Homotopic Path Planning for an AUV on Maps Improved with Scan Matching ⋆

Emili Hernández * **Marc Carreras** * **Pere Ridao** *
Angelos Mallios *

* *Department of Computer Engineering, University of Girona, 17071 Girona, Spain (e-mail: {emilihb,pere,marcc,amallios}@eia.udg.edu).*

Abstract:
The main contribution of this paper is an application for an Autonomous Underwater Vehicle (AUV) which first improves dead-reckoning navigation through scan matching techniques to build and Occupancy Grid Map (OGM) and then performs path planning to find safe paths towards the goal. The path planning process is guided with homotopy classes, which topologically describe how paths avoid the obstacles. Then, a homotopic path planner called Homotopic Bug (HBug) efficiently computes a feasible path for each homotopy class in the OGM. The application has been tested on an AUV in a controlled unknown scenario.

Keywords: Marine systems, path planning, scan matching

1. INTRODUCTION

The work presented in this paper is focused on a motion system for an Autonomous Underwater Vehicle (AUV), which it has to be able to perceive the environment, build an internal local map of the explored area, compute a safe path through the map and, finally, generate the high-level commands to follow it. The target application of this research is autonomous navigation and guidance in harbors, ocean observatories, fleets of ships or steep seabeds for inspection, where the robot is navigating at a fixed depth or at a fixed distance to the seafloor.

In this paper we propose a map building and a path planning application for an AUV. Given a controlled unknown scenario, our method improves the dead-reckoning navigation with the MSISpIC, a probabilistic sonar scan matching algorithm (Hernández et al. [2009b]). This algorithm is well suited to be used with mechanically scanned profiling or imaging sonars commonly available in nowadays AUVs. While being corrected, the navigation is used to build an Occupancy Grid Map (OGM), where path planning is performed.

Despite path planning literature is extensive, most of solutions only generate a path to the goal and do not provide information about how this path avoids the obstacles, which is interesting for surveilling applications. The solution to this problem can be achieved though topological restrictions provided by *homotopy classes*. Essentially, two paths that share the start point and the goal point belong to the same homotopy class if one can be deformed into the other without encroaching any obstacle. Homotopy classes are considered in problems such as VLSI and network routing, where given an input path, it is required to compute its homotopic shortest one (Efrat et al. [2002]). However, very few of them are focused on robotic applications (Schmitzberger et al. [2002]).

The application proposed in this paper addresses path planning by means of homotopy classes, which provide topological information to guide path planners (Hernández et al. [2011a]). Using this topological information, the path planner does not have to explore the whole space but the space confined in a homotopy class. Our method generates the homotopy classes and sort them according to a lower bound (Hernández et al. [2011c]), which let us to know those classes that most-probably contain the lower cost solutions. Therefore, it is possible to generate some good solutions very fast. Moreover, homotopy classes allow to reach a goal position by avoiding obstacles in different manners, which has interest when the robot is surveilling a particular area. In order to explore efficiently the homotopy classes, we propose a new path planner, called *Homotopic Bug* (HBug), which is inspired on the Bug family algorithms (Ng and Braunl [2007]). 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. The method also ensures that the homotopy class of the global optimal path is generated.

The feasibility of the work has been tested on an AUV in a water tank. The robot was teleoperated in a controlled unknown scenario. While navigating, the MSISpIC improved the dead-reckoning to generate a more reliable OGM. Then, the homotopy classes were obtained and the HBug computed a path for each class.

The paper is structured as follows. Section 2 describes the scan matching algorithm. Section 3 reports the method to generate the homotopy classes. Section 4 describes

⋆ This research was sponsored by the Spanish government (DPI2011-27977-C03-02), the TRIDENT EU FP7-Project under the Grant agreement No: ICT-248497 and PERG-GA-2010-276778 (Surf3DSLAM)

the path planner that we propose. Section 5 reports the experimental results and section 6 exposes the conclusions and future work.

2. SCAN MATCHING

Scan matching techniques estimate the robot relative displacement between two configurations by registering the overlap between two consecutive range scans. Usually, these techniques consist of a two steps iterative process. The first step is an association process in which each single data of the second scan looks for its correspondence in the first scan. The second step is a minimization process that estimates the solution according to the correspondences of the previous step. This process is repeated until convergence.

