta_1\alpha_2)$ in the reference frame (a) and in the topological graph (b) that cannot be followed in the workspace (c) because at least one line (l_1 or l_2) in the reference frame intersects more than one obstacle.

Thus, we extend the notation of the segments to describe this fact. The whole construction process can be summarized in three steps:

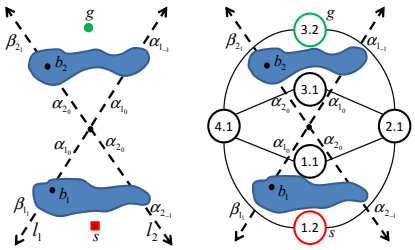
1. Select a random point for each obstacle and label it as b_k, where $k = 1..n$.
2. Select the central point c of the reference frame. This point cannot be inside an obstacle nor being inside the $n(n-1)/2$ lines determined by the pairwise choices of distinct b_k.
3. Construct n lines l_k joining c with each b_k. Each line is partitioned into $m+1$ segments, where m is the number of obstacles that intersect with l_k in the workspace. The segments from b_k and away from c are labeled with β_{k_s}, and the segments in the opposite direction are labeled α_{k_s}, where $s = 0..u$ with $u \in \mathbb{Z}^+$ for the segments of l_k from c that passes through b_k and $s = 0..v$ with $v \in \mathbb{Z}^-$ for the segments in the opposite direction.

This kind of labeling allows us to distinguish different segments created by the l_k-obstacle intersections in the workspace. Figure 3a depicts the new reference frame for the scenario in which we had problems with the original Jenkins topological description.

Topological Graph

The method to construct the topological graph G has been modified to take into account the extended reference frame. Its construction can be divided in three steps:

1. The lines of the reference frame divide the metric space into regions or *wedges* and the obstacles that intersect with more than one line at the same time split these wedges into *sub-wedges*. Each sub-wedge represents a node of G.
2. Each node of G is labeled according to the wedge w and sub-wedge sw using the notation $w.sw$. $w \in \mathbb{N}$ is numbered counterclockwise. For each w, its corresponding $sw \in \mathbb{N}$ are numbered sequentially starting by 1 for the one closest to c.
3. Two nodes of G are interconnected according to the number of segments they share in the reference frame. Each



a) Reference frame b) Topological graph

Figure 3: Example of a reference frame and its correspondent topological graph using the extension of Jenkins method.

edge of G is labeled with the same label of the segment that crosses in the reference frame.

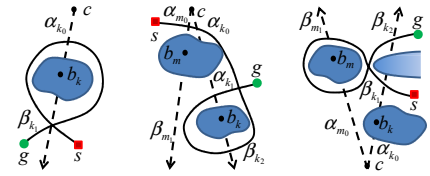
As can be seen in Figure 3b, now the topological graph takes into account the visibility between regions according to the representation of the obstacles in the reference frame, which contains the restrictions of the workspace intrinsically. Notice that any topological path of the topological graph can now be followed in the workspace. Hence, the homotopy class $\beta_1\alpha_2$, which is represented $\beta_{1_1}\alpha_{2_0}$ with the new notation, is not valid anymore to describe a path from s (node 1.2) to g (node 3.2).

Generation of Homotopy Classes

The topological graph is traversed using the version of the BFS algorithm proposed by Jenkins. Unlike the standard BFS, which stops when all vertexes of the graph have been visited, it continues until there are no more homotopy class candidates to explore or the length of the last homotopy class candidate is larger than a given threshold.

During the BFS execution, several restriction criteria are applied to avoid the generation of any homotopy class which either self-intersects or whose canonical sequence is duplicated and has already been considered. All classes that accomplish any of the following restrictions criteria are ignored to avoid using them as a root for future homotopy classes:

