The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A *Bug*-Inspired Algorithm for Efficient Anytime Path Planning

Javier Antich, Alberto Ortiz, and Javier Mínguez

***Abstract*— In recent years, *anytime* algorithms have shown to be a good solution for planning a path in domains with severe restrictions regarding the time for deliberation. They typically operate by quickly finding a highly suboptimal path first, and then improving it until the available time runs out. In this paper, we propose a novel anytime approach called *ABUG* that performs much more efficiently than the competing strategies. ABUG is based on an improved version of a member of the popular family of algorithms known as *Bug*. A formal analysis of the planner is provided and several relevant properties of ABUG are identified. Besides, as done in some heuristic-based anytime approaches, we define bounds on the quality / length of the paths returned by the algorithm. Finally, in order to demonstrate the computational savings associated with the proposal, a comparative study involving a set of well-known path-planning techniques is also carried out.**

I. INTRODUCTION

The problem of path planning is a fundamental issue in mobile robotics. In response to this interest, a huge variety of techniques has been developed along the past years. Among all of them, deterministic / heuristic-based algorithms [1] and probabilistic / sampling-based algorithms [2] are the two most popular solutions to the problem. As it is well known, when the dimensionality of the path-planning problem is low, deterministic planners are preferred because they provide bounds on the quality of the solution/s returned. On the contrary, in high-dimensional problems, probabilistic methods exhibit a much more desirable performance.

Both deterministic and probabilistic strategies comprise a useful class of algorithms named *anytime* (see [3] and [4], just to cite two examples), which is able to trade off running time and solution quality in domains where quick reactions are required. To this end, this kind of algorithms generates a succession of progressively better solutions to the problem at hand, where the time needed for computing each solution is related to its quality. In this way, the first / worst results of the planner are expected to be obtained in a very short time.

Since the reformulation in *anytime* of A* [5], a bunch of anytime algorithms have been developed to accommodate for different planning problems. A common property of these developments is that they formulate the planning problem as a graph with nodes representing a state / point in the configuration space and arcs / edges representing a feasible action to make a transition from one state to another. This gives rise to the dilemma of selecting the appropriate space resolution as a compromise among computation time, optimality, and completeness.

This study has been partially supported by project CICYT-DPI2008-06548-C03-02 and FEDER funds

J. Antich and A. Ortiz are with the Dept. of Maths and Computer Science, University of the Balearic Islands, Spain `javi.antich@uib.es`

J. Mínguez is with the Dept. of Computer Science and Systems Engineering, University of Zaragoza, Spain `jminguez@unizar.es`

This paper puts forward a new anytime path-planning strategy for indoor environments called *ABUG* that also relies on a graph representation but where each edge involves a sequence of actions corresponding to the way of acting of a novel navigation strategy which comes from the well-known family of algorithms *Bug* (see [6] for a recent compilation and comparison of the more representative Bug-like algorithms). ABUG is constrained to be applied to two-dimensional problems with only obstacles and free space (as the large majority of problems in indoor robotics). However, the advantages are that this new technique is complete, much less dependent of the resolution of the space, and largely improves the running time of previous anytime algorithms in the class of problem at hand.

The rest of the document is organized as follows: section II describes a Bug-like algorithm named *Bug2+*, while section III shows how Bug2+ can be used to construct an efficient anytime path planner; section IV presents some experimental results and, finally, section V concludes the paper.

II. THE ALGORITHM *BUG2+*

The anytime approach being proposed plans paths based on *Bug2+*, an enhanced version —suggested by the authors— of an algorithm named *Bug2* which was put forward by Lumelsky and Stepanov in [7]. This new *Bug*-derivative strategy preserves the simplicity as well as the intuitive behavioral description by which the algorithm Bug2 is mostly characterized. Furthermore, the length of a path produced by Bug2+ is always less or equal to the one of Bug2. The strategy Bug2+ is a sensor-based path planner with proven termination conditions for environments which are static, unknown, and two-dimensional. In the following, the proposal is briefly discussed underlining its major points of discrepancy with regard to the classic version of the algorithm Bug2. See [8] for a deeper explanation of Bug2+ as well as the formal proofs for the above-mentioned features of the strategy.

A. Notation

$S, T \in \mathbb{R}^2$ are, respectively, the starting and the target points of the mission. XY $(X, Y \in \mathbb{R}^2$ and $X \neq Y)$ represents the straight-line segment with end points X and Y. The line connecting the starting and the target points, ST, is referred to as *main line*, or *m-line* in brief. On the other hand, O_i denotes a certain obstacle of the environment and ∂O_i its contour curve. Finally, $d(X, Y)$ is a function which measures the Euclidean distance between any two points X and Y $(X, Y \in \mathbb{R}^2)$.

