2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561241

Fast Path Computation using Lattices in the Sensor-Space for Forest Navigation

Bernardo Martinez R. Junior

Guilherme A. S. Pereira

***Abstract*— Fast autonomous motion in cluttered and unknown environments, such as forests, is highly dependent on low-latency obstacle avoidance strategies. In this context, this paper presents a motion planning strategy that relies on lattices for the fast computation of local paths that both avoid obstacles and follow a vector field that encodes the global robot task. Lattices are constructed in the sensor space and represent a set of search trees that can be quickly pruned in function of the detected obstacles. The remaining lattice trees are used to optimize a vector field-dependent functional, thus generating the best free local path that tracks the field. To illustrate the proposed approach, we present simulation and real-world experiments of a planar robot moving in a cluttered, forest-like environment.**

I. Introduction

Real-time motion planning and navigation in cluttered and unknown environments is one of the major challenges in modern robotics [1], [2]. A trending example of this scenario is the navigation of a robot under the trees canopies of a forest, a problem that has many real world applications including timber inventory and survey [3], [4], tracking of forest wildfires [5], and pest detection [6].

However, autonomous forest navigation is a particularly hard problem due to the complexity of the environment. Forests usually have dynamic obstacles of unstructured distribution, potentially changing every time the robot visits the scenario. Because of this dynamism and the difficulty of constructing accurate maps of a forest, most previous forest navigation works assume that the forest is unknown. Some of these works have focused on finding the theoretical speed limits for the motion [7], [8]. These papers suggest that the robot motion is limited by the density of obstacles (trees and bushes) in the environment, the dynamics of the robot, and how fast the robot can perceive and plan its motion.

By considering that some of these factors are inherent to the environment or the available hardware technology, researchers now have been turning their attention to motion planning, which must quickly compute feasible paths during the flight in reaction to newly discovered obstacles. Proposed solutions are based on sampling-based techniques for path and trajectory planning [2], [9], [10], navigation fields [11], precomputed alternative paths selected in real-time [12], and the combination of several of these approaches [13].

This research is supported by the Amazon Research Awards (ARA) program and the Benjamin M. Statler Fellowship.

Authors are with the Department of Mechanical and Aerospace of the Benjamin M. Statler College of Engineering and Mineral Resources at West Virginia University, Morgantown, WV, USA. Emails: bm00002@mix.wvu.edu and guilherme.pereira@mail.wvu.edu

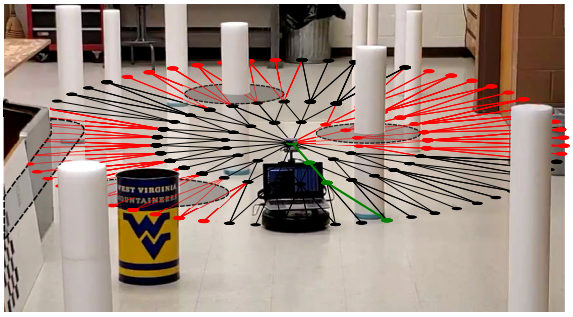


Fig. 1. Illustration of the proposed methodology. A lattice in the sensor space encodes, in a search tree, the set of paths that the robot can take locally. The optimal path is computed in real time by pruning the edges of tree that are in collision and minimizing a path cost functional on the remaining tree.

In this paper, we are considering the problem of real-time path planning in a two-dimensional Poisson forest-like environment densely populated with disk-shaped obstacles representing trees. Such an environment was previously used in [7]. The onboard sensors of the robot are limited to a planar Light Detection and Ranging (LIDAR) sensor and odometry. To solve the problem, we propose a hybrid motion planner in which the global planner, based on artificial vector fields, encodes the high-level task of the robot, and the local planner is in charge of avoiding obstacles. For the later, we propose the usage of a lattice in the sensor space. This lattice encodes a search tree, which indeed represent a set of precomputed path templates. An illustration of such a lattice is shown in Fig. 1. To allow obstacle avoidance, the lattice is quickly pruned using sensor measurements. By using a cost functional that indicates how aligned a path is from the vector field, the costs of the remaining nodes of the search tree are computed, and the current optimal path is selected. Our solution is inspired by the one proposed in [14], which used RRT* to find local optimal paths that follow the vector field. By comparing these strategies, we found that the lattice-based solution is much more efficient than its sampling-based counterpart, whose efficiency highly depends on the number of collision checks performed to construct the search tree [15]. In this paper, the search tree is constructed off-line and pruned on-line, what shown to be a much faster operation.