- Simple Wrap. Any string that contains a substring of the form $\alpha_{k_s}...\chi_{k_t}...\alpha_{k_u}$ or $\beta_{k_s}...\chi_{k_t}...\beta_{k_u}$ where $\chi = (\alpha, \beta)$ with $s = u$ represents a class that wraps around an obstacle and is self-crossing.
- Wrap. Any string that contains a substring of the form $\chi_{k_s}\cdots\chi_{k_t}\cdots\chi_{k_u}$ where $\chi = (\alpha, \beta)$ with $s, t, u \geq 0$ and $s > t < u$ or with $s, t, u \leq 0$ and $s < t > u$ represents a class that wraps around an obstacle and is self-crossing.
- Self-crossing. Any string that contains a substring of the form $\chi_{k_s}...\beta_{m_t}...\alpha_{m_u}...\chi_{k_v}$ where $\chi = (\alpha, \beta)$ with $s, v \geq 0$ and $s < v$ or with $s, v \leq 0$ and $s > v$ represents a class that self-crosses. The reversed substring $\chi_{k_s}...\alpha_{m_t}...\beta_{m_u}...\chi_{k_v}$ with $s, v \geq 0$ and $s > v$ or with $s, v \leq 0$ and $s < v$ also represents a class that self-crosses.



a) Simple wrap b) Wrap c) Self-crossing

Figure 4: Examples of the restriction criteria.

Figure 4a depicts an example of the simple wrap criterion with path $\beta_{k_1}\alpha_{k_0}\beta_{k_1}$. Figure 4b shows a wrap with the path $\alpha_{m_0}\alpha_{k_0}\beta_{k_2}\alpha_{k_1}$ and Figure 4c depicts an example of the self-crossing criterion with path $\beta_{k_1}\beta_{m_1}\alpha_{m_0}\beta_{k_2}$.

- Duplicated strings are not allowed in the list of homotopy class candidates. If a string is not in its canonical form, it can be simplified without modifying its topology. Then, it is ensured that the resultant string has been already computed by the BFS algorithm because it would be shorter than the input string. Finally, the algorithm cannot traverse through the same edge on two consecutive occasions. By doing that, a string with a repeated pair would be generated. Consequently, the pair would be simplified and the string discarded for being duplicated.

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. It is used to set up a preference order to compute the homotopy classes path in the workspace when operating under time restrictions. Notice that 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 that reason, a homotopy class with a smaller lower bound may have a longer path in the workspace than an another homotopy class with a higher lower bound.

Guided 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. Our proposal is a variation of the goal-biased RRT algorithm called *Homotopic RRT* (HRRT). It allows a constrained growing of the tree only in those directions that satisfy a given homotopy class. Before adding a new node into the tree, the topological path traversed is checked to ensure that belongs to the homotopy class by computing the intersections of the path with the reference frame.

The algorithm is detailed in Algorithm 1. It receives as input a candidate homotopy class to follow ($tPath$) and the reference frame ($refFrame$). The nodes of the tree T are tuples that contain the configuration of the robot q and the topological path from q_{start} to q. These values are accessible through the functions Q and P respectively. Just like the RRT, the function $Extend$ (line 31) iteratively extends the tree T until the distance between the configuration of n_{new} ($Q(n_{new})$) and n_{goal} ($Q(n_{goal})$) is lower than a $distThreshold$. In this function, $ComputeQRand$ selects a random configuration q_{rand} from the workspace. Then, the $NearestNeighbor$ function returns the nearest node $n_{nearest}$ regarding a random configuration q_{rand} by looking for the node whose topological path is closer to $P(n_{goal})$ (line 4). If there is more than one candidate, the node selected is the closest to the goal according to the Euclidean distance. After q_{new} is computed using the function $ComputeQNew$, $FindIntersections$ (line 21) checks whether the segment [$Q(n_{nearest})$, q_{new}] intersects with any segment of the reference frame F[2]. The function returns the intersected edges sorted by distance from $Q(n_{nearest})$. Then the function $UpdatePath$ (line 22) generates the new topological $path$ according to the intersections. No intersection with F means that the tree grows in the $n_{nearest}$ $wedge$ and hence, the function returns $P(n_{nearest})$. If there are intersections and these intersections follow the topological path, the function returns $P(n_{nearest}) \cup I$ in order to create a candidate new node n_{new} to be added to the tree; otherwise a $null$ path is returned and no node is added to the tree.

