G-BEAM: Graph-Based Exploration And Mapping for Autonomous Vehicles

Leonardo Cecchin, Danilo Saccani and Lorenzo Fagiano

***Abstract*— A novel solution to the problem of exploration and mapping of an unknown environment by an autonomous vehicle is presented. A hierarchical control system is adopted: a low-level reactive controller manages obstacle avoidance, and two high-level strategies are in charge of mapping and navigation tasks. The decision strategy implemented at the high-level is named G-BEAM, standing for "Graph-Based Exploration And Mapping". It builds a reachability graph used both as a trajectory planning tool and as a map. The reachability graph representation requires less storage resources with respect to a more traditional occupancy-map, and it can be directly exploited to compute the system's path towards a given target or unexplored locations. The latter are ranked according to the expected information gain that is realized when they are visited. Such information gain is then used in the cost function of the navigation strategy, which is based on a receding horizon concept. The controller has been successfully tested in various simulated environments. Comparison with other approaches in state of the art shows promising performance.**

I. Introduction

In recent years the use of unmanned vehicles, such as marine, ground, and aerial ones has been increasing significantly and steadily [1]. A traditional but still very active topic of research pertains to fully autonomous operation in environments that are partially or entirely unknown a-priori [2]. In such a framework, the autonomous vehicle must achieve three goals: safety (moving in the environment without colliding with obstacles), mapping (building a model of the environment, containing the information on accessible and blocked areas), and navigation (planning a feasible path from the current position to a reachable one in the environment).

Achieving these goals is challenging, also considering that the controller must run in real-time, limiting the complexity of the numerical computations used to model the environment and take action. Safety is usually fulfilled by reactive obstacle avoidance algorithms, taking into account real-time available information provided by sensors such as LiDAR (Light Detection And Ranging) devices, 2D or 3D vision cameras, etc., see e.g. [3]–[8].

When the environment is known, safety and navigation can be achieved together through properly designed path planning algorithms. These can be based on model predictive approaches, [9], Dynamic Visibility Graphs, [10], and graph search approaches, [11]. The environment can be represented in different ways, with Voronoi diagrams, generalized cones, occupancy grids, or graphs. In the case of partially or entirely unknown environments, however, path planning approaches cannot be directly used. Here, the second goal comes into play: mapping. The most common solutions make use of occupancy maps, see, e.g., [12], [13]. The mapping routine must be paired with an exploration one, which is typically frontier-based: it aims to detect the border between free and unexplored regions to select the next target, e.g., [14]. The frontiers can be computed by updating the ones from the previous time step with sensor readings, or by re-processing the entire map. Graph-based exploration has been proposed as well, see [15].

Despite the variety of solutions to achieve either mapping or navigation, an integrated approach that manages both of them at once appears to be not available in the literature. The lack of integration between the algorithms employed to address each goal may lead to poor efficiency and computationally expensive control routines. To address this problem, a novel approach is proposed here, named G-BEAM (Graph-Based Exploration And Mapping). The main idea is to use a reachability graph as the map of the environment, which can be used seamlessly for both exploration and navigation tasks. At the same time, a new reactive logic is proposed to enforce safety. The approach provides very competitive exploration performance at low computational cost and with high safety. The approach is tested in simulation with a detailed model of an autonomous multicopter, showing excellent results in environments of various size and complexity. The G-BEAM navigation performance is finally compared with that of state-of-the-art solutions based on randomization, demonstrating a significant improvement in terms of both computational speed and length of the computed paths.

II. System description and problem formulation

We consider an autonomous aerial vehicle, in particular a multicopter drone, equipped with a 360-degrees planar LiDAR. The drone's position is supposed to be measured accurately. The system needs to move in an initially unknown environment. The approach we propose can also be used with different types of vehicles, with relatively minor changes on the low-level control algorithms. Moreover, the work focuses on motion in a two-dimensional (2D) environment for simplicity. Extension of the approach to 3D is conceptually possible with rather limited modifications.