Thus, the main contributions of this work are: 1) a lattice-based method that optimizes a local, vector field dependent, functional for obstacle avoidance and field tracking; 2) a method for generating a lattice in the robot's sensor space; 3) a strategy that uses a precomputed map between sensor

measurements and lattice edges to speed up collision checks and tree pruning.

The rest of this paper is structured as follows: related work is presented in Sect II. The proposed methodology is described in Sect. III and evaluated in Sect. IV, where we present simulations and real-world experiments. Section V compiles the conclusions and proposals for future work.

II. Related Work

This work is based on three main pillars, namely: motion planning using vector fields, obstacle avoidance, and discretization of motion. Vector fields represent a simple way of providing the robot with a preferred direction of motion for each point in space. Vector fields can be easily computed with the use of artificial potential functions (APF) [16] or navigation functions [17]. However, using these techniques, which consider obstacles, the vector field would change every time the map of the environment changes. One strategy is then to construct the vector field ignoring the obstacles and solve obstacle avoidance in a lower level planner, as suggested by [14] and [9]. Thus, the vector field is used to encode the high-level specification of a task, which may be, for example, the periodic survey of a given curve [9]. Vector fields for curve circulation were proposed in [18].

If obstacles are ignored by the field, local motion planning is needed to avoid obstacles and follow the vector field. Three aspects affect the development of such an obstacle avoidance method [19]: the type of scenario, the robot hardware, and the avoidance technique. Two legacy obstacle avoidance approaches that use planar sensors to measure the surroundings of the robot are the Virtual Force Field (VFF) [20] and Vector Field Histogram (VFH) [21]. The former builds an occupancy grid centered on the vehicle. Repulsive forces are calculated for each cell using the same idea of APF but corrected with a weighting function. VFH is an improved version of VFF that maps obstacles to a polar graph that has peaks in directions where the obstacles are more likely and valleys in directions that are probably free. In a second step, it uses heuristics to choose which valley to follow. Similar to these methods, our work uses sensor information directly to build knowledge about free and occupied space in the surroundings of the robot.

Another important aspect of our approach resembles the methodologies introduced in [22] and [12]. In [22], precomputed relations between sensor space and planning, that were embedded in collision detection circuits (CDCs), are used to prune collision nodes in a probabilistic roadmap. In [12], precomputed alternative paths generated offline are pruned using real-time data from the robot's onboard sensor. Similarly, our method uses precomputed relations to eliminate paths in collision with obstacles. However, we chose to build a regular discretization of the sensor's field-of-view using trees (graphs without cycles) that not only eliminates paths that are in collision after pruning, but also speeds up searching for the optimal path.

To create the tree on the sensor space, we decided to use a lattice inspired by Bethe Lattices and Cayley trees [23]. Our approach is related to ego-graphs for path planning, proposed in [24], state lattices that precompute a set of actions while accounting for dynamic constraints [25], and motion primitives [26] methods. The main difference of our proposed discretization is that the spatial representation of our lattice is carefully designed to obtain the mappings between sensor measurements and the traversable paths, thus allowing for fast pruning. Further, we use our lattice to track the global behavior (vector field) by minimizing a function that measures how much the paths are aligned with the field, as is done in [14] using RRT*.

The next section will present the details of our approach.