In this work, the navigation is improved using the MSISpIC algorithm (Hernández et al. [2009b]), an extension to the pIC method (Montesano et al. [2005]) adapted to be used with mechanically scanned imaging or profiling sonars. The pIC algorithm is a statistical extension of the ICP algorithm (Besl and McKay [1992]), which is able to deal with sparse sonar data. It was designed to be used with range scans gathered in one shot and hence, it cannot be applied to slow mechanically scanned sonars, commonly used in underwater applications. The MSISpIC compounds the robot trajectory with the range and bearing data of all the beams, represents them with their uncertainty in a unique reference frame using a scan grabbing procedure and then applies the standard pIC algorithm.

2.1 Scan Grabbing

The scan grabbing procedure builds a static scan referenced to a single frame to be used by the pIC algorithm. Whenever a sonar beam is read and segmented using a predefined threshold, an Extended Kalman Filter (EKF) similar to the one used in Ribas et al. [2008] estimates the robots's pose. The information of the system at step k is stored in the state vector $\mathbf{x}_k$ with estimated mean $\hat{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$:

$$\hat{\mathbf{x}}_k = \left[\hat{\eta}^B, \hat{\nu}^R\right]^T \quad \mathbf{P}_k = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)^T\right] \tag{1}$$

with:

$$\eta^B = [x, y, z, \phi, \theta, \psi]^T; \quad \nu^R = [u, v, w, p, q, r]^T \tag{2}$$

The vehicle's movement prediction is performed using the 6DOF kinematic model:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{w}_k) = \begin{bmatrix} \eta^B_k \\ \nu^R_k \end{bmatrix} = \begin{bmatrix} \eta^B_{k-1} + J(\eta^B_{k-1}) \left[\nu^R_{k-1}\Delta T + \frac{1}{2} w_k \Delta T^2\right] \\ \nu^R_{k-1} + w_k \Delta T \end{bmatrix} \tag{3}$$

$$J(\eta) = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi & 0 & 0 & 0 \\ s\psi c\theta & s\phi s\psi s\theta + c\psi c\phi & s\psi s\theta c\phi - s\phi c\psi & 0 & 0 & 0 \\ -s\theta & c\theta s\phi & c\theta c\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s\phi t\theta & c\phi t\theta \\ 0 & 0 & 0 & 0 & c\phi & -s\phi \\ 0 & 0 & 0 & 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \tag{4}$$

Although in this model the velocity is considered to be constant, in order to allow for slight changes, a velocity perturbation modeled as the integral of a stationary white noise w_k is introduced. The covariance matrix $\mathbf{Q_k}$ of this acceleration noise is diagonal and in the order of magnitude of the maximum acceleration increment that the robot may experience over a sample period.

$$E[w_k] = 0; \quad E[w_k w_j^T] = \delta_{k_j} \mathbf{Q_k} \tag{5}$$

Hence, w_k is an acceleration noise which is first integrated and then added in velocity, being nonlinearly propagated to the position.

Since we are only interested in the robot's relative displacement (and uncertainty) with respect to the beginning of the scan, a reset in position is performed (setting x, y, z to 0 in the vector state $\hat{\mathbf{x}}_k$) whenever a new scan is started. On the other side, ϕ, θ and ψ angle values are kept since a reset would mean unreal high rotations during the scan. The reset process also affects the x, y, and z terms of the covariance matrix $\mathbf{P}$. The modified filter provides the robot's relative position where the beams are gathered with their uncertainty accumulated during the scan. Hence, using a similar procedure than in Ribas et al. [2008], it is possible to reference all the ranges computed from the beams to the initial frame I, removing the distortion induced by the robot's motion.

Fig. 1 illustrates the scan grabbing process while the algorithmic notation is shown in Algorithm 1. First, the function $P2C$ transforms the range and bearing data ρ from Polar to Cartesian space. The result is the observed point referenced to the sensor frame S. Then, by means of a rigid body transformation, the point is referenced to the robot's frame R using $\mathbf{x_S^R}$. Next, the robot's relative position $\mathbf{x_R^B}$ obtained with the EKF is compounded with the robot's referenced point to get the data referenced to B frame, which is aligned to the north. Finally, the last compounding operation rotates the point to reference it to the initial frame I getting $\hat{\mathbf{n}}$, which is aligned with the initial heading of the robot at the beginning of the scan. Finally, the uncertainty $\mathbf{P_n}$ of the point $\hat{\mathbf{n}}$ is computed and the point is introduced in the new scan S.