B. Description

The algorithm Bug2+ exhibits two different behaviors: motion-to-goal and boundary-following. During the former,

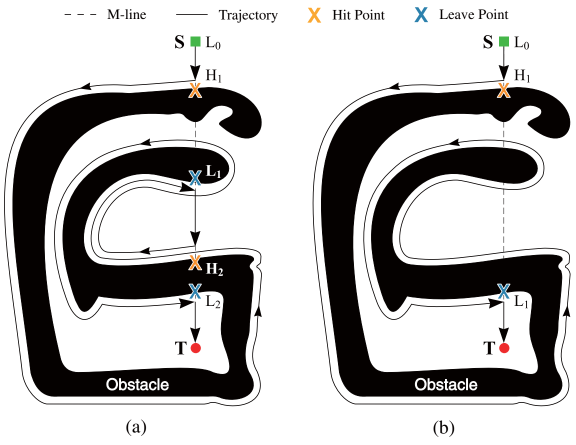


Fig. 1. Comparison of the paths generated by (a) Bug2 and (b) Bug2+ in a scenario with an intricate obstacle. In both cases, the parameter *pCFD* was assumed to be *right*.

which is activated first, the robot moves towards the target (T) along the m-line[1]. The boundary-following behavior, on the other hand, is invoked when the robot encounters an obstacle (O_i) on its way. The point where this obstacle is found is called hit point (H_j). Next, the robot follows the contour of the obstacle (∂O_i) to the left or right according to a user-definable parameter named *pCFD*. During this contour following process, a special situation may occur, in which the robot returns to H_j meaning that a loop around the obstacle boundary has been completed. In such a case, the target is inside the obstacle, not being thus achievable. More usual is, however, the situation where the robot reaches a new point on the m-line closer to T than H_j. At that moment, a leave point (L_j) is defined and the motion-to-goal behavior is invoked again.

Algorithm 1 provides a formal description of Bug2+, highlighting, in **bold**, the most important changes which have been done regarding the strategy Bug2. In short, the algorithms Bug2 and Bug2+ use a different criterion to invoke the motion-to-goal behavior when the robot is circumnavigating the contour of an obstacle. Specifically, in Bug2, this transition occurs when a point Q on the m-line nearer the target than H_j is found. Moreover, for really abandoning the boundary-following behavior, the point Q should satisfy condition C_2 as well, which requires the robot to be able to move along the straight-line segment QT without immediately hitting the current obstacle. The strategy Bug2+ differs from Bug2 in considering the points which do not meet condition C_2 into the decision associated with leaving the contour following process. Let Γ denote the set of m-line's points not satisfying condition C_2 which have been found by the robot during the last —and still current— activation of the boundary-following behavior. Then, the algorithm Bug2+ will perform a transition to the motion-to-goal behavior when reaching a point Q on ST with $Q \notin \Gamma$ and satisfying the inequality $d(Q,T) < min\{d(\gamma,T)\ \forall\gamma \in \Gamma\}$ (observe that $H_j \in \Gamma$ according to

[1]The mobile robot is supposed to be a point fitted with a complete set of error-free tactile sensors.

Alg. 1 *Bug2+*: An improvement of the strategy Bug2

```
0) Set $j = 1$ and $L_0 = S$
1) Move along the straight-line segment $L_{j-1}T$ until one of the following occurs:
    a) $T$ is reached. The algorithm stops
    b) An obstacle $O_i$ is found. Define the hit point $H_j$, **set $\boldsymbol{D = d(H_j, T)}$** and, finally, go to step 2
2) Follow the contour of the obstacle ($\partial O_i$) to the left or right according to pCFD until one of the next three possible situations arises:
    a) $T$ is reached. The algorithm stops
    b) The robot returns to $H_j$. The algorithm stops because the target is unreachable
    c) The robot gets to a point $Q$ satisfying condition $C_1$. As a result, either action $A_1$ or action $A_2$ is taken depending on whether condition $C_2$ is satisfied ($A_1$) or not ($A_2$)

    $C_1$: $Q$ is a point on the m-line ($Q \in ST$) such that $\boldsymbol{d(Q,T) < D}$
    $C_2$: the straight-line segment $QT$ does not cross the obstacle $O_i$ at point $Q^*$
    $A_1$: define the leave point $L_j = Q$, set $j = j + 1$ and, lastly, go to step 1
    $A_2$: **update $\boldsymbol{D}$ $\big(= \boldsymbol{d(Q,T)}\big)$** and continue in step 2

    * The straight-line segment $QT$ is considered to cross the obstacle $O_i$ at point $Q$ when a segment of $QT$ lies inside $O_i$ in the vicinity of $Q$
```

the definition of hit point). Fig. 1 illustrates with an example the differences between the strategies Bug2 and Bug2+.

