Proceedings of the 18th World Congress
The International Federation of Automatic Control
Milano (Italy) August 28 - September 2, 2011

A Search-based Path Planning Algorithm with Topological Constraints. Application to an AUV ⋆

E. Hernández * **M. Carreras** * **P. Ridao** * **J. Antich** **
A. Ortiz **

* *Department of Computer Engineering, University of Girona, 17071 Girona, Spain (e-mail: {emilihb,pere,marcc}@eia.udg.edu).*
** *Department of Mathematics and Computer Science, University of the Balearic Islands, 07122 Balearic Islands, Spain (e-mail: {javi.antich,alberto.ortiz}@uib.es).*

Abstract:
The paper proposes a method that uses topological information to guide the path search in any 2D workspace. Our method generates topological paths taking into consideration the constraints of the workspace. Then, a path planner based on the A*, called *Homotopic A** (HA*), guides the path search in the workspace using the generated topological paths. 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.

Keywords: Marine robotics, path planning, marine control architectures.

1. INTRODUCTION

The main goal of the 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 safe path towards the goal. Once a path has been found, the real-time guidance of the vehicle is assumed to be achieved conveniently merging a path following algorithm with a simple reactive obstacle avoidance technique. The motion system has to be able to update the map, and hence, the path according to the information obtained from the unknown environment in a reduced amount of time. 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 flight 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 a path planning algorithm to generate a path through the local map in a very short time to meet realtime requirements. This problem has been commonly tackled by the mobile robotics community using anytime planners (Ferguson et al. [2005]): they find a first solution, possibly highly suboptimal, very quickly and then they refine it until time runs out.

Most of the anytime path planning algorithms require an ϵ value and a decrement factor (Likhachev et al. [2005], Ferguson and Stentz [2007]). The ϵ value is a multiplication factor used to control the cost of the generated solution. At each iteration, it is decreased by the decrement factor until $\epsilon = 1$, if time does not expire before. Anytime Repairing A* (ARA*) proposed by Likhachev et al. [2004] is a reference within the class of deterministic/heuristic based anytime path planners. It 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.

Topological approaches are another way to tackle the path planning problem. Essentially, this kind of solution works with a graph-based abstraction of the workspace, what makes the environment to be represented by a reduced number of potential states over the aforementioned strategies (Dudek et al. [1991], Fabrizi and Saffiotti [2000]). Visibility graphs-based approaches constitute a well known class of algorithms in this regard (Latombe [1991]). Another set of solutions adds a further level of abstraction by means of the homotopy concept, what leads to working with homotopy classes instead of standard topological paths (Jenkins [1991], Cabello et al. [2002]).

⋆ This research was sponsored by the Spanish government (DPI2008-06545-C03-03 and -02) and the TRIDENT EU FP7-Project under the Grant agreement No: ICT-248497.

This paper proposes the use of homotopy classes to guide topologically a path planning algorithm. We generate the homotopy classes that can be followed in any 2D workspace using the method we presented in Hernández et al. [2011]. Using the topological information, the path planner does not have to explore the whole space but the space confined in a homotopy class. If the homotopy classes that most-probably contain the lower cost solutions are known, the algorithm can generate some good solutions very fast and therefore, act as an anytime algorithm. The path planner we propose to follow homotopy classes is called *Homotopic A** (HA*), and is based on the A* algorithm, which has been applied successfully in many planning problems (Dolgov et al. [2010]). The HA* 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 each homotopy class that has 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. The HA* performance and scalability has been tested in simulation. 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 HA* was applied generating a path for each homotopy class on the map.

The paper is structured as follows. Section 2 describes the method to obtain topological paths from a metric environment. Section 3 describes the topological-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

Given a workspace with obstacles, in Hernández et al. [2011] we propose a method to generate a topological representation of the environment which is used to compute all the different homotopy classes from the start point to the end point. 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.

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. The whole construction process is summarized in three steps:

(1) Select a random point inside 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.

Using the reference frame, any path p can be defined by the sequence of labels of the segments being crossed in order from the starting to the ending point. For instance, Fig. 1a depicts a reference frame for a scenario with two obstacles. The path that traverses it is labeled $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$. 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_s} 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. With the notation used, 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 Fig. 1 the canonical sequence of the path $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$ is $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$.