This research has been supported by the Italian Ministry of University and Research (MIUR) under the PRIN 2017 grant n. 201732RS94 "Systems of Tethered Multicopters". The authors are with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy. Email addresses: {name.surname}@polimi.it

A. Autonomous system model

The position of the autonomous vehicle is denoted by

$$\boldsymbol{p} = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$$

Where (x, y) are the inertial coordinates of the environment at hand. The orientation of the vehicle, ψ, is supposed to be known as well. $[\cdot]^T$ indicates the matrix transpose operation, while bold notation is used for 2D vectors. The multicopter is controlled using a common nested architecture. At the lowest level, attitude and altitude controllers manipulate the rotors' speed to generate suitable rotational moments and total thrust force to track a velocity vector reference, denoted as $\dot{\bar{\boldsymbol{p}}} = \begin{bmatrix} \dot{\bar{p}}_x & \dot{\bar{p}}_y \end{bmatrix}^T$, and angular speed reference, $\dot{\bar{\psi}}$.
Vehicle movements are limited by its maximum velocity vector $\dot{\boldsymbol{p}}_{\boldsymbol{max}}$ and acceleration vector $\ddot{\boldsymbol{p}}_{\boldsymbol{max}}$, which for simplicity are assumed to be equal in all directions.
A position controller generates the speed reference $\dot{\bar{\boldsymbol{p}}}$ in order to reach the desired position and orientation references

$$\bar{\boldsymbol{p}} = \begin{bmatrix} \bar{p}_x & \bar{p}_y \end{bmatrix}^T \quad , \bar{\psi}$$

The 360-degrees planar LiDAR sensor is assumed to be always horizontal, independently from the vehicle movements: this can be achieved for example with an active gimbal. The LiDAR produces at each time step a set of $N = \frac{2\pi}{\alpha_r}$ measurements:

$$\mathcal{S} = \{(\rho_k; \theta_k)\}_{k=1}^{N}$$

where α_r is the angular resolution of the sensor. The LiDAR is also characterized by its maximum sensing range, r_{max}. Each measurement can be expressed in polar coordinates, ρ_k, θ_k, $k = 1, \ldots, N$, with respect to the drone position, so that the absolute position of the $k-$th sensed point is:

$$\begin{aligned} x_k &= p_x + \rho_k \cos(\theta_k) \\ y_k &= p_y + \rho_k \sin(\theta_k) \end{aligned}$$

B. Environment model

The environment considered in this work can be represented by a two-dimensional map. The obstacles can be of any shape. An example of an environment considered in our tests is reported in Fig. 3.

C. Problem formulation

The problem we address is that of thoroughly exploring the unknown environment, building a map, while trying to reach an external target location, $\bar{\boldsymbol{p}}^{\boldsymbol{ext}}$, if provided. The controller must rely on the available information from the position feedback of the vehicle, $\boldsymbol{p}$, and the LiDAR sensor measurements, $\mathcal{S}$. The output of the controller is a suitable speed reference, $\dot{\bar{\boldsymbol{p}}}'$, so that the resulting behaviour satisfies safety, mapping and navigation to the target requirements. The resulting controller has to be executed in real-time onboard the autonomous vehicle. All the required computations must be carried out within the execution period of the control routine (e.g. 1 s for the high-level logic, and 10 ms for the low-level ones), also on a micro-controller with relatively low computational power.

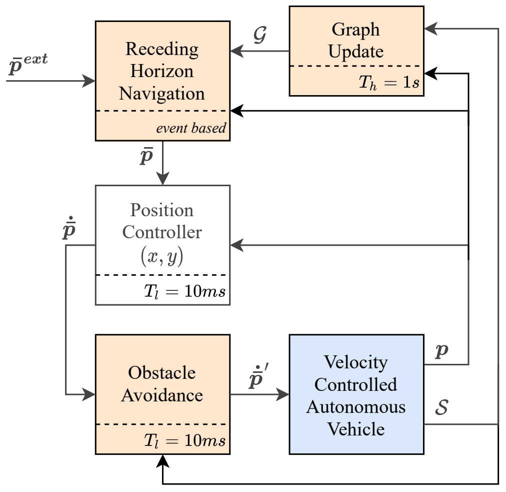

$\bar{\boldsymbol{p}}^{ext}$
Receding Horizon Navigation
event based
$\mathcal{G}$
Graph Update
$T_h = 1s$
$\bar{\boldsymbol{p}}$
$\dot{\bar{\boldsymbol{p}}}$
Position Controller (x, y)
$T_l = 10ms$
Obstacle Avoidance
$T_l = 10ms$
$\dot{\bar{\boldsymbol{p}}}'$
Velocity Controlled Autonomous Vehicle
$\boldsymbol{p}$
$\mathcal{S}$

Fig. 1. Overview of the controller layout. The highlighted blocks represent functions for which a new approach is proposed in this paper: the Graph Update procedure (Section III-B), executed with sampling period $T_h = 1$ s, the Receding Horizon Navigation technique (Section III-B), event based, and the Obstacle Avoidance controller (Section III-A), executed with sampling period $T_l = 10$ ms.