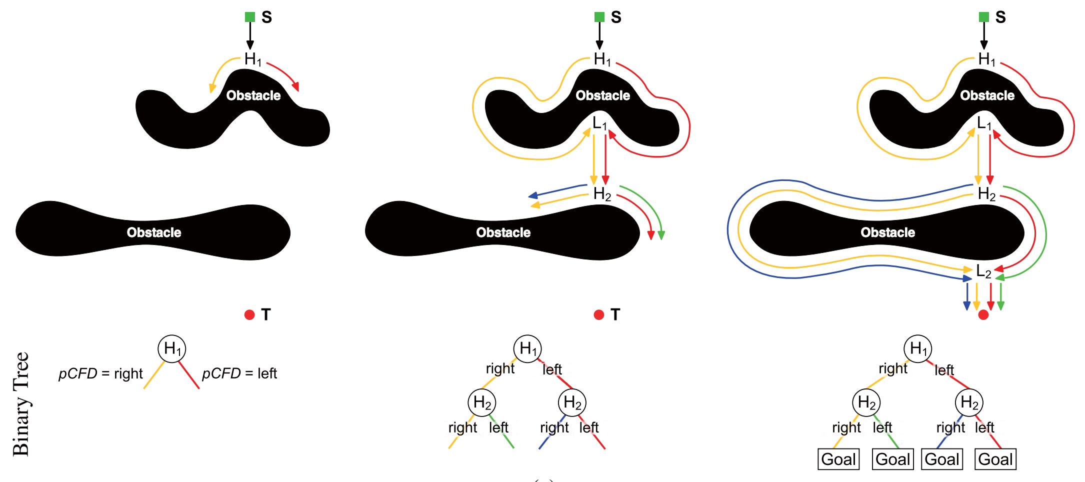
The length of a path planned by Bug2+ never exceeds the limit given by expression 1, where i denotes an obstacle of the scene (O_i), n_i represents the number of intersections of ∂O_i with the m-line, and B_i refers to the O_i's perimeter.

$$d(S,T) + \sum_i \frac{n_i}{2} B_i \tag{1}$$

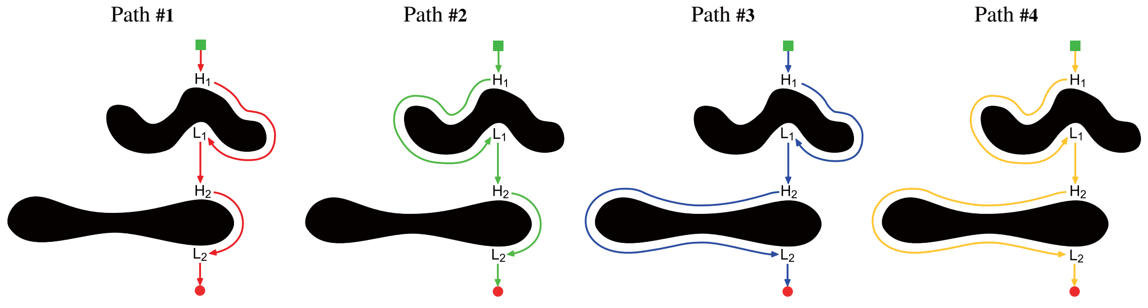
III. THE ALGORITHM *ABUG*

The algorithm Bug2+, as was described in section II, allows us to plan a single path in an unknown and static environment. ABUG uses Bug2+ to generate multiple paths in an a priori known scenario just by considering both alternatives, left and right, whenever an obstacle is found, instead of keeping the parameter *pCFD* constant. Such a flexibility in the strategy Bug2+ does not jeopardize its convergence nor the rest of its properties. Fig. 2 illustrates the above-mentioned exhaustive search in a simple scenario where four topologically different paths are planned.

Next, a deeper description of ABUG embedded into an A* framework [9] is presented. Additionally, a fast mode of operation based on inflating the heuristic cost function of the A* search is also put forward. Finally, the theoretical properties of the proposal are set out.



(a)



(b)

Fig. 2. Exemplifying how the algorithm Bug2+ can be used for planning multiple paths: (a) the search step by step; (b) planned paths.

A. Description of the Planner

As can be observed at the bottom of fig. 2(a), the algorithm ABUG makes use of a binary tree to search for paths, where each node represents a point of the environment in which a decision must be made regarding the direction —left or right— to be taken during the subsequent contour following process. On the other hand, taking into account both that a binary tree is a particular case of a graph and that A* is a well-known and efficient method for exploring graphs, ABUG adopts A* for conducting the search of all Bug2+-compliant solutions. To this end, we define an estimated cost function f that returns as output the sum of two values: g and h. Specifically, given a still incomplete Bug2+ path P, g denotes the current P length and h the expected additional distance to be traveled until achieving the target (T). This distance is assumed to be the Euclidean, which means that ABUG applies an optimistic/admissible —and also consistent [10]— heuristic h, ensuring thus the optimality of the planner (or in other words, the shortest path within the graph will be found). Nodes/Paths are expanded /extended in the order of increasing f-values by means of a priority queue. The search starts with a degenerated path merely containing the starting point (S). This path is later prolonged on the basis of step 1 in algorithm 1, which involves moving straight towards T until finding an obstacle. At that moment, the resultant path is duplicated and, next, both are extended by following the boundary of the obstacle in opposite directions. Finally, once for a path the contour following process has finished (step 2 in algorithm 1), the path is placed into the aforementioned priority queue —$qPrio$— in a position in accordance to its updated f-value. The strategy continues by taking out from $qPrio$ the estimated least-cost solution —the one with the minimum f-value— as well as by applying on it the preceding actions. Algorithm 2 describes formally the approach.