Algorithm 1 Homotopic RRT

NearestNeighbor(T, q_{rand})
1: $n \leftarrow T;\ d \leftarrow Distance(Q(n), q_{rand})$
2: **for all** $c \leftarrow T.Children()$ **do**
3: $[n', d'] \leftarrow NearestNeigbor(T, q_{rand})$
4: **if** $(|P(n')| > |P(n)|)$ **or** $(|P(n')| = |P(n)|$ **and** $d' < d)$ **then**
5: $n \leftarrow n';\ d \leftarrow d'$
6: **end if**
7: **end for**
8: **return** $\{n, d\}$

FindIntersections$([q_{nearest}, q_{new}], F)$
9: $r \leftarrow \emptyset$
10: **for** $i \leftarrow 1$ **to** $|F|$ **do**
11: **if** $p \leftarrow Intersection([q_{nearest}, q_{new}], F[i]) \neq null$ **then**
12: $r \leftarrow r \cup \{Edge(i), Distance(q_{nearest}, p)\}$
13: **end if**
14: **end for**
15: $r \leftarrow SortByDistance(r)$
16: **return** r

Extend(T, n_{goal}, F)
17: $n_{new} \leftarrow \{\infty, null\}$
18: $q_{rand} \leftarrow ComputeQRand()$
19: $n_{nearest} \leftarrow NearestNeighbor(T, q_{rand})$
20: $q_{new} \leftarrow ComputeQNew(Q(n_{nearest}), q_{rand})$
21: $I \leftarrow FindIntersections([Q(n_{nearest}), q_{new}], F)$
22: $path \leftarrow UpdatePath(P(n_{nearest}), I\}$
23: **if** $(path \neq null)$ **then**
24: $n_{new} \leftarrow \{q_{new}, path\}$
25: $n_{nearest}.Add(n_{new})$
26: **end if**
27: **return** n_{new}

HRRT()
28: $n_{new} \leftarrow \{q_{start}, \emptyset\};\ n_{goal} \leftarrow \{q_{goal}, tPath\}$
29: $T.Add(n_{new})$
30: **while** $Distance(Q(n_{new}), Q(n_{goal})) > distThreshold$ **do**
31: $n_{new} \leftarrow Extend(T, n_{goal}, refFrame)$
32: **end while**

Results

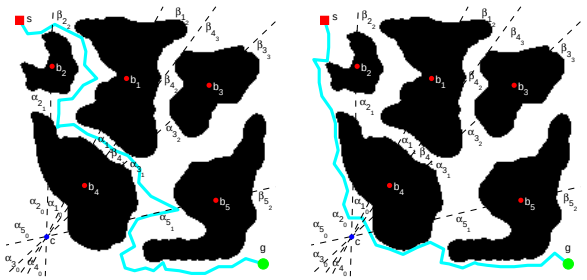
The extension of the Jenkins method and the path planning algorithm we propose have been implemented and tested in different scenarios. To identify the obstacles of the scenarios, we have adapted a Component-Labeling algorithm (CL) that efficiently labels connected cells and their contours in greyscale images at the same time (Chang, jen Chen, and Lu 2004). For the construction of the reference frame, the c point has been set at a fixed position in 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.

[2]Notice that it is possible to intersect with more than one segment of the reference frame depending on the step between $n_{nearest}$ and q_{new} and how close these nodes are to the c point.

Simulated Results

Figure 5 depicts paths of two homotopy classes in a 200x200 pixels bitmap used as cluttered environment to test extension of the Jenkins method and the HRRT algorithm. 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 optimal path cost computed with the A* algorithm. The time to build the topological graph, generate the homotopy classes with their lower bound and compute the path with the HRRT for the 13 homotopy classes is $281.11ms$. To ensure the stability of the results, the path cost and time are the average of 100 executions.

Path Planners Comparison The scalability of the method has been tested in a bitmap of 1000x1000 pixels with 15 irregular obstacles (Figure 6). The construction of the refer-



$2\ \alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$ $1\ \alpha_{5_0}\alpha_{3_0}\alpha_{4_0}\alpha_{1_0}\alpha_{2_0}$

Figure 5: Paths of the homotopy classes with the smaller lower bound (index 2) and with the best path cost (index 1) in Table 1.