III. Proposed approach

The proposed solution entails the addition of a further hierarchical layer above the position controller and the inclusion of a low-level obstacle avoidance strategy, see Fig. 1. The obstacle avoidance technique guarantees a safe movement, while at high-level two algorithms perform mapping and navigation: the Graph Update procedure, used to add new features to the graph representing the map of the environment, and the receding horizon navigation procedure, which computes the position reference by solving a suitable optimal control problem on the updated graph.
The choice of this structure is motivated by the difference in complexity and speed of the involved algorithms. In Fig. 1 we also indicate typical sampling periods of each introduced function: for notational simplicity, in the remainder we omit the dependence of variables on the (either continuous- or discrete-) time, as this will be clear from the context.

A. Obstacle Avoidance strategy

The obstacle avoidance controller receives as input the LiDAR sensor readings, $\mathcal{S}$, and the reference speed $\dot{\bar{\boldsymbol{p}}}$. It computes a modified speed reference $\dot{\bar{\boldsymbol{p}}}'$ so that

$$\dot{\bar{p}}'_{//,k} < \lambda \left(\rho_k - \rho_{min}\right), \forall k \tag{1}$$

where ρ_{min} is the minimum allowed distance from the center of the vehicle to any object, and $\dot{\bar{p}}'_{//,k}$ is the component of the modified speed reference in the direction of the $k-$th LiDAR measurement relative to the vehicle:

$$\dot{\bar{p}}'_{//,k} = \begin{bmatrix} \cos(\theta_k) & \sin(\theta_k) \end{bmatrix} \dot{\bar{\boldsymbol{p}}}'$$

The parameter λ defines the relationship between the distance from the obstacle and the maximum allowed speed in its direction.
Note that, with this formulation, as the vehicle moves closer

to an obstacle, the reference speed component in its direction decreases, becoming null when the minimum distance is reached, preventing the vehicle from getting any closer.

For the algorithm to be effective, the modified velocity reference must always be small enough so that the vehicle can stop before reaching the minimum allowed distance from the obstacle. This property depends on the choice of λ, in particular it holds if

$$\lambda < 2\frac{\ddot{\boldsymbol{p}}_{\boldsymbol{max}}}{\dot{\boldsymbol{p}}_{\boldsymbol{max}}}.$$

B. Graph update

This controller performs mapping of the environment by creating a reachability graph and by expanding and modifying it in order to include newly acquired information, as new areas are explored. The update is carried out by a mechanism that guarantees that nodes are added only in reachable positions, and only obstacle-free arcs are added. In this approach, the graph has a threefold use: as a tool to compute obstacle-free trajectories, as a map of the environment, and as a repository of information about the borders between explored and unexplored areas. The graph update procedure is composed of two main steps.

1) Polytope generation: The first step is the generation of a convex polytope containing the current vehicle position in its interior, which under-approximates the surrounding obstacle-free area. This simplification is essential for the subsequent operations: it allows one to reduce the amount of data needed to represent the area, provides few essential points that characterize the free space, and yields fast and straightforward computations thanks to convexity.
The polytope is represented by the set of its vertices:

$$\mathcal{V} = \{1 \dots n\} \subseteq \mathbb{N} \tag{2}$$

each vertex h features a position $\boldsymbol{v_h} = \begin{bmatrix} v_{x,h} & v_{y,h} \end{bmatrix}^T$, an exploration gain value $\eta_h \in \mathbb{R}^+$ and a boolean variable γ_h indicating whether the vertex is close to an obstacle (within a selected tolerance) or not.

The polytope is then denoted with $\mathrm{Co}(\boldsymbol{V})$, indicating the convex hull of its vertices: $\boldsymbol{V} = \{\boldsymbol{v_h}\}_{h=1}^{n}$.