It is important to note that ABUG is an anytime approach and not a classic planner that generates the best/shortest Bug2+ path as could be guessed so far. Anytime path planning, as pointed out in section I, requires the progressive improvement of the quality of the solutions while time permits and the optimal path is not found. With this purpose, the algorithm ABUG makes use of the mathematical/topological concept of *path homotopy*, which provides us with an efficient way for optimizing a given path by solving the so-called *shortest homotopic path* problem [11]. Informally speaking, a path is regarded as an elastic band joining points S and T which is tightened to shorten it (see fig. 3 for an example). Many methods have been proposed to compute the shortest homotopic path in $\mathbb{R}^2$. Among all of them, the one published in [12] has been finally applied because it presents the minimum —to the best authors' knowledge— algorithmic complexity $\big(O(log^2\ n')$ per output vertex being n' the number of obstacles in the environment$\big)$.

The search performed by ABUG does not stop after finding and, later, improving the best Bug2+ path. The strategy actually considers all the solutions within the graph induced

Alg. 2 *ABUG*: A description in pseudocode

```
f(P)
 1: return  $g(P) + h(P)$
Store(P)
 2: Declare/Initialize the static variable $P_{Shortest}$ to NULL
 3: if $P_{Shortest}$ == NULL or $g(P) < g(P_{Shortest})$ then
 4:     Set $P_{Shortest} = P$
 5:     output ← $P$
 6:     {$P$ becomes a new solution for the anytime planner
        which continues looking for shorter paths}
 7: end if
Improve(P)
 8: $P_{Improved} = ShortestHomotopicPath(P)$
 9: $Store(P_{Improved})$
Extend(P, qPrio)
10: $Motion\text{-}To\text{-}Goal(P)$
11: if $T$ has been reached then
12:     $Store(P)$; $Improve(P)$
13: else {An obstacle has been found}
14:     for $pCFD$ = left to right do
15:         Copy $P$ into $P_{New}$
16:         $Boundary\text{-}Following(P_{New}, pCFD)$
17:         if $T$ has been reached then
18:             $Store(P_{New})$; $Improve(P_{New})$
19:         else if $T$ is unreachable then
20:             Stop search
21:         else {Conditions $C_1$ and $C_2$ have been met at $Q$}
22:             Insert $P_{New}$ into $qPrio$
23:         end if
24:     end for
25: end if
Main()
26: Build a path $P$ consisting of only the starting point ($S$)
27: Insert $P$ into $qPrio$
28: repeat
29:     Pick $P_{Best}$ from $qPrio$ such that $f(P_{Best}) \leq f(P)$,
        $\forall P \in qPrio$
30:     Remove $P_{Best}$ from $qPrio$
31:     $Extend(P_{Best}, qPrio)$
32: until $qPrio$ is empty or $T$ is known to be unreachable
```

by the binary tree data structure which is built. As will be seen in section III-C, the number of solutions to be considered is bounded by a value that increases according to the complexity of the mission, although it grows in a reasonable way. The main reason for exploring the whole space of solutions is due to the fact that worse Bug2+ paths can become better solutions after being optimized as it so happens in the scenario of fig. 4. Finally, observe that, each time a path is either computed or optimized by the algorithm ABUG, the new solution is compared with the best/shortest path found so far, and only if the former improves the latter, the strategy provides as output the new solution —i.e. line 5 in algorithm 2 is executed. Otherwise,

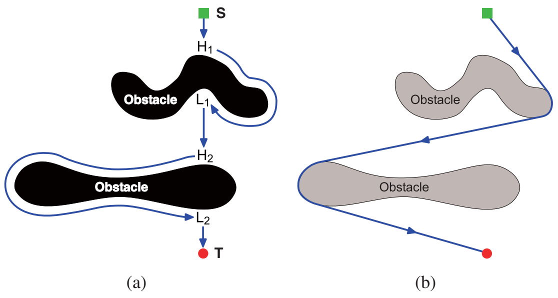


Fig. 3. The shortest homotopic path problem: (a) a path; (b) the shortest path preserving the homotopy class of a).

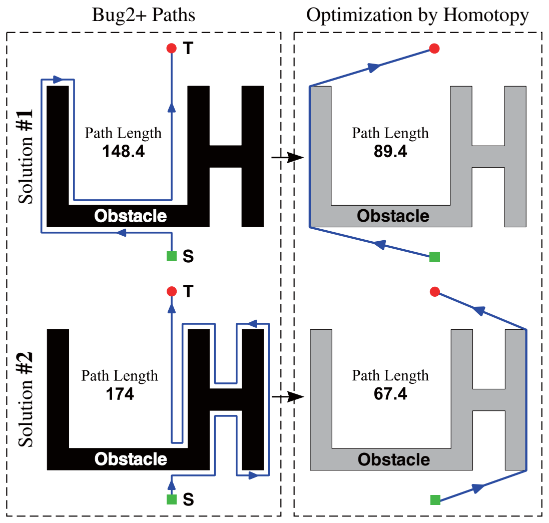


Fig. 4. An example where the worst Bug2+ path (#2) turns into the optimal solution after optimization.

the path is simply discarded. By applying such a filtering process, ABUG guarantees the generation of a strictly monotonically decreasing sequence of solutions regarding path length.