Idx	Homotopy class	Lower bound	Cost	Cumulative time (ms)
2	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.83	1.35	22.53
8	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.84	1.44	36.18
3	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	0.90	1.50	50.43
9	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	0.91	1.49	55.09
1	$\alpha_{5_0}\alpha_{3_0}\alpha_{4_0}\alpha_{1_0}\alpha_{2_0}$	0.98	1.18	61.62
11	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	0.99	1.71	81.10
10	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.01	1.64	99.31
13	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.05	1.31	120.67
5	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	1.16	2.06	139.83
4	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.18	2.16	170.06
12	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.19	1.80	193.43
7	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.22	1.69	243.37
6	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.36	2.34	281.11

Table 1: Homotopy classes of Figure 5 environment sorted by their lower bound.

ence frame, the topological graph and the generation of 112 homotopy classes with their lower bound computation took $0.304s$. Figure 7 depicts the normalized cost with respect to the optimal path cost and the computation time for each homotopy class sorted by their normalized lower bound. Each path has been generated 50 times and the figure represents the average of them.

It is difficult to compare our proposal because, to best of the authors' knowledge, there is no other probabilistic sampling-based path planning algorithm that guides the search topologically. Nevertheless, we have implemented the A*, the RRT and their respective anytime versions (ARA* and ARRT) to enhance the comparison. As a result of the HRRT, we have chosen the five homotopy classes with the smaller lower bound. These classes are listed in Table 2 and their possible paths are depicted in Figure 6.

The results of the RRT and ARRT planners are the average of 100 executions. As shown in Figure 8, the RRT algorithm took $0.012s$ to compute a path with a cost of 1.48 times the optimal. The ARRT took around $2s$ to obtain the first solution and $52s$ to obtain all of them, but it ensured that any new generated solution was closer to the optimal one. The A* returned the optimal path in $11.9s$ and the ARA* generated the first solution in $8.32s$ and found the optimal solution after $301s$. Our method computed the best solution (index 25) in $0.373s$ with a cost 1.39 times the optimal and obtained the path for the five homotopy classes with the smaller lower bound in $0.603s$.

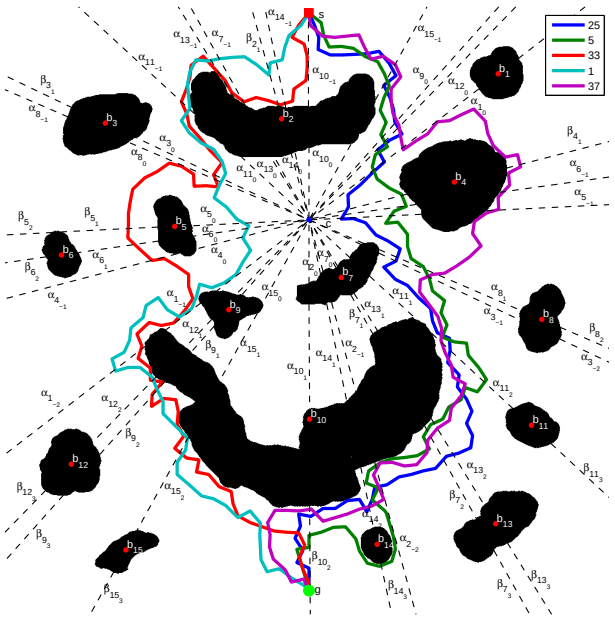


Figure 6: Paths of the five homotopy classes with the smaller lower bound. The class associated to the index can be found in Table 2.

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

Table 2: The five homotopy classes of Figure 6 environment with the smaller lower bound with their index.

The HRRT computation time was better than the deterministic approaches and the ARRT. The reduced computation time of the RRT could not be reached due to the computation of the homotopy classes and the extra load of checking the topological restrictions of the HRRT. Despite the first homotopy class according to its lower bound (index 25) had a lower cost than the RRT solution, the difference was small. However, following a homotopy class added information about how the obstacles were avoided. As a subject of the further research work, we think it could be possible to speed up the HRRT by guiding the random sampling towards the regions of the reference frame that follow the homotopy class instead of preventing the tree growing out of them.

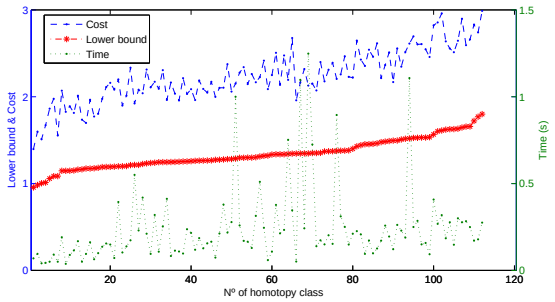


Figure 7: Normalized cost, normalized lower bound and computation time for paths generated with the HRRT for each homotopy class.