The polytope generation procedure begins by positioning the n vertices along the directions $\alpha_h = \frac{2\pi}{n}h + \psi$, $h = 1, \dots, n$, so that $\|\boldsymbol{v_h} - \boldsymbol{p}\|_2 = \rho_{min}$:

$$\boldsymbol{v_h} = \begin{bmatrix} p_x + \rho_{min}\cos(\alpha_h) & p_y + \rho_{min}\sin(\alpha_h) \end{bmatrix}^T \tag{3}$$

Then, the position of the vertices is iteratively modified: cycling anti-clockwise, each one is shifted further away from the vehicle by a quantity $\boldsymbol{d_h} = \delta_v \begin{bmatrix} \cos(\alpha_h) & \sin(\alpha_h) \end{bmatrix}^T$:

$$\boldsymbol{v_h} = \boldsymbol{v_h} + \boldsymbol{d_h} \tag{4}$$

If the convex hull of the new polytope contains any LiDAR reading (i.e. it interferes with an obstacle), the modification is discarded and vertex h is marked to not be moved again ($m_h = true$):

$$\exists\, k : (x_k; y_k) \in \mathrm{Co}\left(\boldsymbol{V}\right) \Rightarrow \boldsymbol{v_h} = \boldsymbol{v_h} - \boldsymbol{d_h};\ m_h = 0 \tag{5}$$

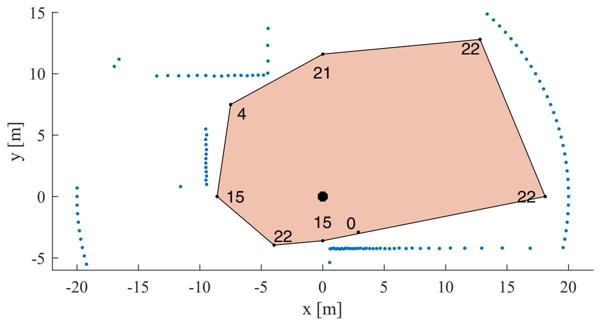


Fig. 2. Polytope generation step: example of a resulting polytope with indication of the exploration gains η_h of each vertex $h = 1, \dots, n$.

The iterations stop when $m_h = 0 \vee \|\boldsymbol{v_h} - \boldsymbol{p}\|_2 \geq r_{max}, \forall h$. Finally, the exploration gain value η_h of each vertex is computed. This value shall be proportional to the amount of additional information on the environment that could be gathered by moving to the location of that particular vertex h. To estimate such an information gain, a subset R_h of LiDAR measurements is assigned to each vertex, subdividing them in n circular sectors around the vehicle, each one centered around a vertex of the polytope:

$$R_h := \left\{ k : |\theta_k - \alpha_h| < \frac{2\pi}{n} \right\} \tag{6}$$

Once the sets are defined, the exploration gain of each vertex is computed with the following equation:

$$\eta_h := \left| \left\{ k \in R_h : \|\boldsymbol{v_k} - \boldsymbol{v_{k-1}}\|_2 \geq d_{min}^{obs} \vee \rho_k \geq r_{max} \right\} \right| \tag{7}$$

where the operator $|\mathcal{A}|$ denotes the cardinality of a countable set $\mathcal{A}$. According to equation (7), the exploration gain of each vertex is equal to the number of nearby LiDAR readings that are out of range, plus the number of subsequent measurement pairs that leave a gap large enough for the vehicle to pass through it. Fig. 2 shows an example of a polytope obtained with the proposed algorithm.

2) Graph update: The second step consists in updating the reachability graph with the latest available information. This operation shall retain all the useful information, without compromising the simplicity of the representation. The graph shall moreover represent all obstacle-free paths inside the environment, to enable navigation to every explored location, but should be at the same time composed of the least amount of elements possible, so that operations on it can be computationally efficient. The graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is composed of a set of nodes, $\mathcal{N} = \{1, \dots, n_{nodes}\} \subseteq \mathbb{N}$, and a set of edges, $\mathcal{E} \subseteq \{(i;j) : i, j \in \mathcal{N} \wedge i \neq j\}$ In a way similar to the polytope vertices (2), each node $i \in \mathcal{N}$ is characterized by its position $\boldsymbol{n_i} = \begin{bmatrix} n_{x,i} & n_{y,i} \end{bmatrix}^T$, its exploration gain value η_i, and a boolean variable γ_i, that indicates proximity to an obstacle.

The graph update is performed by adding nodes and edges if certain criteria involving the nodes $\mathcal{V}$ are met.