III. Sensor Space Lattice Motion Planner

We propose a motion planner that is divided in two decoupled parts: 1) a preprocessing part that creates a lattice-based search graph, generates the mapping between range measurements and graph edges, and defines a global task; 2) an online part that prunes the graph in function of the sensor measurements, assigns costs to edges, and finds the optimal, collision-free, path in the tree. These parts are explained next.

A. Preprocessing

In this subsection, we explain how to generate a lattice in the sensor space, how to obtain the mappings between sensor measurements and the lattice, and how to generate a vector field for global planning.

1) Lattice generation: We create graph $\mathcal{G}(V, E)$ as a directed tree spatially contained in the sensor space, $\mathcal{S} \subset \mathbb{R}^2$. The tree can be constructed recursively from the root node, which is coincident with the origin of the sensor space. New vertices are created in circular layers around the root. In the first layer, the vertices are equally spaced in a circle of radius r_0 and are connected by edges coming from the root node. The number of vertices in this layer, N_T, is defined as the number of trunks of the tree, which is equivalent to the number of sub-trees that will be connected to the root. Each vertex of the first and subsequent layers branch-out to N_B children vertices in the next layers. The process of creation of vertices is repeated until the number of layers, N_L, is reached. The relationship of the layers radii is defined by the growth ratio, K. With these parameters (namely: r_0, N_T, N_B, N_L, and K), the polar coordinates $(r, \theta) \in \mathcal{S}$ of each vertex $v \in V$ of the graph can be found from the polar coordinates of its parent vertex as:

$$r_{\text{new}} = K^{(N_L - l - 1)} r_0 \tag{1}$$

$$\theta_{\text{new}} = \begin{cases} \left(\frac{2\pi}{N_T}\right)(t-1) & \text{if the parent is the root node} \\ \theta_{\text{parent}} + \left(\frac{2\pi}{N_T}\right)\left(\frac{b-(N_B+1)/2}{(N_B-1)^{(N_L-l-1)}}\right) & \text{otherwise}, \end{cases} \tag{2}$$

where $\theta_{parent} = \tan^{-1}(y_{parent}/x_{parent})$, l is the index of the l-th layer, and t is the index of the t-th trunk, and b is the index of the b-th branch of each trunk (see Fig. 2).

The proposed formulation is generic and will generate a myriad of different trees. In the results shown in this paper, we are setting the number of branches to three ($N_B = 3$)

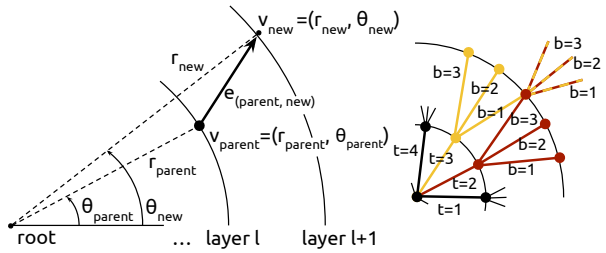


Fig. 2. Construction of the lattice adds vertices in layers around the sensor origin. Each new vertex (v_{new}) is added according to its parent vertex (v_{parent}). This process is executed recursively starting from the root node, where trunks $t = 1, 2, \ldots, N_T$ are added connecting it to the first layer of the lattice. Then, for each new layer, the leaf nodes are subdivided in branches $b = 1, 2, \ldots, N_B$. Notice that, during this process, sub-trees are created with coincident spatial representation for some of their vertices, as exemplified in red and yellow.

and the growth ratio to two ($K = 2$). By selecting these values, we guarantee that there will be twice the number of circumferential discretization points in consecutive layers and that arc lengths between successive vertices in each layer remain constant. A lattice with this particular choice of parameters is shown in Fig. 3. Notice that there will be more than one vertex in the same position, but, since they come from different trunks, they are not connected (see yellow and red sub-trees in Fig. 2). In fact, although this generates extra data in the graph representation, keeping the nodes disconnected also keeps the graph as tree, which is important for the on-line optimization we perform using the graph.