2.2 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. Its construction can be divided in three steps:

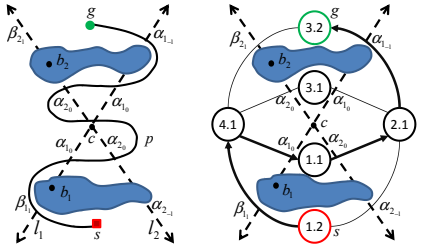
(1) The lines of the reference frame divide the metric space into regions or *wedges* and the obstacles that intersect with more than one line at the same time split these wedges into *sub-wedges*. Each sub-wedge represents a node of G.
(2) Each node of G is labeled according to the wedge w and sub-wedge 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 edge of G 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 G it turns into traversing the graph from the starting node to the ending node[1]. Fig. 1a depicts a path in the reference frame and Fig. 1b its equivalent description in the topological graph.

2.3 Generation of Homotopy Classes

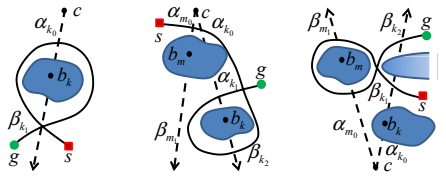
Once the topological graph is constructed, it is traversed using the version of the BFS algorithm used in Hernández et al. [2011]. The BFS is a graph search algorithm that begins at the root node and explores all the neighboring

[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

Fig. 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.



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

Fig. 2. Examples of the restriction criteria.

nodes. Then, for each of those nearest nodes, it explores their unexplored neighbors. The process is repeated until the goal is found. The algorithm runs 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 $\chi_{k_s}...\chi_{k_t}...\chi_{k_u}$ 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.

  Fig. 2a depicts an example of the simple wrap criterion with path $\beta_{k_1}\alpha_{k_0}\beta_{k_1}$. Fig. 2b shows a wrap with the path $\alpha_{m_0}\alpha_{k_0}\beta_{k_2}\alpha_{k_1}$ and Fig. 2c 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.

3. GUIDED PATH PLANNING

Once the homotopy classes are computed, a path planning algorithm has to find a path in the workspace that follows a given homotopy class, which essentially means to turn a topological path into a metric one. 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 checking the intersections –in order– from the initial configuration to the current configuration. We propose a variation of the A* algorithm called *Homotopic A** (HA*), which just allows to explore the zones of the workspace that satisfy a given homotopy class

The algorithm is detailed in Alg. 1. The states of the algorithm 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 A*, the visited states are stored in a list V and the open states are processed according their position in a priority queue $OPEN$. Each state in this queue is ordered according to the sum of its current path cost from the start, $g(s)$, and a heuristic estimate of its path cost to the goal, $h(s, s_{goal})$. The state with the minimum sum is at the top of the priority queue.

The algorithm receives as input the start configuration q_{start}, the goal configuration q_{goal}, a candidate homotopy class to follow H and the reference frame F. The configurations q_{start} and q_{goal} are used to set up the initial state s_{start} and the goal state s_{goal} (line 34). The function $ComputePath$ computes the shortest path that follows H. It starts by adding the s_{start} into the $OPEN$ queue. While $OPEN$ is not empty, the function pops the state s at the top of the queue. If $s = s_{goal}$ (line 14), then a path that follows H have been found and the function returns with success. Otherwise, for all the configurations q' reachable from $Q(s)$, the function $FindIntersections$ (line 18) returns the intersections of the segment $[Q(s), q']$ with F sorted by distance[2]. Then the $UpdatePath$ (line 19) generates the new topological path p according to the intersections. No intersection with F means that the explored configuration is in the same subwedge of the workspace and the function returns $P(s)$. If there are intersections and these intersections follow H, the function returns $P(s) \cup I$ in order to create a candidate new state s'. If s' has already been visited (line 21) and its cost $g(s')$ plus the cost of traversing from s to s', $c(s, s')$ is less than its current cost (line 22), $g(s')$ is set to this new, lower value. If s' does not exists in V, it is added into both, the $OPEN$ queue and the visited nodes list V.

The completeness of the HA* is ensured because when s_{goal} is not reachable, the algorithm will explore all the

[2] Notice that it is possible to intersect with more than one segment of the reference frame depending on how close $Q(s)$ and q' are to the c point.

states in $OPEN$ before returning that no path has been found. On the other side, when s_{goal} can be reached, the HA* will find the solution following the process described in the paragraph before.