Algorithm 1 Scan grabbing

***ScanGrabbing*()**

1: $ResetDeadReckoningXYZ()$
2: $\{\hat{\boldsymbol{x}}^I_B, \boldsymbol{P}_{IB}\} \leftarrow GetDeadReckoning()$
3: $S \leftarrow \emptyset$
4: **while** $|S| < beamsPerScan$ **do**
5: $\quad beam \leftarrow Get\&SegmentBeam()$
6: $\quad \{\hat{\boldsymbol{\rho}}, \boldsymbol{P}_\rho\} \leftarrow LocalMaximaFinder(beam)$
7: $\quad \{\hat{\boldsymbol{x}}^B_R, \boldsymbol{P}_{BR}\} \leftarrow GetDeadReckoning()$
8: $\quad \hat{\boldsymbol{n}} \leftarrow \hat{\boldsymbol{x}}^I_B \oplus \hat{\boldsymbol{x}}^B_R \oplus \boldsymbol{x}^R_S \oplus P2C(\hat{\boldsymbol{\rho}})$ {$\hat{\boldsymbol{\rho}}$ from local frame I}
9: $\quad \boldsymbol{P}_B \leftarrow \boldsymbol{J}_{B1\oplus}\boldsymbol{P}_{BR}\boldsymbol{J}^T_{B1\oplus} + \boldsymbol{J}_{B2\oplus}\boldsymbol{J}_{R\oplus}\boldsymbol{J}_{S\oplus}\boldsymbol{P}_\rho\boldsymbol{J}^T_{S\oplus}\boldsymbol{J}^T_{R\oplus}\boldsymbol{J}^T_{B2\oplus}$
10: $\quad \boldsymbol{P}_n \leftarrow \boldsymbol{J}_{I1\oplus}\boldsymbol{P}_{IB}\boldsymbol{J}^T_{I1\oplus} + \boldsymbol{J}_{I2\oplus}\boldsymbol{P}_B\boldsymbol{J}^T_{I2\oplus}$ {$\boldsymbol{P_\rho}$ from frame I}
11: $\quad S \leftarrow S \cup \{\hat{\boldsymbol{n}}, \boldsymbol{P}_n\}$
12: $\hat{\boldsymbol{q}} \leftarrow \hat{\boldsymbol{x}}^B_R$; $\boldsymbol{P_q} \leftarrow \boldsymbol{P}_{BR}$
13: **return** $\{S, \hat{\boldsymbol{q}}, \boldsymbol{P_q}\}$

2.2 The pIC algorithm

As stated before, the pIC algorithm is a statistical extension of the ICP algorithm that deals with sparse sonar data. Detailed in Algorithm 2, the inputs are the reference scan S_{ref}, the new scan S_{new} and the initial relative displacement estimation $\hat{\mathbf{q}}$ between them with its covariance

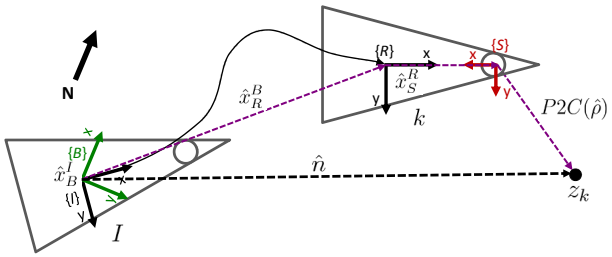


Fig. 1. Scan forming process: any beam k of the scan is represented with respect the position of the robots pose when the first beam I was gathered.