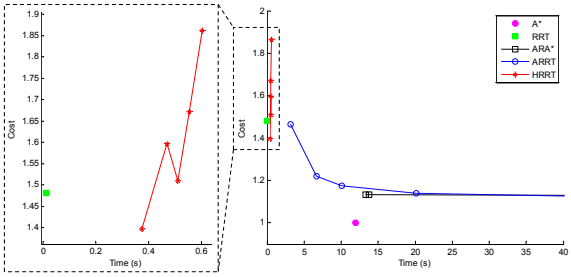


Figure 8: Comparison of the HRRT paths of the five homotopy classes with the smaller lower bound vs A*, RRT, ARA* and ARRT algorithms.

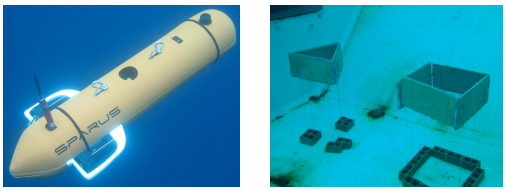
Experimental Results

The algorithm described in this paper has been tested with the SPARUS^{AUV} (Figure 9a). It is a $35Kg$ torpedo shaped vehicle of $1.22m$ length x $0.23m$ diameter whose motion is controlled by three Seabotix thrusters. It is equipped with an embedded computer with an Intel®Core^{TM} Duo Processor U2500@$1.2GHz$. Among its sensor suit, the robot has a MTi Motion Reference Unit (MRU) from XSens Technologies, a Micron Mechanical Scan Imaging Sonar (MSIS) from Tritech, and a Doppler Velocity Log (DVL) from LinkQuest which also includes a compass/tilt sensor.

The experiment took place in the Underwater Robotics Lab. of the University of Girona. In order to put obstacles of different shapes and sizes stacked in the water tank, we built a set of plates, each one made of a 1.20x0.60x0.04m roof insulator panel covered with a plastic mesh and concrete in both sides.

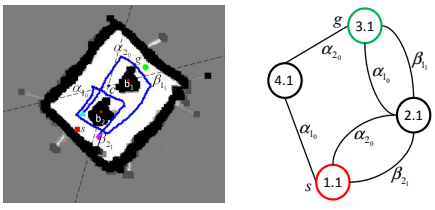
For the experiment, a triangular and squared obstacles where set up. Each panel was stack at $3m$ depth using weights (Figure 9b). The MSIS was configured to scan the whole 360° sector and it was set to fire up to a $5m$ range with a $0.1m$ resolution and a 1.8° angular step. The trajectory of the robot is based on dead-reckoning, computed using the velocity readings coming from the DVL and the heading data obtained from the MRU sensor, both merged with an Extended Kalman Filter (EKF).

During the experiment, the robot was teleoperated through the obstacles as shown in the trajectory of Figure 10a. While navigating at $3m$ depth, the vehicle was building a 18x16m OGM with a $0.1m$ resolution using the navigation from the EKF and the beam information received from the MSIS (Hernández et al. 2009). The inverse sensor model of the MSIS was implemented as a modified version of a regular sonar sensor with an overture of 3°. Figure 10a also depicts the Configuration Space (C-Space) based on the OGM, the obstacles identified by the CL algorithm, the reference frame and its topological graph (Figure 10b). Using the modified version of the Jenkins method, four homotopy classes were generated. Table 3 shows the homotopy classes with their path lengths and Figure 11 depicts each path with its corresponding tree generated with the HRRT algorithm. The process of applying the CL algorithm, the reference frame and topological graph construction and the generation of the paths in the C-Space using the HRRT took less than $100ms$.



a) SPARUS^{AUV} b) Underwater environment

Figure 9: Real experiment set up.



a) C-Space with reference frame b) Topological graph

Figure 10: Trajectory of the robot and the reference frame plotted in the 18x16m C-Space with $0.1m$ resolution and its topological graph.