Algorithm 1 Homotopic A*

FindIntersections$([q, q'], F)$
$r \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $|F|$ **do**
  if $x \leftarrow Intersection([q, q'], F[i]) \neq null$ **then**
    $r \leftarrow r \cup \{Edge(i), Distance(q, x)\}$
  end if
end for
$r \leftarrow SortByDistance(r)$
return r

ComputePath(s_{start}, s_{goal}, F)
$OPEN \leftarrow \emptyset$; $V \leftarrow \emptyset$
$OPEN.push(s_{start})$
repeat
  $s \leftarrow OPEN.top()$
  $OPEN.pop()$
  if $s = s_{goal}$ **then**
    return $true$
  end if
  for all $q' \in Succ(Q(s))$ **do**
    $I \leftarrow FindIntersections([Q(s), q'], F)$
    if $p \leftarrow UpdatePath(P(s), I)$ **then**
      $s' \leftarrow \{q', p\}$
      if $Exists(V(s'))$ **then**
        if $g(s') + c(s, s') < g(V(s'))$ **then**
          $g(s') \leftarrow g(s) + c(s, s')$
        end if
      else
        $g(s') \leftarrow g(s') + c(s, s')$
        $OPEN.push(s')$ with $g(s') + h(s', s_{goal})$
        $V \leftarrow V \cup s'$
      end if
    end if
  end for
until $|OPEN| > 0$
return $false$

HA*$(q_{start}, q_{goal}, H, F)$
$s_{start} \leftarrow \{q_{start}, \emptyset\}$; $s_{goal} \leftarrow \{q_{goal}, H\}$
$ComputePath(s_{start}, s_{goal}, F)$

4. RESULTS

The topological path search 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 et al. [2004]). For the construction of the reference frame, the c point and the b_k points have 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.

4.1 Simulated Results

Fig. 3 depicts the bitmap of 1000x1000 pixels with 13 irregular obstacles. The construction of the reference frame,

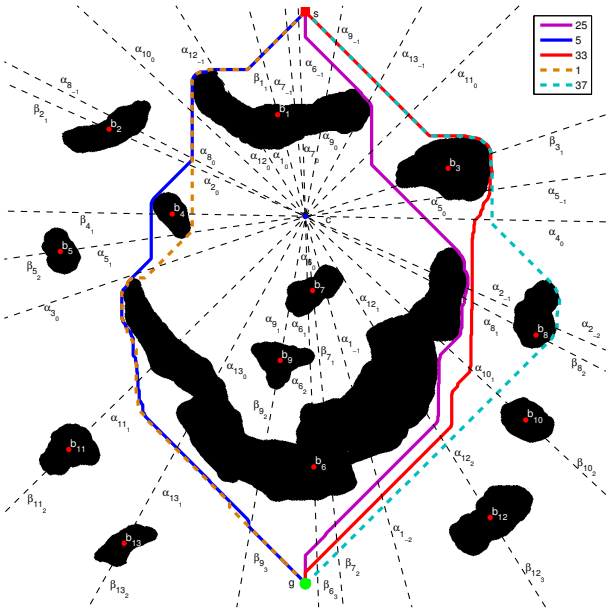


Fig. 3. Paths of the five homotopy classes whose paths are closer to optimal. The class associated to the index can be found in Table 1.

the topological graph and the generation of 96 homotopy classes took 0.876s. Fig. 4 depicts the normalized cost and the computation time for each homotopy class sorted by the quality of its solution. Then, it is possible to know which is the homotopy class that contains the optimal path. Nevertheless, the challenge consist on finding a quantitative measure of each homotopy class to estimate their quality before computing their path. Then it would be possible to plan at the topological level before applying the HA* to the optimal class, which is the subject of the further research work.

Although it is difficult to compare our proposal because, to best of the authors'knowledge, there is no other deterministic path planning algorithm based on the A* that guides the search topologically, we have implemented the A*, and its anytime version (ARA*) to enhance the comparison. For plotting purposes and as a result of the HA*, we have chosen the five homotopy classes whose paths cost are closer to the optimal solution, which has been obtained with the A* algorithm. These classes are listed in Table 1 and their paths are depicted in Fig. 3. Notice that the index of each homotopy class in Table 1 and Fig. 3 corresponds to its order of generation with the modified BFS algorithm.