$\mathbf{P_q}$. The following procedure is iteratively executed until convergence. First, the points of S_{new} are compounded with the robot displacement $\mathbf{q_k}$. The result are the points of the new scan referenced to the reference frame ($\mathbf{c_j}$). Then, for each $\mathbf{c_j}$, a set A_j of all the compatible points in S_{ref} is established using a compatibility test over the Mahalanobis distance. Next, it is computed the virtual association point $\mathbf{a_j}$ as the expectancy over the random variable defined by the set A_j. After computing its uncertainty $\mathbf{P_{a_j}}$, it is possible to obtain the uncertainty $\mathbf{P_{e_j}}$ of the matching error ($\hat{\mathbf{a}}_\mathbf{j}-\hat{\mathbf{c}}_\mathbf{j}$). Then, it is possible to estimate the displacement $\hat{\mathbf{q}}_\mathbf{min}$, which minimizes the mean square error of the Mahalanobis distance between $\hat{\mathbf{a}}_\mathbf{j}$ and $\hat{\mathbf{c}}_\mathbf{j}$. This is done using Least Squares minimization method. If there is convergence the function returns, otherwise another iteration is required.

Algorithm 2 Probabilistic Iterative Correspondance

pIC$(S_{ref}, S_{new}, \hat{\boldsymbol{q}}, \boldsymbol{P_q})$

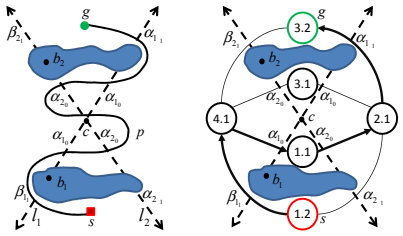
1: $k \leftarrow 0$; $\hat{\boldsymbol{q}}_k \leftarrow \hat{\boldsymbol{q}}$
2: **repeat**
3: **for** $j \leftarrow 1$ **to** $|S_{new}|$ **do**
4: $\hat{\boldsymbol{c}}_j \leftarrow \hat{\boldsymbol{q}}_k \oplus \hat{\boldsymbol{n}}_j$
5: $A_j \leftarrow \{\boldsymbol{r}_i \in S_{ref} / D_M^2(\boldsymbol{r}_i, \boldsymbol{c}_j) \leq \chi^2_{2,\alpha}\}$
6: $\hat{\boldsymbol{a}}_j \leftarrow \sum_{\boldsymbol{r}_i \in A_j} \hat{\boldsymbol{r}}_i p(\boldsymbol{r}_i = \boldsymbol{c}_j)$
7: $\boldsymbol{P_{a_j}} \leftarrow \sum_{\boldsymbol{r}_i \in A_j} [(\hat{\boldsymbol{r}}_i - \hat{\boldsymbol{a}}_j)(\hat{\boldsymbol{r}}_i - \hat{\boldsymbol{a}}_j)^T] p(\boldsymbol{r}_i = \boldsymbol{c}_j)$
8: $\boldsymbol{P_{e_j}} \leftarrow \boldsymbol{P_{a_j}} + \boldsymbol{J_q} \boldsymbol{P_q} \boldsymbol{J_q}^T + \boldsymbol{J_n} \boldsymbol{P_{n_j}} \boldsymbol{J_n}^T$
9: $\hat{\boldsymbol{q}}_{min} \leftarrow arg\ min_{\boldsymbol{q}} \left\{ \sum_j \left((\hat{\boldsymbol{a}}_j - \hat{\boldsymbol{c}}_j)^T \boldsymbol{P}_{\boldsymbol{e}_j}^{-1} (\hat{\boldsymbol{a}}_j - \hat{\boldsymbol{c}}_j) \right) \right\}$
10: **if** *Convergence*() **then**
11: $\hat{\boldsymbol{q}}_{pIC} \leftarrow \hat{\boldsymbol{q}}_{min}$
12: **else**
13: $\hat{\boldsymbol{q}}_{k+1} \leftarrow \hat{\boldsymbol{q}}_{min}$; $k++$
14: **until** *Convergence*() **or** $k \geq maxIterations$
15: **return** $\hat{\boldsymbol{q}}_{pIC}$

3. 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.

3.1 The reference frame

The reference frame determines in the metric space the topological relationships between obstacles and it is used



a) Reference frame b) Topological graph

Fig. 2. 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.

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, Fig. 2a 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]).

3.2 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 starting node to the ending node. Fig. 2a depicts a path in the reference frame and Fig. 2b its equivalent description in the topological graph.

3.3 Generation of Homotopy Classes

The topological graph is is traversed with a modified version of the Breadth-First Search (BFS) algorithm used in Hernández et al. [2011a,c], 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. Those homotopy classes which either self-intersect or whose canonical sequence is duplicated are not considered.