Each vertex $h \in \mathcal{V}$ is added as a node of the graph if the following condition is met:

$$(\gamma_h \wedge \delta_h > d_{min}) \vee \delta_h > d_{open} \tag{8}$$

Where $\delta_h = \min_{i \in \mathcal{N}}(\|\boldsymbol{v_h} - \boldsymbol{n_i}\|_2)$ i.e. it is the distance from vertex h to the nearest node of the graph.
Condition (8) checks if one of two alternatives occurs: either the vertex is close to an obstacle and at a distance $d_h > d_{min}$ from the nearest node, or it is at a distance $d_h > d_{open}$ from all other nodes. The two threshold distances d_{min} and d_{open} are used to set respectively a minimum distance among vertices near obstacles, and in open spaces, in order to influence the density of graph nodes. Examples of how these parameters affect the behaviour of the algorithm are presented in Section IV. Let us denote with $\mathcal{N}^+$ the new set of graph nodes after the described procedure has been applied to all vertices in $\mathcal{V}$. Then, new graph edges are created by connecting each pair of nodes in $\mathcal{N}^+$ that are inside the polytope $\text{Co}(\boldsymbol{V})$.

$$\mathcal{E}^+ = \mathcal{E} \cup \left\{(i;j) \mid i,j \in \mathcal{N}^+ \wedge \boldsymbol{n_i}, \boldsymbol{n_j} \in \text{Co}(\boldsymbol{V}) \wedge i \neq j\right\} \tag{9}$$

Since by construction the polytope $\text{Co}(\boldsymbol{V})$ is an under-approximation of the obstacle-free area surrounding the vehicle, the additional edges are guaranteed to be obstacle-free connections. They are created indistinctly between new nodes, new and old nodes, and also among old nodes. This ensures that the largest possible information on the explored area is gathered and stored inside the graph, and connections are added if missing.
The last operation is the update of the exploration gain values of the nodes: we set to zero that of all nodes that are at a distance $\|\boldsymbol{n_i} - \boldsymbol{p}\|_2 < \rho_{min}$ from the vehicle, or that lay in the interior of $\mathcal{A}^+$. This ensures that nodes on the borders of the explored area maintain their exploration gain value, while internal ones have zero gain. This is intuitive since very little to no information can be gained by moving to an already explored area. The nodes $\mathcal{N}^+$ and edges $\mathcal{E}^+$are finally used as starting quantities $\mathcal{N}$, $\mathcal{E}$ for the graph update at the following time step.

C. Receding Horizon Navigation

The navigation approach in G-BEAM is an event-based receding horizon one: each time the current reference node is reached, a new one is computed by planning a path of several connected nodes in the graph and culminating at a target node $t \in \mathcal{N}$. The latter is computed by maximizing a reward function that trades off the amount of information gathered and the will to reach an assigned target location (if provided).
The starting node s of the path to be planned is taken as the nearest one to the current vehicle's position:

$$s = \underset{i \in \mathcal{N}}{\text{argmin}} \|\boldsymbol{p} - \boldsymbol{n_i}\|_2 \tag{10}$$

Let us denote with $\boldsymbol{Q} = \{q_1, q_2, \ldots q_L\}$ a sequence of indexes of length L. For a given target node t, the shortest path $\boldsymbol{Q^*(t)}$ starting from s is then computed as

$$\boldsymbol{Q^*(t)} = \arg\min_{\boldsymbol{Q}} D(t) \tag{11a}$$

s.t.

$$q_i \in \mathcal{N} \; \forall i = 1, \ldots, L \tag{11b}$$

$$(q_i; q_{i+1}) \in \mathcal{E} \; \forall i = 1, \ldots, L-1 \tag{11c}$$

$$D(t) = \sum_{i=0}^{L-1} \left\|\boldsymbol{n_{q_i}} - \boldsymbol{n_{q_{i+1}}}\right\|_2 \tag{11d}$$

$$q_0 = s \tag{11e}$$

$$q_L = t \tag{11f}$$

Let us denote with $D^*(t)$ the length of path $\boldsymbol{Q^*(t)}$ (see (11d)). Then, the best target t^* is obtained by solving the following optimization problem:

$$t^* = \arg\max_{t \in \mathcal{N}} \frac{\eta_t}{\left(D^*(t) + \|\boldsymbol{n_t} - \boldsymbol{\bar{p}^{ext}}\|_2\right)^{\varepsilon}} \tag{12}$$