By way of notation, from now on, each of the Bug2+-compliant paths found by the strategy ABUG before being optimized will be denoted as π_k, where the index k is a sequence number that indicates the order in which the path was obtained. On the other hand, let $\Pi = \{\pi_1, \ldots, \pi_q\}$[2] be a set containing all these solutions. Notice that, as a direct consequence of the heuristic defined by the planner, π_k is shorten or equal than π_l if $k < l$.

B. Accelerated Mode of Operation of ABUG

$A^{\star}$-based anytime approaches make frequent use of the fact that, in many domains, inflating the heuristic values often results in a substantial speed-up at the cost of solution optimality (see [13] for some popular examples of this kind of algorithms). Moreover, if the heuristic h employed is consistent, then, by multiplying it by an inflation factor $\epsilon > 1$, the strategy produces a solution which is ensured not to cost more than ϵ times the cost of the optimal path.

The previous idea can be exploited to provide ABUG with an accelerated mode of operation that additionally gives

[2]Π is the empty set when there is not a solution to the path-planning problem; that is to say, when the target point (T) is not reachable.

bounds on the suboptimality of the solutions generated. To this end, a simple change is required in line 1 of algorithm 2, which consists in inflating the heuristics by ϵ as discussed before, so that the final expression for the cost function is, therefore, $f(P) = g(P) + \epsilon \cdot h(P)$. Any finite real value larger than or equal to one can be assigned to the inflation factor ϵ, which constitutes the first and only user-definable parameter of our approach. By setting $\epsilon = 1$, the strategy ABUG adopts its original and non-inflated form. In such a case, a series of paths $\Pi = \{\pi_1, \ldots, \pi_q\}$ increasingly sorted by length is progressively found and improved. On the other hand, for $\epsilon > 1$, the same Π paths as before are computed but the sequence in which they are obtained does not apparently obey to any ordering (in section III-C —fourth property—, some restrictions will be imposed with regard to such a sequencing).

C. Theoretical Properties of ABUG

The most important theoretical properties of ABUG are enumerated next. Some additional notation is, nevertheless, introduced first.

1) Notation: As was already said, $\Pi = \{\pi_1, \ldots, \pi_q\}$ represents the set of non-optimized solutions found by ABUG to the path-planning problem at hand. The approach improves each solution $\pi \in \Pi$ by computing the shortest path for the homotopy class to which π belongs. Consequently, let $\Pi^* = \{\pi_1^*, \ldots, \pi_q^*\}$ be the resultant set of improved solutions $\big(\pi_k^* = \text{SHP}(\pi_k)\ \forall \pi_k \in \Pi$, where SHP refers to the shortest homotopic path function$\big)$. On the other hand, the cost —Euclidean length— of paths $\pi_k \in \Pi$ and $\pi_k^* \in \Pi^*$ is given by, respectively, σ_k and σ_k^*. This results in two new sets: $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ and $\sigma^* = \{\sigma_1^*, \ldots, \sigma_q^*\}$. Additionally, σ_{best} and σ_{best}^* are used to designate the length of the shortest paths in Π and Π^* $\big($or in other words, $\sigma_{best} = min\{\sigma\}$ and $\sigma_{best}^* = min\{\sigma^*\}\big)$. Following the same terminology, $\sigma_{best,k}$ denotes the minimum cost for a subset of the solutions of Π. More precisely, $\sigma_{best,k} = min\{\sigma_l \in \sigma \mid l \geq k\}$ (observe that $\sigma_{best,1} = \sigma_{best}$ as a particular case of the formulation). Finally, to conclude, π_{opt} symbolizes the optimal solution to the path-planning problem, and σ_{opt} its cost/length.

2) Properties:

$p1$. $\forall \pi_k, \pi_l \in \Pi$ such that $k \neq l$, $\pi_k \neq \pi_l$.

$p2$. $\forall \sigma_k \in \sigma$, σ_k is bounded by expression 1.

$p3$. The maximum number of paths found by ABUG never goes above the limit

$$|\,\Pi\,| \leq 2^{\frac{n}{2}} \tag{2}$$

where n denotes the number of intersections between the m-line and the boundary of the obstacles in the environment $\big($i.e. $n = \sum_i n_i\big)$.