2) Sensor-to-graph mappings: After generating the lattice, it is possible to create a tessellation of the region around the origin as a mesh of non-overlapping triangles that have exactly two of their sides coincident with the edges of the graph. An example of such triangulation is shown in Fig. 4. Let $\mathcal{T}$ be the set of triangles obtained by the triangulation. By comparing the coordinates of the vertices that compose each edge and the coordinates of the triangles, it is possible to compute a map $\mathcal{R}^E_{\mathcal{T}} : \mathcal{T} \to E$ for which each edge $e \in E$ will have two triangles $t \in \mathcal{T}$ assigned to it. As the sub-trees of $\mathcal{G}$ may overlap spatially, this will give a one-to-many relation where each triangle will be linked to many edges.

Now, let $\mathcal{L}$ be the set of range measurements coming from the sensor. For planar LIDARs, it is well known that these measurements have a particular angular distribution. For example, the sensor can have a beam at each $0.5°$ around the $360°$ circumference. It is then possible to compute a map $\mathcal{R}^{\mathcal{T}}_{\mathcal{L}} : \mathcal{L} \to \mathcal{T}$ by checking which triangles can be intersected by each range measurement. Thus, each sensor range will have a list of triangles that need to be tested for collisions.

By composing $\mathcal{R}^{\mathcal{T}}_{\mathcal{L}}$ and $\mathcal{R}^E_{\mathcal{T}}$ we can quickly prune the edges of the tree that are related to the obstacles detected by the sensor. Notice that, for each range measurement, only a few well-defined edges can be pruned.

3) Definition of the global task: The lattice-shaped tree presented in the previous sub-section is limited to be inside the sensor's field-of-view. Therefore, it cannot account for the entire robot task. In the proposed methodology, we frame the global plan as an artificial vector field that is

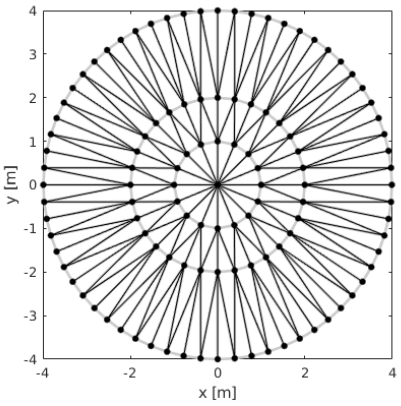


Fig. 3. A lattice with $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 1)$. In black, we see the edges that belong to the graph. In gray, the segments that together with edges from the graph form a mesh of triangles, discretizing the sensor space around the origin.

designed to follow curves in the workspace. Computation of such artificial vector fields was extensively explored in [18] and [27], where it is proposed that a vector field in two-dimensional environment can be obtained by the superposition of a normal component, $\mathbf{N}$, which makes the robot to converge to the curve to be followed, and a tangential component, $\mathbf{T}$, which drives the robot along the curve. The vector field is defined for the coordinates $(x, y)_{\mathcal{W}}$ in the world frame $\mathcal{W}$ as:

$$\mathbf{v}\big((x,y)_{\mathcal{W}}\big) = \mathbf{N}\big((x,y)_{\mathcal{W}}\big) + \mathbf{T}\big((x,y)_{\mathcal{W}}\big)\,. \tag{3}$$

With the correct choice of functions, it is possible obtain a normalized vector field for a given curve. For example, suppose a vector field to follow a straight line aligned with the x-axis of the world is wanted. This field can be obtained by taking $\mathbf{N}\big((x,y)_{\mathcal{W}}\big) = 1/\sqrt{1+f(y)^2}$ and $\mathbf{T}\big((x,y)_{\mathcal{W}}\big) = f(y)/\sqrt{1+f(y)^2}$ where $f(y) = -\arctan c\,y$, where c is a convergence parameter.

B. Online operation

By avoiding the necessity of generating the paths themselves, the motion planning problem to be executed on-line is reduced to perceiving the obstacles, pruning the tree, assigning costs to the different paths embedded in the tree, and choosing the best path to be followed. These steps are presented in this section.