The reward function in (12) is designed in order to maximize the ratio between the amount of information that can be acquired by reaching the target t and the a possible length of the path to the external reference location $\boldsymbol{\bar{p}^{ext}}$, $D^*(t) + \left\|\boldsymbol{n_t} - \boldsymbol{\bar{p}^{ext}}\right\|_2$. The coefficient ε is used to adjust the relative weight between information gain and distance to be travelled. The selected path is thus $Q^*(t^*) = \{q_1^*(t^*), q_2^*(t^*), \ldots, q_L^*(t^*)\}$. In an analogy with model predictive control, the value of L can be considered as the prediction horizon. After solving problem (12), the first node in the sequence, $q_1^*(t^*)$, is used as reference for the position controller (recall Fig. 1):

$$\boldsymbol{\bar{p}} = \boldsymbol{n_{q_1^*(t^*)}} \tag{13}$$

The path planning procedure is then repeated in a receding horizon fashion, when the selected reference is reached, in particular when the following event occurs:

$$\|\boldsymbol{\bar{p}} - \boldsymbol{p}\| \leq d_{thr}, \tag{14}$$

with d_{thr} being a suitably selected threshold distance.

Finally, the exploration task is considered to be concluded when there are no more graph nodes whose exploration gain is greater than zero:

$$\eta_i = 0, \; \forall i \in \mathcal{N} \tag{15}$$

Remark 1: If an external reference position $\boldsymbol{\bar{p}^{ext}}$ is provided and it is also in the interior of the explored area, i.e. $\boldsymbol{\bar{p}^{ext}} \in \mathcal{A}$, then instead of (12) the node t^* is selected as:

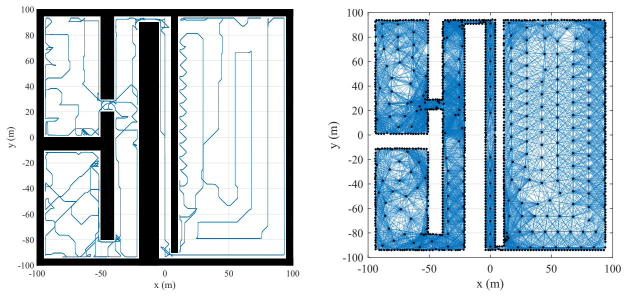
$$t^* = \arg\min_{t \in \mathcal{N}} \|\boldsymbol{n_i} - \boldsymbol{\bar{p}^{ext}}\|_2 \tag{16}$$

i.e. the closest node to the external reference position.
Regarding the computational complexity, note that problem (11) amounts to a standard shortest path computation over the graph, which can be efficiently solved, and problem (12) is an integer program with one optimization variable, which can be solved by extensive evaluation in very short time, as shown in the results presented in the next section.

TABLE I

PARAMETERS EMPLOYED IN THE SIMULATION TESTS.

d_{step}	0.5m	ρ_{min}	1.5m
d_{min}	1.5m	λ	1
d_{open}	10m	n	8
		ε	10
α_r	5°	R_{max}	20



(a) Vehicle trajectory (b) Reachability graph

Fig. 3. Exploration test results in a first environment featuring both narrow corridors and wide open areas: (a) environment and final trajectory of the drone; (b) final graph.

IV. SIMULATION RESULTS

We tested the approach in a simulation environment developed in Matlab and Simulink, using a nonlinear multicopter model [16], [17]. A ROS implementation of the approach is available at `https://github.com/leonardocecchin/gbeam_ros`. Multiple tests have been performed to evaluate all aspects of the proposed solution. The first tests assess the exploration capabilities of the controller: different environments are mapped, with different levels of detail. The other tests compare the path planning performances of the proposed approach with a very common one, these give a reference on the computational effort required to run this controller.