$p4$. When performing ABUG with an inflation factor $\epsilon = 1$, $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ becomes a totally ordered set under the relation $\leq$ ($\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_q$). On the other hand, if $\epsilon > 1$, the following holds: assuming that the execution of the algorithm ABUG is in a state where k paths $\pi_1, \ldots, \pi_k$ have been computed[3], the length of the next path to be obtained π_{k+1} is bounded by inequality 3. As can be observed, the strategy produces a solution which is guaranteed not to have an additional cost on the best of the paths that remain to be found ($\sigma_{best,k+1}$) of more than $\epsilon - 1$ times the cost of moving from S to T by following a straight-line path $\big(d(S,T)\big)$. In this way, we can provide bounds on the suboptimality of the solutions generated by ABUG when applying the accelerated mode of operation described in section III-B.

$$\sigma_{best,k+1} \leq \sigma_{k+1} \leq \sigma_{best,k+1} + (\epsilon - 1) \cdot d(S,T) \tag{3}$$

$p5$. Some scenarios, such as the one of fig. 5(a), can be constructed where σ_{best}^* and σ_{opt} do not match each other $\big(\sigma_{best}^* > \sigma_{opt}\big)$, which means that the strategy ABUG does not always end up yielding the optimal path π_{opt}. However, there is a particular class of problems where π_{opt} is guaranteed to be in the set of —improved— solutions computed by our approach $\big(\sigma_{best}^* = \sigma_{opt}\big)$. First of all, assume an environment composed of obstacles of generic shape meeting the requirements imposed by the Jordan Curve Theorem [11] (the contour of each obstacle ∂O_i defines a simple closed curve). On the other hand, let χ denote the minimal convex subset of $\mathbb{R}^2$ containing S, T, and the contour curve points of those obstacles which intersect the m-line in only two locations $\Big($i.e. $\partial\chi = H_{convex}\Big(\{\text{S, T}\} \bigcup \big(\bigcup_i \partial O_i \mid n_i = 2\big)\Big)$, where ∂ means *the boundary of* and H_{convex} represents the geometric concept of convex hull$\Big)$. Then, if equation 4 holds —see fig. 5(b) for a case where it happens—, the algorithm ABUG can be formally proved to find π_{opt}, or in other words, the set of solutions given by the strategy converges to the optimal value when having enough time for deliberation.

$$\chi \bigcap \left(\bigcup_i O_i \quad | \quad n_i \neq 2 \right) = \emptyset \tag{4}$$

[3] $k \in \{0, \ldots, q-1\}$. In case $k = 0$, the set $\{\pi_1, \ldots, \pi_k\}$ is supposed to be empty.

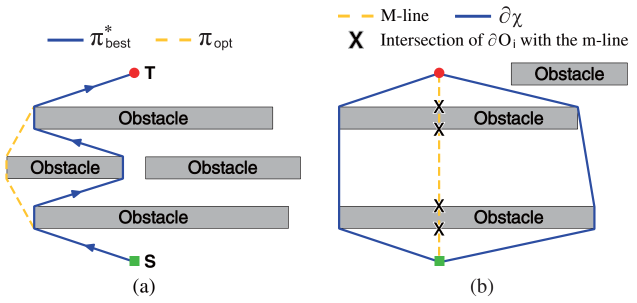


Fig. 5. About the optimality of our planner: (a) comparing the best solution computed by ABUG (π^*_{best}) with the global optimal path (π_{opt}); (b) illustration of the conditions required for optimality.

IV. EXPERIMENTAL RESULTS

This section compares *ABUG* with other competing approaches. Some of the most popular path-planning techniques have been included into the comparison by choosing from each a representative member. More precisely, the planners considered are: *Bug2+* [8], *NF1* [14], *ARA** [3][4], and *RRT* [15][5]. On the one hand, the planner Bug2+ represents the non-anytime version of the algorithm ABUG. On the other hand, the strategy NF1 constitutes the simplest and more efficient way of building an artificial potential function with its only minimum located at the target point T. By applying a wave-propagation technique and a gradient-descent method, NF1 computes the shortest collision-free path from S to T. Alternatively, ARA* is a heuristic-based anytime strategy which operates by executing a series of $A^\star$ searches with decreasing inflated heuristics. The approach provides suboptimality bounds for each successive search whose solution is guaranteed not to cost more than ϵ times the cost of the optimal path. ARA* gains efficiency by making each $A^\star$ search reuse the results of the previous search iterations. Finally, regarding probabilistic / sampling-based planners, a goal-biased version of the algorithm RRT has also been taken into account. This randomized strategy incrementally builds a search tree that attempts to rapidly and uniformly explore the free space. RRT has shown to be extremely good in finding feasible paths but with no control on the quality of the solutions produced.

Four different scenarios are proposed to assess the performance of ABUG against each of the above-mentioned planners (see fig. 6). The first mission is intended to test the ability of the strategies to realize that the path-planning problem has no solution. It is important to note that sampling-based methods do not assume that such a situation can happen so that no performance data will be given for the algorithm RRT in mission 1. On the other hand, the second mission corresponds to a simple scenario where no obstacles are located in the environment. Beyond this simplicity, mission 3 defines a topologically complex scenario under the form of a multiply-connected maze. Finally, in mission 4, many small obstacles are strategically spread throughout the environment. This last scenario constitutes an important challenge for the algorithm ABUG because of the high number of paths / solutions which the strategy finds.