3.4 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 preference order to compute the homotopy classes path in the workspace when operating under time restrictions. Fig. 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 Fig. 3a).

4. HOMOTOPIC PATH PLANNING

Once the homotopy classes are computed and sorted according to their lower bound, we propose a bug-based path planner called *Homotopic Bug* (HBug) to generate paths in the workspace according to a given homotopy class. The algorithm tries to follow directly the lower bound path obtained with the funnel algorithm, which ensures that the homotopy class is being followed. 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 3, receives as an input 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 workspace. 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 data is 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 18). Finally, the function *BuildPath* creates the path P' in the workspace 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} 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 $(Q(n_h) - b_k)$ and $(Q(n_{h+1}) - b_k)$ computes the direction (line 12). 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.

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 13). In such case, d is obtained according to 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 perpendicular dot product form the start s to the first χ_k with the same subindex than l_k [1] (line 14). 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 of the χ_k that contains n_{h+1} (line 16).

Fig. 3b depicts an example for homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. The lower bound path (dashed line) intersects with the first obstacle generating two boundary nodes n_1 and n_2, both 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 s point 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.

4.1 Theoretical properties

Although the HBug has been designed to be used with the automated generation of homotopy classes, the algorithm itself has several properties:

- Complete. As a Bug-based algorithm, when there is a solution, the HBug finds it. On the other side, when there is no solution, the function *BuildPath* tries to follow the boundary of the obstacle that does not allow the solution to exists. However, by doing this, it crosses segments of the reference frame that do

[1] 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. This requirement also stands for the generation of the homotopy classes.

Algorithm 3 Homotopic Bug

BoundaryNodes(P)

$N \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $|P|-1, p_i \in P$ **do**
  $C \leftarrow$ ContourPoints(p_i, p_{i+1})
  for all $j \leftarrow 1$ **to** $|C|, c_j \in C$ **do**
    $k \leftarrow$ Label(c_j)
    $N \leftarrow N \cup \{c_j, k\}$
return N

ObstacleNodes(N, H, F)

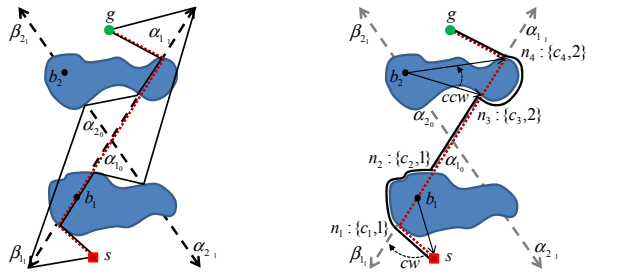
$O \leftarrow \emptyset$; $h \leftarrow 1$
while $n_h \in N/h < |N| - 1$ **do**
  $n_l \leftarrow$ last $n_j \in N/j > h$ without changing $Obst(n_h)$
  $b_k \leftarrow$ point of $Obst(n_h)$ in F
  $d \leftarrow (Q(n_h) - b_k)^{\perp} \cdot (Q(n_{h+1}) - b_k)$
  if $d = 0$ **then** {parallel}
    $d \leftarrow (s - b_k)^{\perp} \cdot$ (point of 1^{st} $\chi_k \in H - b_k$)
    $i_k \leftarrow$ index of $\chi_k \in H$ where n_{h+1} relies on
    if $|\chi_k| \in H_{1..i_k}$ is *even* **then**
      switch d
  $O \leftarrow O \cup \{n_h, n_l, d\}$
  $h \leftarrow l + 1$
return O

BuildPath(O)

$P' \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $|O|, o_i \in O$ **do**
  $P' \leftarrow P' \cup$ Boundary(o_i)
return P'

HBug(P, H, F)

$N \leftarrow$ BoundaryNodes(P)
$O \leftarrow$ ObstacleNodes$(N,\ H,\ F)$
$P' \leftarrow$ BuildPath(O)

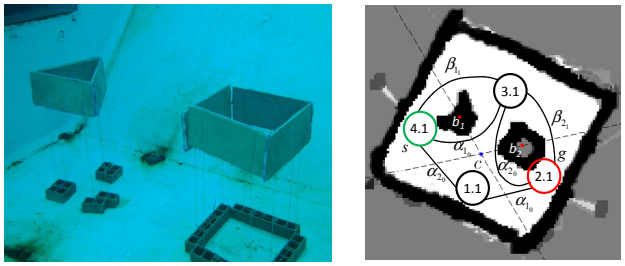


a) Channel and lower bound path b) Path computed with the HBug

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

not describe the specific homotopy class that it has to follow. To avoid checking the intersections with the reference frame, it is possible to known that no solution exists when the algorithm reaches again the hit point after surrounding the whole obstacle.