A. Simulation setup

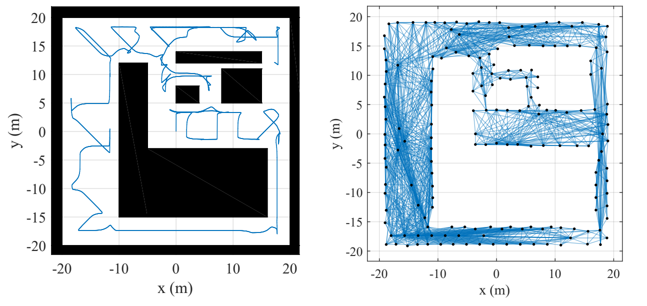
The vehicle model and the "Position controller" are executed with a cycle time of $T_s^l = 10\text{ms}$, while the high-level exploration and mapping controller is executed with a cycle time of $T_s^h = 1\text{s}$. The parameter values used in the simulation setup are listed in Table I.

The tests have been executed on a laptop equipped with an Intel core i7-10520U CPU and 16 Gb of RAM.

B. Environment exploration

In this batch of tests the vehicle started from position $(0;0)$, with no external goal provided, and its only goal was to explore and map the environment.

Fig. 3 shows the results of a first test. Fig. 3(a) shows the environment used for this test, together with the trajectory of the drone. Fig. 3(b) shows the final graph obtained by the controller. From a comparison of the graph with the model of the environment, it is possible to appreciate the ability of the former to represent all the reachable areas of the map. It is possible to see the effect of the two different distance thresholds, d_{min} and d_{open} (see (8)): thanks to them the density of nodes is much lower in open areas, whilst being higher where more detail is needed. The final graph contains 1096 nodes and 10736 edges.



(a) Vehicle trajectory (b) Reachability graph

Fig. 4. Exploration test results in a second environment featuring both narrow passages and wide corridors: (a) environment and final trajectory of the drone; (b) final graph.

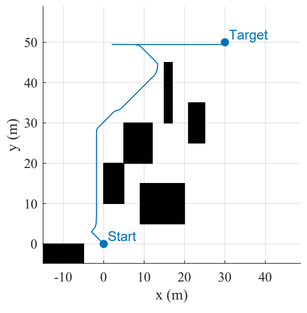
Fig. 4 shows the results of a second test. The graph shown in Fig. 4(b) is again complete, and includes all the walkable paths in the environment, without representing at all the small corridor in the top right of the map, which is too small for the vehicle to pass through. This graph is composed of 177 nodes and 1362 edges, which is much less than the previous tests, because of the smaller size of the map. By comparing the trajectory of the vehicle (Fig. 4(a)) with the graph, it can be noted that the three vertical passages around $(-2;6)$, $(6;6)$ and $(18;8)$, have been mapped, and connections across them are present, also if the vehicle never actually passed trough neither one of them. This is due to the fact that if two nodes are inside the free polytope around the vehicle, then they are connected, without the need for the vehicle to move to one of the nodes. This feature allows the vehicle to reduce the traveled distance, improving the overall efficiency. A video representing the graph being created during another test on this approach can be found in [18].
It is also worth noting that the setup for these tests was the same, and there was no need to set parameters such as map dimensions or map resolution. This is another strong point of this approach.

C. Target navigation test

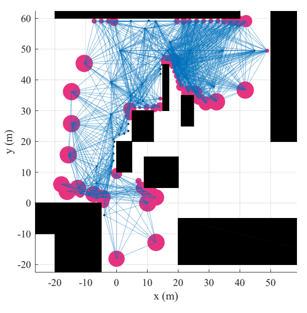
This test aims at testing the ability of the controller in guiding the vehicle through an unknown environment towards a target location, in this case the target was $(30;50)$. The results are reported in Fig. 5: the approach explores the environment moving towards the target until the latter falls in the mapped area, at which point it is reached directly by solving problem (16) in the receding horizon navigation, instead of problem (12).

D. Performance comparison with Probabilistic RoadMap

We finally compared G-BEAM to a benchmark approach using an occupancy grid mapping combined with a Probabilistic RoadMap (PRM) path planning approach, using Random Space Sampling, see [19]. The test was executed



(a) Vehicle trajectory



(b) Reachability graph

Fig. 5. Target navigation test

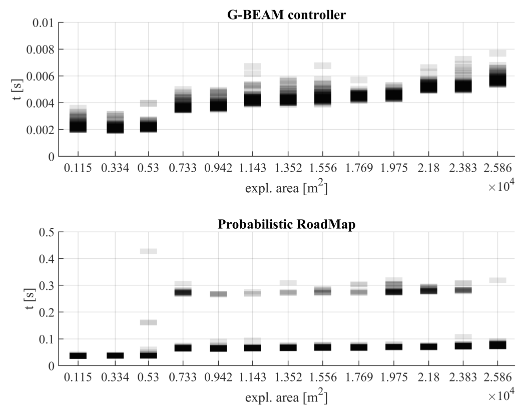


Fig. 6. Comparison between G-BEAM and an approach based on Probabilistic RoadMap (PRM) path planning.