All the scenarios are represented as a grid-based map with a resolution of 5cm, and a size of 150m $\times$ 150m —9,000,000 cells— for mission 3 and 100m $\times$ 100m —4,000,000 cells— for the rest of scenarios.

As for the configuration of parameters, the more usual settings defined by their corresponding authors have been used for the strategies NF1, goal-biased RRT, and ARA* (more precisely, regarding the latter, the inflation factor has been set to $\epsilon = 3.0$, decreasing in 0.5 steps, which leads to a succession of five $A^\star$ searches). On the other hand, in case of Bug2+, a *left* value has been assigned to the parameter *pCFD*. Finally, the algorithm ABUG has been executed in its default / non-accelerated mode of operation ($\epsilon = 1.0$) in all the scenarios except for mission 4 where the value for ϵ was 3.0 to speed up ABUG when dealing with the 1024 resulting paths $\big(|\Pi|{=}|\Pi^*|{=}512$ in this troublesome mission$\big)$. Nevertheless, for comparison purposes, the results associated with the execution of ABUG in its default mode are also reported for mission 4.

[4]In [3], the strategy ARA* is favorably compared against another anytime algorithm named *Anytime A** [5]. Consequently, if the comparative study of this section demonstrates that ABUG is clearly more efficient than ARA*, we can expect ABUG to outperform Anytime A* as well.

[5]By including the strategy RRT into the comparative study, we are also considering, although only in part, the anytime version of such an algorithm named *ARRT* [4]. The *Anytime RRT* approach computes its first path / solution by growing a standard RRT without any cost considerations.

Fig. 6(a),(b),(c) and (d) present the results obtained on the four scenarios previously described. The processing times provided correspond to a PC laptop Intel Core Duo @ 1.66 GHz running Windows XP Media Center SP2. Observe that, in mission 2, just one solution —and not five— is given for the strategy ARA*. This is because the path found with $\epsilon = 3.0$ —the first and highest inflation factor for ARA* according to the proposed parameter setting— was already optimal being thus irrelevant the four remaining solutions.

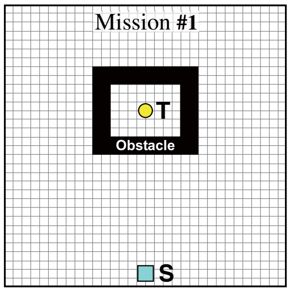
As can be observed in fig. 6(a),(b),(c) and (d), generally speaking, the strategy Bug2+ computes its first —and only— path slightly faster than ABUG. This is due to the fact that ABUG needs some extra time to look for the best ($\epsilon = 1$), or a bounded ($\epsilon > 1$), Bug2+-compliant path. However, the restricted search performed by ABUG results in a first path of higher quality than the one by Bug2+.

In comparison with the other strategies, ABUG is 63 times quicker than the best competing approach in detecting the impossibility of reaching the target in mission 1. Additionally, from mission 2 to 4, ABUG, on average, generates its first solution 9 times more rapidly and converges to the optimal path 50 times faster than the algorithms NF1, ARA*, and RRT.

V. CONCLUSIONS

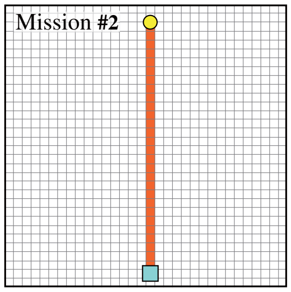
A deterministic anytime path planner inspired by a *Bug*-derivative algorithm has been formally described and, later, compared against some well-known strategies in the field, such as *NF1*, *ARA** (indirectly, *Anytime A** as well), and *RRT* (indirectly, *ARRT* as well). The performance of the approach proposed, *ABUG*, turns out to be significantly better than the one provided by the aforementioned competing planners. ABUG efficiently computes a succession of progressively better solutions or rapidly indicates failure when the given target is unreachable, which makes it well suited for planning on robots with severe computational power limitations.

ABUG has been defined for two-dimensional Euclidean configuration spaces. Nevertheless, the strategy can be extended to higher-dimensional problems maintaining both the efficiency of the algorithm and some of its more relevant properties, although at the cost of not guaranteeing convergence to the optimal solution. Such an extension, assuming a three-dimensional configuration space, could be achieved by searching for *Bug*-compliant paths in two-dimensional manifolds containing, each of them, the initial and the target configurations. This work is already finished and will be the matter of a future paper. However, bringing forward some of these results, fig. 6(e) shows a path computed by ABUG in a three-dimensional grid-based environment.