1) Collision detection: Let $l_m = (x, y)_{\mathcal{S}} \in \mathcal{L}$ be the m-th range measurement coming from the LIDAR sensor and C_m be a circular region with robot radius r_R centered at $c_R = l_m + d^R_L$, with d^R_L being an offset representing the translation from the sensor frame to the robot frame (we assume that both frames have the same orientation). For each of the triangles $t \in \mathcal{T}$ obtained from $\mathcal{R}^{\mathcal{T}}_{\mathcal{L}}(l_m)$, a collision is detected if $t \cap C_m \neq \emptyset$. Computationally, this check can be done by a function that returns true if: $(c_R \cap t \neq \emptyset) \wedge (e_{t,i} \cap C_m \neq \emptyset)$ for $i = 1, 2, 3$ and false otherwise, where $e_{t,i}$ are edges of the triangle t. An illustration of this approach is shown in Fig. 4.

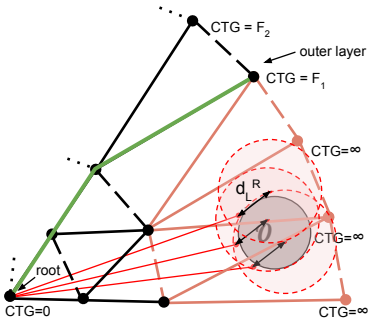


Fig. 4. Collision detection approach. In red, the range measurements are transformed into circles in the sensor space that account for the robot dimensions in the sensor frame. The triangles painted in light red overlap with these circles and are marked as untraversable. The motion planning algorithm chooses the path, shown in green, from the remaining edges that minimizes the cost-to-go (CTG) from root to the outer layer.

2) Cost assignment: A breadth-first search algorithm is used to assign the cost-to-go (CTG) for each vertex of the graph, assuming that the CTG of the root node is zero. As the algorithm progresses, the pair of triangles that are coincident with the edge ξ_i^j connecting a given vertex v_j with its parent v_i is checked for collision using the method of the previous subsection. If any of these triangles is in collision, the edge is not traversable and we assign an infinite cost to it. If there is no collision, the CTG_j of the vertex v_j is obtained by:

$$CTG_j = CTG_i + \mathcal{F}[\xi_i^j, \mathbf{v}]\,, \tag{4}$$

where CTG_i is the CTG of the parent vertex v_i. The cost $\mathcal{F}[\xi_i^j, \mathbf{v}]$ measures the co-directionality of the vector field $\mathbf{v}$ and the edge ξ_i^j. For this, we use the upstream criterion, proposed in [28] and [14] as the following functional:

$$\mathcal{F}[\xi_i^j, \mathbf{v}] = \int_0^1 \left(1 - \frac{\xi_i^{'j}(\tau)}{\|\xi_i^{'j}(\tau)\|} \cdot \frac{\mathbf{v}(\xi_i^j(\tau))}{\|\mathbf{v}(\xi_i^j(\tau))\|}\right) \|\xi_i^{'j}(\tau)\| d\tau\,, \tag{5}$$

where $\xi_i^{'j}(\tau) = \partial \xi_i^j / \partial \tau$ is the first derivative of the edge with respect to the spatial parameterization variable $\tau = (x, y)_{\mathcal{W}}$. Notice that the derivative of the edge is constant. The cost for the case in which the path is parallel to the field is zero and, for the anti-parallel case, the cost is equal to the length of the path. By using this functional, the CTG of each node in the tree will be as low as the path from the root to that node is "close" to the vector field.

3) Searching for an optimal path: After assigning costs, the vertices of $\mathcal{G}$ that belong to the outer layer are sorted by their CTG. By selecting the node with minimum cost and following the back-pointers of the tree to the root node we extract the best path of the tree. If every CTG in the outer layer is infinite, the precedent layer is searched for the minimum value. A minimum cost can always be found because this procedure can be repeated until the root node (current position of the robot), which has zero CTG. If this case robot will stop and return failure.