- Performance. Approaches based on the A* (Hernández et al. [2011b]) and Rapidly-exploring Random Trees (RRTs) (Hernández et al. [2011c]) consider the whole free regions of the workspace that can be reached within an specific homotopy class. However, the HBug only considers the lower bound path, which is already computed, while does not traverse an obstacle of the workspace. In such situation, the algorithm only looks into the free space that surrounds the intersected obstacle.
- Upper bound. For every homotopy class, the ideal path would be the lower bound path, whose length is



a) Underwater environment b) OGM

Fig. 4. Underwater environment and a detail of the OGM with its reference frame and topological graph.

denoted by LB. The lower bound path can only be reached when it does not intersect with any obstacle in the workspace. However, most of the homotopy classes requires to circumnavigate the n obstacles of the environment that crosses with the lower bound path. Depending on the shape of the obstacles, the worst possible case would be to follow almost the whole perimeter p of the involved obstacles. Thus, the upper bound UB is

$$UB \leq LB + \sum_{i=1}^{n} p_i \tag{6}$$

5. EXPERIMENTAL RESULTS

The method described in this paper has been tested with the Sparus AUV, a $35Kg$ torpedo shaped vehicle of 1.22x0.23m whose motion is controlled by three Seabotix thrusters. It is equipped with an embedded computer with an Intel® Core™ 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 Doppler Velocity Log (DVL) from LinkQuest.

The experiment took place in the Underwater Robotics Lab. of the University of Girona. For the experiment, a triangular obstacle and a squared obstacle composed of roof insulator panels covered with concrete where set up. Each panel was stack at 3m depth using weights (Fig. 4a). 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 dead-reckoning trajectory of the robot is computed using the velocity readings coming from the DVL and the heading data obtained from the MRU sensor, both merged with an EKF using a constant velocity model acceleration noise.

During the experiment, the robot was teleoperated through the obstacles at 3m depth. Fig. 5 shows an improved raw map obtained with the trajectory computed with the MSISpIC compared with the dead-reckoning trajectory.

Fig. 4b depicts a detail of a 0.1m resolution OGM obtained with the MSISpIC trajectory with the inverse sensor model of the MSIS implemented as a profiler sonar sensor with an overture of 3° (Hernández et al. [2009a]). During its construction, obstacles were expanded according to the robot'size in order to use the OGM as a Configuration Space (C-space), ensuring the generation of paths that would be feasible to follow with the vehicle. Fig. 4b also depicts the reference frame with its topological graph.

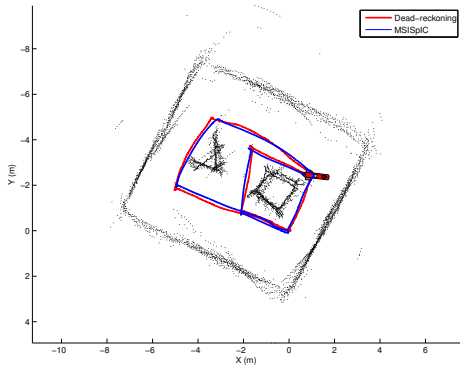


Fig. 5. Trajectory estimated with scan matching against dead-reckoning. Range data is plotted according to the MSISpIC trajectory.

Idx	Homotopy class	Lower bound(m)	Length(m)
1	$\alpha_{2_0}\alpha_{1_0}$	7.00	9.89
2	$\alpha_{2_0}\beta_{1_1}$	7.11	10.81
4	$\beta_{2_1}\beta_{1_1}$	7.50	10.53
3	$\beta_{2_1}\alpha_{1_0}$	7.75	11.14

Table 1. Homotopy classes with their path length sorted according to the lower bound.