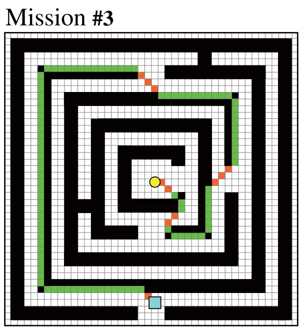
Alg.	t_U
NF1	503
RRT	—
ARA*	44,343
Bug2+	8
ABUG ($\epsilon = 1$)	8

(a)



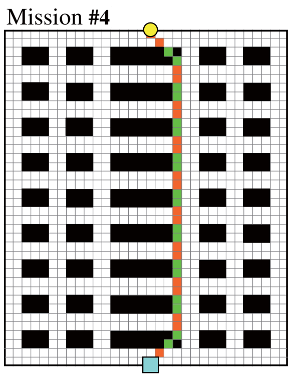
Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$
NF1	750	1.00
RRT	66	1.09
ARA*	621	1.00
Bug2+	6	1.00
ABUG ($\epsilon = 1$)	6	1.00

(b)



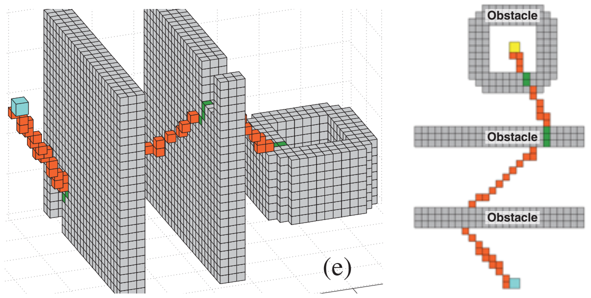
Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$	t_2	$\frac{l_2}{\sigma_{opt}}$	t_3	$\frac{l_3}{\sigma_{opt}}$	t_4	$\frac{l_4}{\sigma_{opt}}$	t_5	$\frac{l_5}{\sigma_{opt}}$
NF1	18,500	1.00								
RRT	701	1.28								
ARA*	47,687	1.01	1,083 $\times 10^3$	1.01	1,480 $\times 10^3$	1.01	2,024 $\times 10^3$	1.01	2,082 $\times 10^3$	1.00
Bug2+	34	3.79								
ABUG ($\epsilon = 1$)	59	1.61	138	1.32	399	1.00				

(c)



Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$	t_2	$\frac{l_2}{\sigma_{opt}}$	t_3	$\frac{l_3}{\sigma_{opt}}$	t_4	$\frac{l_4}{\sigma_{opt}}$	t_5	$\frac{l_5}{\sigma_{opt}}$
NF1	2,640	1.00								
RRT	72	1.24								
ARA*	6,406	1.19	6,422	1.19	6,422	1.19	44,016	1.16	430 $\times 10^3$	1.00
Bug2+	11	2.68								
ABUG ($\epsilon = 1$)	626	2.24	629	1.00						
ABUG ($\epsilon = 3$)	15	2.49	25	2.07	78	1.66	188	1.30	1,961	1.00

(d)



(e)

t_U = time elapsed (ms) until reporting that the goal is unreachable

l_i = length of the ith path

t_i = time instant (ms) in which a strategy provides its ith path (remember that, in case of ABUG, a solution is not provided for each element in the sets Π and Π^* but after finding / computing a path which is shorter than all the previous ones —worse paths are simply rejected)

Fig. 6. Experimental set-up and results: (a, b, c, d) processing times and quality of the solutions in the order that they are produced for the four scenarios considered (the shortest path generated by ABUG (π^*_{best}), if exists, is superimposed on the layout of each mission; observe that ABUG assumes that navigation is allowed along the obstacle boundaries); (e) ABUG solving a three-dimensional path-planning problem.

References

[1] J-C. Latombe. *Robot Motion Planning*. Kluwer Academic, 1991.

[2] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[3] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proc. of Advances in Neural Information Processing Systems*. MIT Press, 2003.

[4] D. Ferguson and A. Stentz. Anytime RRTs. In *Proc. of IROS*, 2006.

[5] R. Zhou and E. Hansen. Multiple sequence alignment using A*. In *Proc. of the National Conference on Artificial Intelligence*, 2002.

[6] J. Ng and T. Braunl. Performance comparison of Bug navigation algorithms. *Journal of Intelligent and Robotic Systems*, 2007.

[7] V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 1987.

[8] J. Antich, A. Ortiz, and J. Minguez. Bug2+: Details and formal proofs. Technical Report A-1, University of the Balearic Islands, 2009.

[9] P. Hart, N. Nilsson, and B. Rafael. A formal basis for the heuristic determination of minimum cost paths. *IEEE T-SSC*, 1968.

[10] J. Pearl. *Heuristics*. Addison-Wesley, 1984.

[11] W. Massey. *Algebraic Topology*. Harcourt, Brace & World, 1967.

[12] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. *Journal of Algorithms*, 2003.

[13] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *Proc. of the workshop on Planning under Uncertainty for Autonomous Systems at ICAPS*, 2005.

[14] J. Barraquand, B. Langlois, and J. Latombe. Numerical potential field techniques for robot path planning. *IEEE T-SMC*, 1992.

[15] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 2001.