IV. Results

In this section, we present results from simulations and real world experiments. For the simulated environments, our goal was to validate the methodology and obtain some performance parameters regarding processing time and cost optimization. For the real world experiments, we focus on simple experiments to demonstrate the planner.

A. Simulation results

Our methodology was first implemented in MATLAB®. We have used the Navigation Toolbox implementation of a range sensor to simulate a planar robot moving in a squared forest with $120\,\mathrm{m}$ side. Trees of diameter $r_{\text{trees}} = 0.1\,\mathrm{m}$ were distributed using a Poisson distribution at different densities ρ, as proposed in [7]. The number of LIDAR beams was 1024, the sensor range was $10\,\mathrm{m}$, and the measurement error was $0.01\,\mathrm{m}$. The lattice used was the one in Fig. 3.

Once a path is computed as a sequence of nodes in the lattice, the robot followed only part of this path, which we call committed path. In our simulations, we chose the committed path to include the first node of the planned path only. The travel distance was obtained by using a reference vehicle speed of $10\,\mathrm{m/s}$ and time equivalent to the processing time needed for the path generation. We ran our simulations in an Intel® Core™ i7-4700MQ $2.40\mathrm{GHz} \times 8$ CPU. Fig. 5 shows the computed collision-free paths for 2000 iterations of the simulation for different forest densities.

Table I shows metrics regarding the planner and the path found for each density in Fig. 5. As expected, the processing time for computing the paths in each iteration was not greatly affected by tree density (ideally, it should only be dependent of the lattice parameters and the number of laser beams). In fact, larger computation times for the first lower density values is due to vector field computation inside the functional, since fewer edges are pruned when the environment is free. The table also shows that the cost of the actual path (computed using (5)) increased with the density, indicating an increasing difficulty in following the vector field as the environment gets more cluttered, as expected.