Idx	Homotopy class
25	$\alpha_{9_{-1}}\alpha_{13_{-1}}\alpha_{11_0}\alpha_{3_0}\alpha_{5_0}\alpha_{4_0}\alpha_{2_{-1}}\alpha_{8_1}\alpha_{10_1}\alpha_{12_2}\alpha_{1_{-2}}\beta_{7_2}\beta_{6_3}$
5	$\alpha_{6_{-1}}\alpha_{7_{-1}}\beta_{1_1}\alpha_{12_{-1}}\alpha_{10_0}\alpha_{8_0}\alpha_{2_0}\beta_{4_1}\alpha_{5_1}\alpha_{3_0}\alpha_{11_1}\alpha_{13_1}\beta_{9_3}$
33	$\alpha_{9_{-1}}\alpha_{13_{-1}}\alpha_{11_0}\beta_{3_1}\alpha_{5_{-1}}\alpha_{4_0}\alpha_{2_{-1}}\alpha_{8_1}\alpha_{10_1}\alpha_{12_2}\alpha_{1_{-2}}\beta_{7_2}\beta_{6_3}$
1	$\alpha_{6_{-1}}\alpha_{7_{-1}}\beta_{1_1}\alpha_{12_{-1}}\alpha_{10_0}\alpha_{8_0}\alpha_{2_0}\alpha_{4_0}\alpha_{5_0}\alpha_{3_0}\alpha_{11_1}\alpha_{13_1}\beta_{9_3}$
37	$\alpha_{9_{-1}}\alpha_{13_{-1}}\alpha_{11_0}\beta_{3_1}\alpha_{5_{-1}}\alpha_{4_0}\alpha_{2_{-2}}\beta_{8_2}\alpha_{10_1}\alpha_{12_2}\alpha_{1_{-2}}\beta_{7_2}\beta_{6_3}$

Table 1. The five homotopy classes whose paths have the lower cost with their index.

As shown in Fig. 5, the A* returned the optimal path in 21.80s and the ARA* generated the first solution in 13.32s and found the optimal solution after 775s. The optimal solution computed by the A* was used to normalize the costs presented in Fig. 4 and Fig. 5. The HA* computed

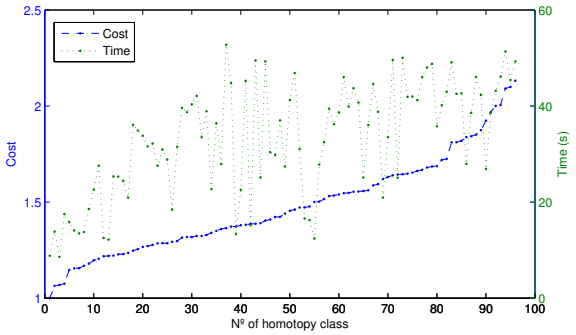


Fig. 4. Normalized cost and computation time for paths generated with the HA* for each homotopy class.

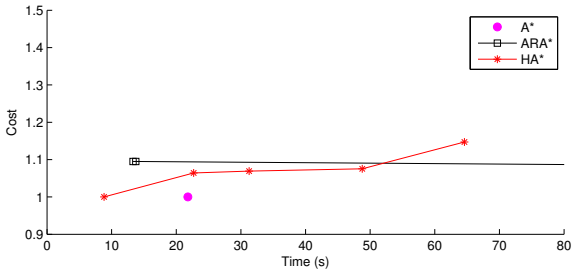


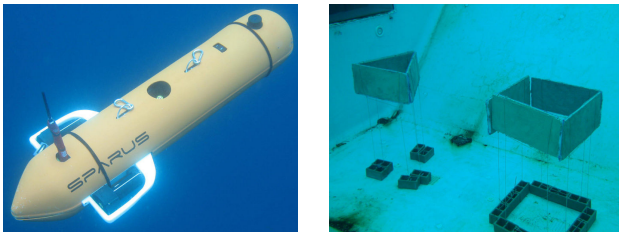
Fig. 5. Comparison of the HA* paths of the best five homotopy classes vs A* and ARA* algorithms.

the optimal path (index 25) in 8.83s and obtained the path for the five selected homotopy classes in 64.57s. Notice that computing the optimal path of the best homotopy class using the HA* was 2.47 times faster than the conventional A* and 1.51 times faster than the ARA* when computing the first solution. However, the homotopy class that contains the optimal path is not known in advance.

4.2 Experimental Results

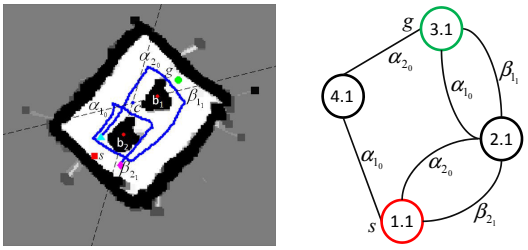
The algorithm described in this paper has been tested with the SPARUS^{AUV} (Fig. 6a) , a 35Kg torpedo shaped vehicle of 1.22m length x 0.23m diameter whose motion is controlled by three Seabotix thrusters. It is equipped with an embedded computer with an Intel®Core^{TM} Duo Processor U2500@1.2GHz. Its sensor suite is composed by a forward-looking and down-looking color video cameras, a MTi Motion Reference Unit (MRU) from XSens Technologies, a Micron Mechanical Scan Imaging Sonar (MSIS) from Tritech, an echosounder, a pressure sensor, a Doppler Velocity Log (DVL) from LinkQuest which also includes a compass/tilt sensor and several temperature, voltage, pressure sensors and water leak detectors for safety purposes.