Conclusions and Future Work

This paper proposes a method to guide path planning algorithms with topological information. Given a map with obstacles, we first use an extension of the Jenkins method to construct a reference frame which allows representation of any path in the workspace as a topological sequence. The ex-

Homotopy class	Lower bound (m)	Length (m)
$\alpha_{2_0}\beta_{1_1}$	7.47	13.62
$\alpha_{1_0}\alpha_{2_0}$	7.53	12.42
$\beta_{2_1}\beta_{1_1}$	7.65	11.15
$\beta_{2_1}\alpha_{1_0}$	7.91	14.24

Table 3: Homotopy classes generated for the Underwater Robotic Lab. environment with their length sorted according to the lower bound.

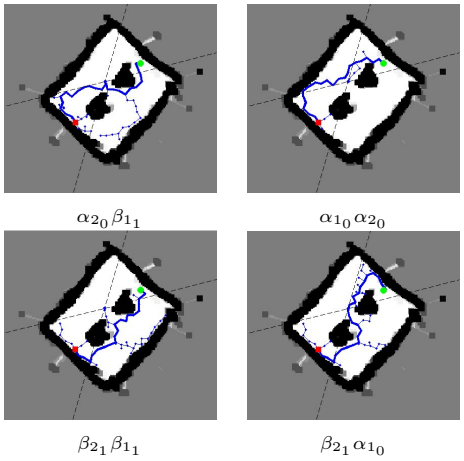


Figure 11: Paths and trees generated with HRRT using homotopy classes from Table 3. The squared and the circular points correspond with the start and goal points.

tension of the Jenkins method constitutes the main and first contribution of this paper. As a second contribution, we have developed the HRRT, a path planning algorithm that generates paths in the workspace following the homotopy classes previously found and sorted according to a lower bound. The effectiveness of the algorithm has been shown in simulated scenarios. As a third contribution of this paper, our proposal has also been tested in real conditions with an underwater robot in a controlled unknown environment to test its applicability to real applications. While navigating, the robot was generating the C-Space based on an OGM, built using navigation and MSIS data. The robot identified the obstacles of the scenario with the CL algorithm, applied the modified Jenkins method to generate the reference frame, the topological graph, and generated the path for each homotopy class using the HRRT algorithm in less than $100ms$, which is enough to accomplish our robot real-time specifications. Future work will consist in using the paths generated by the HRRT to guide the robot autonomously. The algorithm will allow generating new paths each time that substantial changes will be detected in the OGM.

The HRRT algorithm we propose is based on an implementation of the RRT that guides the random sampling towards the goal without considering that it has to follow a particular homotopy class. The current implementation of the HRRT just prevents the tree to grow out of the regions of the reference frame that do not follow the homotopy class. Therefore, we think it is possible to improve the performance and the quality of the solution of the HRRT by guiding the random sampling towards the regions of the reference frame that follow the homotopy class.

Acknowledgments

We gratefully acknowledge the support from Spanish government under the grant DPI2008-06545-C03-03 and the TRIDENT EU FP7-Project under the grant agreement No: ICT-248497.

References

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, 1230–1237.

Cabello, S.; Liu, Y.; Manler, A.; and Snoeyink, J. 2002. Testing homotopy for paths in the plane. In *Proceedings of the Symposium on Computational Geometry (SoCG)*.

Chang, F.; jen Chen, C.; and Lu, C.-J. 2004. A linear-time component-labeling algorithm using contour tracing technique. *Comput. Vis. Image Underst* 93:206–220.

Chazelle, B. 1982. A theorem on polygon cutting with applications. In *23rd Annual Symposium on Foundations of Computer Science (SFCS '08)*, 339 –349.

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

Ferguson, D., and Stentz, A. 2007. Anytime, dynamic planning in high-dimensional search spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*.

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)*.

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)*, 17–24. New York, NY, USA: ACM.

Hernández, E.; Ridao, P.; Mallios, A.; and Carreras, M. 2009. Occupancy grid mapping in an underwater structured environment. In *Proceedings of the 8th IFAC International Conference on Manoeuvring and Control of Marine Craft (MCMC)*.

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.

Kim, J., and Ostrowski, J. 2003. Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings of the IEEE International Conference on Intelligent Robotics and Automation (ICRA)*, volume 2, 2200–2205 vol.2.

Lavalle S. M., K. J. J. 1999. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on In Robotics and Automation (ICRA)*, volume 1, 473–479.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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.