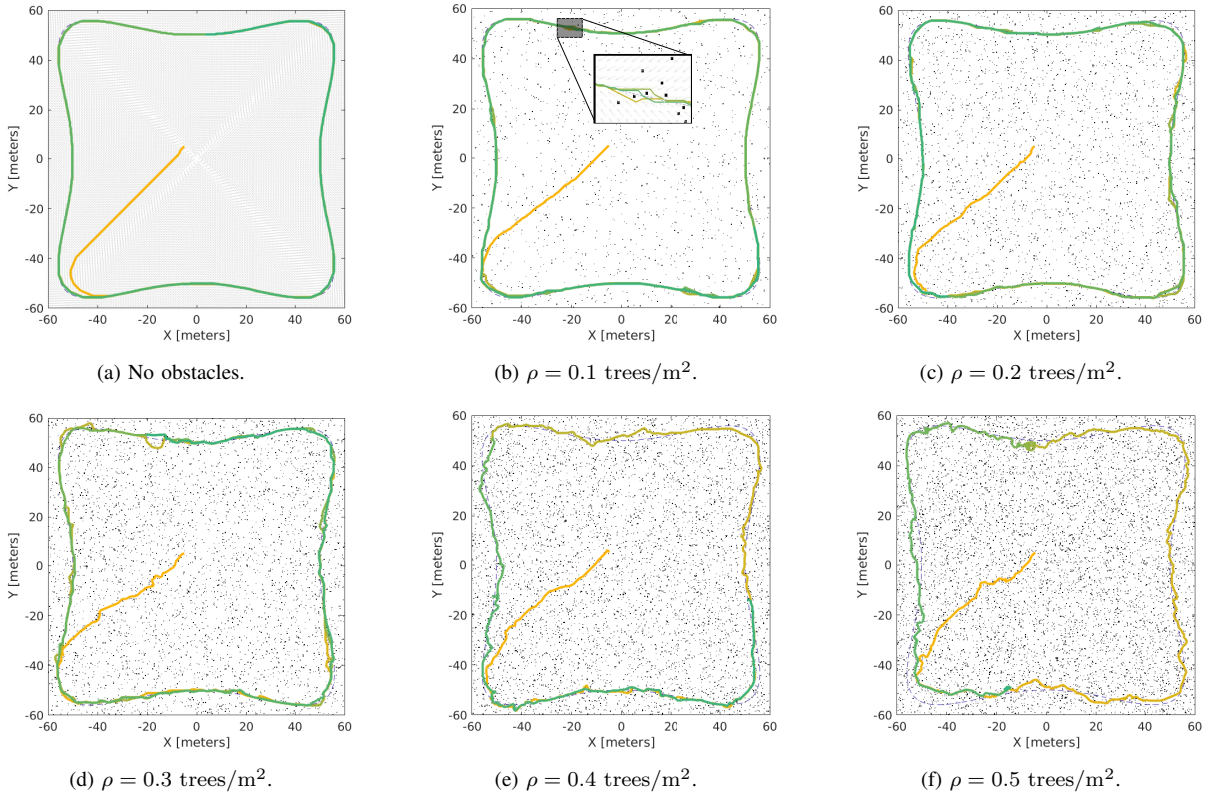
TABLE I

Simulation metrics: processing time and cost.

Tree Density [trees/m^2]	**Performance parameters**			
	Time per it. [ms]	***Total time*** [s]	***Cost per it.*** [mm]	***Total Cost*** [m]
0.0	82.1 ± 4.6	164.2	6.4 ± 5.7	12.9
0.1	76.2 ± 6.5	152.4	9.9 ± 20.9	19.8
0.2	70.8 ± 5.7	141.6	14.6 ± 29.5	29.3
0.3	66.6 ± 3.8	133.3	26.1 ± 53.1	52.2
0.4	66.0 ± 3.5	132.0	119.2 ± 118.5	238.5
0.5	65.6 ± 4.5	131.2	134.7 ± 139.3	266.4

B. Experimental results

We implemented and tested our motion planner in an iRobot's Create 2. This differential-drive robot has $0.34\,\mathrm{m}$ of diameter, a maximum speed of $0.5\,\mathrm{m/s}$ and a maximum turning radius of $2\,\mathrm{m}$. The robot was equiped with the YDLIDAR X4 360-degree planar LIDAR. This sensor has a range of $10\,\mathrm{m}$, with scan frequency up to $12\,\mathrm{Hz}$. The angular resolution is $0.50 \pm 0.02°$. The robot is controlled by a laptop with an Intel® Core™ i3 1.8GHz CPU running Linux Ubuntu 18.04 and ROS Melodic. We used the



(a) No obstacles. (b) $\rho = 0.1$ trees/m^2. (c) $\rho = 0.2$ trees/m^2.

(d) $\rho = 0.3$ trees/m^2. (e) $\rho = 0.4$ trees/m^2. (f) $\rho = 0.5$ trees/m^2.

Fig. 5. Computed paths for a simulated forest environment, with different obstacle densities. The artificial vector field used for all the simulations is shown in (a), its construction is detailed in [18]. (b) shows a detailed view of the robot paths in the three times it crosses the same region.

SLAM package `gmapping` [29] for localization, necessary to compute the vector field. We controlled the robot using a simple ROS node developed in-house. Our planner was implemented in C++ and wrapped in a ROS node.

Our tests were performed indoors using cylindrical obstacles randomly placed in our laboratory. These tree-like obstacles are supposed to simulate a cluttered forest environment. The obstacles diameters ranged between $100\,\mathrm{mm}$, and $350\,\mathrm{mm}$. In our tests, the lattice was generated using the following parameters: $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 0.4)$, which was shown to be adequate to the room and obstacle sizes. The planning range (lattice outer-layer radius) is $1.6\,\mathrm{m}$, which is smaller than the LIDAR range. Therefore, any measurement that was greater than the lattice range was disregarded in our implementation to avoid unnecessary computation of ranges that are far from the region of interest (as our laboratory walls, for example). The vector field used was the one used to follow a straight line, as exemplified in the methodology section, with convergence parameter $c = 2$.