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. The insulator is a cheap alternative to wood which offers buoyancy and the concrete adds weight and allows to simulate harbors texture. Notice that none of the materials of the panels are ferromagnetic,



a) SPARUS^{AUV} b) Underwater environment

Fig. 6. Real experiment set up.



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

Fig. 7. Trajectory of the robot and the reference frame plotted in the 18x16m C-Space with 0.1m resolution and its topological graph.

hence the compasses of the vehicle are not affected while navigating at close distance.

For the experiment, a triangular and a squared obstacles where set up. Each panel was stack at 3m depth using weights (Fig. 6b). 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). To avoid perturbations on the DVL measurements, the test zone was limited in the maximum depth area, which is at the centre of the pool.

Although in the final system the local map building and the path planning processes will run in parallel, for the sake of simplicity they have been tackled here independently. Hence, during the experiment, the robot was teleoperated through the obstacles as shown in the trajectory of Fig. 7a. 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°. Fig. 7a 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 (Fig. 7b). Four homotopy classes were generated. Table 2 shows the homotopy classes with their path lengths and Fig. 8 depicts each path generated with the HA* 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 HA* took less than 150ms.

Homotopy class	Length (m)
$\alpha_{1_0}\alpha_{2_0}$	8.38
$\alpha_{2_0}\beta_{1_1}$	8.76
$\beta_{2_1}\alpha_{1_0}$	9.58
$\beta_{2_1}\beta_{1_1}$	8.40

Table 2. Homotopy classes generated for the Underwater Robotic Lab. environment with their length.

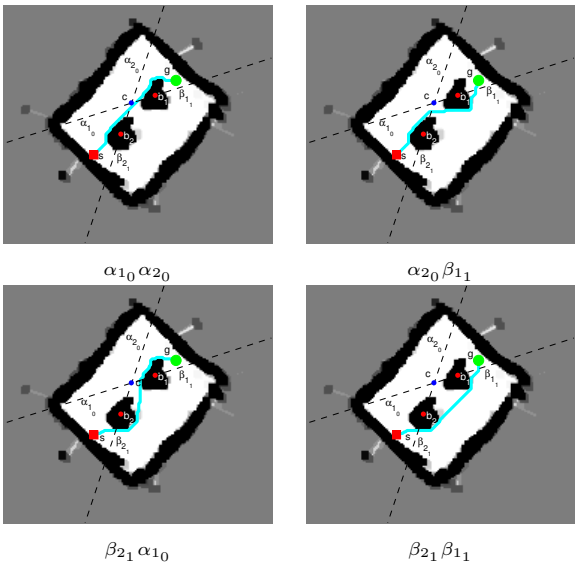


Fig. 8. Paths generated with HA* using homotopy classes from Table 2.

5. CONCLUSIONS AND FUTURE WORK

This paper proposes a method that uses topological information to guide path planning algorithms. Given a map with obstacles, we first generate a reference frame which allows to represent any path in the workspace as a topological sequence. We propose the HA*, a path planning algorithm that generates paths in the workspace following the homotopy classes previously found. The effectiveness of the algorithm has been shown in a big and relative complex simulated scenario. 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, computed the reference frame, the topological graph, and generated the path for each homotopy class using the HA* algorithm in less than $150ms$, which is enough to accomplish our robot realtime specifications for the scenario purposed. The HA* is, to the best of the authors'knowledge, the first planner that finds the optimal path of an specific homotopy class. Therefore, it can be used as a ground truth for other modified planners to follow homotopy classes such as the probabilistic approaches.

Future work will consist in using the paths generated by the HA* to guide the robot autonomously. The algorithm will allow generating new paths each time that substantial changes will be detected in the OGM. And, as stated in section 4.1, in finding a quantitative measure for homotopy classes to get an estimation of their quality before computing their path with the HA* in order to improve the effectiveness of the whole process

REFERENCES

S. Cabello, Y. Liu, A. Manler, and J. Snoeyink. Testing homotopy for paths in the plane. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, 5-7 June 2002.

Fu Chang, Chun jen Chen, and Chi-Jen Lu. A linear-time component-labeling algorithm using contour tracing technique. *Comput. Vis. Image Underst*, 93:206–220, 2004.

D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Rob. Res.*, 29(5):485–501, 2010. ISSN 0278-3649.

G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859 –865, December 1991. ISSN 1042-296X.

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

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

D. Ferguson, M. Likhachev, and A. Stentz. 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)*, June 2005.

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

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

K. D. Jenkins. 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, June 1991.

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

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

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