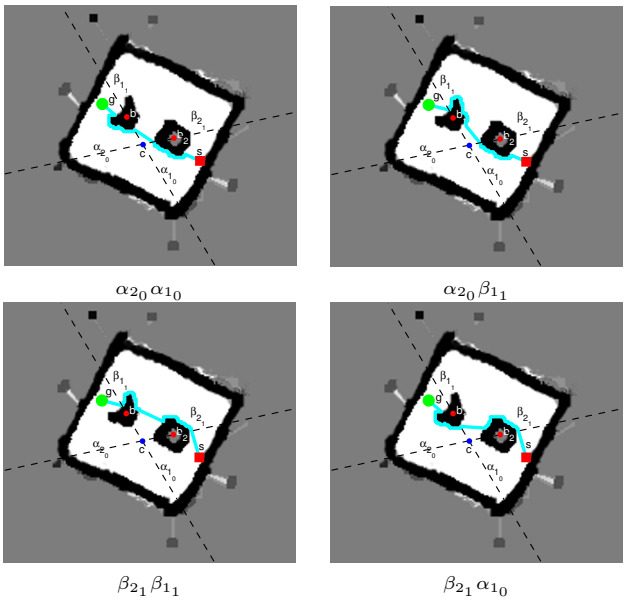


Fig. 6. Paths generated with HBug using homotopy classes from Table 1.

In this scenario four homotopy classes were generated. Table 1 shows the homotopy classes classified according to their lower bound and path length. Fig. 6 depicts each path generated with the HBug algorithm. Our method executed all the required steps to compute the four homotopy classes and their respective paths in the C-space using the HBug in just 157.85ms

6. CONCLUSIONS AND FUTURE WORK

In this paper we show a map building and a path planning application for an AUV. Given a controlled unknown scenario, our method first builds an OGM of the area traversed by the robot using the navigation computed with the MSISpIC, a probabilistic sonar scan matching algorithm designed to work with mechanically scanned profiling or imaging sonars. Then, our path planning method generates the homotopy classes sorted according to their lower bound, providing topological information about how paths avoid the obstacles. Next, the HBug algorithm generates paths in the workspace 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. Our proposal has been tested with an AUV in a controlled unknown environment showing that is fast enough to achieve our real-time specifications.

Future work will consist in improving the paths generated by the HBug taking into account the non-holonomic restrictions of the robot. These paths will be used to guide the robot autonomously. The method will also allow generating new paths each time that substantial changes will be detected in the OGM.

REFERENCES

P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992. ISSN 0162-8828.

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

A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. In *Algorithms*, volume 2461, pages 277–288. Springer Berlin / Heidelberg, 2002.

E. Hernández, P. Ridao, A. Mallios, and M. Carreras. Occupancy grid mapping in an underwater structured environment. In *Proc. of the 8th IFAC Int. Conf. on Manoeuvring and Control of Marine Craft*, 2009a.

E. Hernández, P. Ridao, D. Ribas, and A. Mallios. Probabilistic sonar scan matching for an AUV. In *Proc. of Int. Conf. on Intelligent Robots and Systems*, 2009b.

E. Hernández, M. Carreras, J. Antich, P. Ridao, and A.Ortiz. A topologically guided path planner for an AUV using homotopy classes. In *Proc. of Int. Conf. on Robotics and Automation*, pages 2337 –2343, 2011a.

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

E. Hernández, M. Carreras, and P. Ridao. A path planning algorithm for an AUV guided with homotopy classes. In *Int. Conf. on Automated Planning and Scheduling*, 2011c.

L. Montesano, J. Mínguez, and L. Montano. Probabilistic scan matching for motion estimation in unstructured environments. In *Int. Conf. on Intelligent Robots and Systems*, pages 3499 – 3504, 2005.

J. Ng and T. Braunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent & Robotic Systems*, 50:73–84, 2007. ISSN 0921-0296.

D. Ribas, P. Ridao, J.D. Tardós, and J. Neira. Underwater SLAM in man made structured environments. *Journal of Field Robotics*, 25:898–921, 2008. ISSN 1556-4967.

E. Schmitzberger, J.L. Bouchet, M. Dufaut, D. Wolf, and R. Husson. Capture of homotopy classes with probabilistic road map. In *Int. Conf. on Intelligent Robots and Systems*, volume 3, pages 2317–2322, 2002.