choosing a fixed trajectory for the drone and then applying the two control strategies successively in order to create a map of the environment. Each time the explored area increases by $2000\,m^2$, the two approaches are used to compute the path towards 100 randomly selected points inside the already explored area. From this test it is possible to analyze the computing time required, as well as the resulting path length. It is also possible to evaluate the trend of the computational time with respect to the map size. Fig. 6 presents the results of this test: it is easy to appreciate the beneficial effects of the graph representation of the environment, with computational times for G-BEAM that are one order of magnitude smaller than the randomization-based approach. This result indicates that the computational effort is low enough to allow the use of G-BEAM also on low-end micro-controllers.
We also compared the length of the obtained paths towards the same target point: PRM resulted to provide on average 16% longer paths than G-BEAM.

V. Conclusions

A new, integrated technique to exploration and navigation of unknown environments by an autonomous vehicle has been proposed. The approach, named G-BEAM, features several novel ideas: a velocity-based obstacle avoidance strategy, the use of a graph as environment map, a graph update strategy based on convex polytopic under-approximations of the free space, and a receding horizon navigation strategy. The approach has been tested in many scenarios, showing good performance and outperforming state-of-the-art methods based on randomization. Next steps in this research aim to provide theoretical coverage guarantees and to test G-BEAM in real-world experiments.

References

[1] D. Goodwin, "The evolution of autonomous mobile robots." Available at `https://control.com/technical-articles/the-evolution-of-autonomous-mobile-robots/`.

[2] S. Tang and V. Kumar, "Autonomous flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 29–52, 2018.

[3] V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1069, 1990.

[4] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, no. 1-4, pp. 403–430, 1987.

[5] I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 429–435, IEEE, 1996.

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, pp. 396–404, Springer, 1986.

[7] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-d lidar," in *2015 IEEE international conference on information and automation*, pp. 1648–1653, IEEE, 2015.

[8] J.-H. Cho, D.-S. Pae, M.-T. Lim, and T.-K. Kang, "A real-time obstacle avoidance method for autonomous vehicles using an obstacle-dependent gaussian potential field," *Journal of Advanced Transportation*, vol. 2018, 2018.

[9] A. Bemporad and C. Rocchi, "Decentralized linear time-varying model predictive control of a formation of unmanned aerial vehicles," in *2011 50th IEEE conference on decision and control and European control conference*, pp. 7488–7493, IEEE, 2011.

[10] H.-P. Huang and S.-Y. Chung, "Dynamic visibility graph for path planning," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2813–2818, IEEE, 2004.

[11] J.-M. Yang, C.-M. Tseng, and P. Tseng, "Path planning on satellite images for unmanned surface vehicles," *International Journal of Naval Architecture and Ocean Engineering*, vol. 7, no. 1, pp. 87–99, 2015.

[12] G. C. Anousaki and K. J. Kyriakopoulos, "Simultaneous localization and map building of skid-steered robots," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 79–89, 2007.

[13] D.-L. Almanza-Ojeda, Y. Gomar-Vera, and M.-A. Ibarra-Manzano, "Occupancy map construction for indoor robot navigation," *Robot Control*, 2016.

[14] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151, IEEE, 1997.

[15] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "Graph-based path planning for autonomous robotic exploration in subterranean environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3105–3112, IEEE, 2019.

[16] L. Fagiano, "Systems of tethered multicopters: modeling and control design," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4610–4615, 2017.

[17] M. Bolognini and L. Fagiano, "Lidar-based navigation of tethered drone formations in an unknown environment," *arXiv preprint arXiv:2003.12981*, 2020.

[18] L. Cecchin, D. Saccani, and L. Fagiano, "G-beam: a fast exploration, mapping, and graph generation approach for autonomous vehicles." Available at `https://youtu.be/9D0L84BI0Cg`.

[19] E. Tsardoulias, A. Iliakopoulou, A. Kargakos, and L. Petrou, "A review of global path planning methods for occupancy grid maps regardless of obstacle density," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1-4, pp. 829–858, 2016.