The robot was able to navigate at its maximum speed through the forest-like environment and snapshots of the robot navigation for one of our trials are shown in Fig. 6. The path executed by the robot during the same trial and the distribution of the obstacles in the map is depicted Fig. 7. Notice that the robot tries to go back to the preferred path (straight-line) every time it can, but it is repelled by the cylinders. When there is no more obstacles (Fig. 6(f)), the robot successfully returns to the desired route.

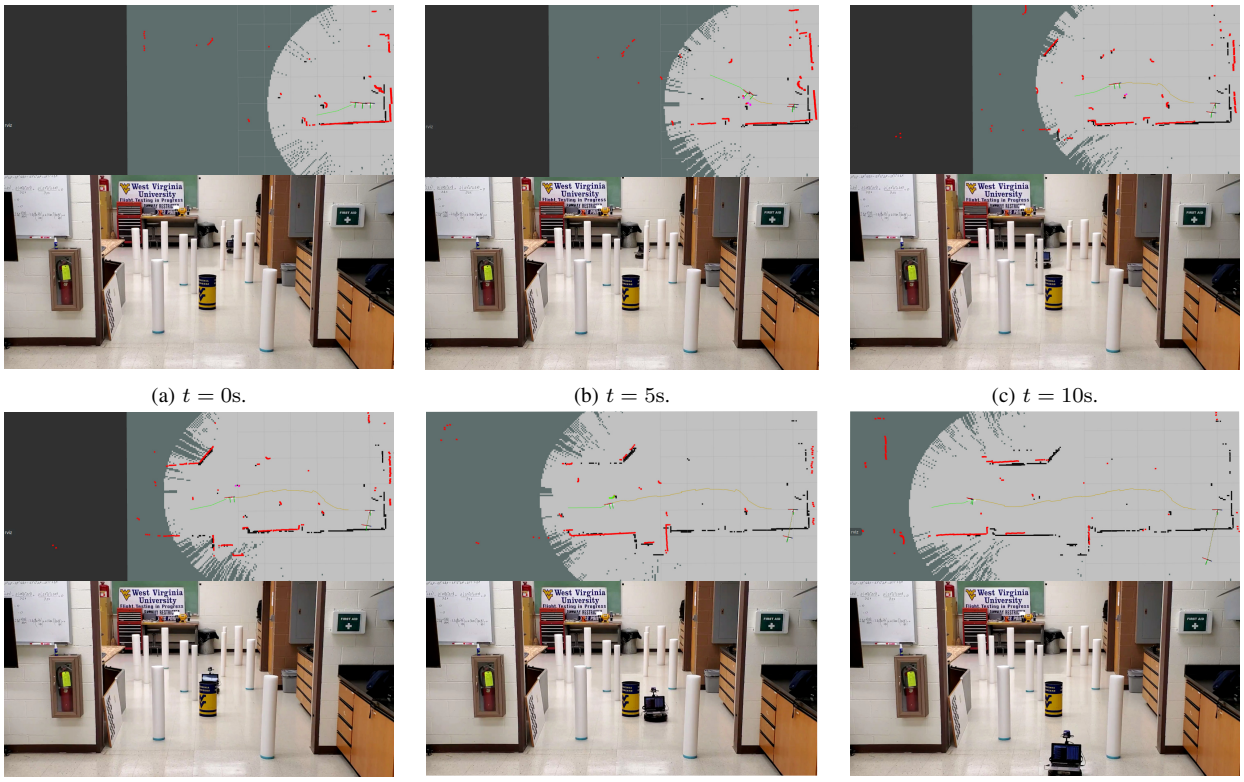
The time necessary to generate the paths compared to the sensor frequency provides an indication of the performance of our planner. While the data coming from the LIDAR is streamed at a period of $200\,\mathrm{ms}$, the paths were generated at $32.85 \pm 7.45\,\mathrm{ms}$ (collision: $30.20 \pm 7.64\,\mathrm{ms}$, cost assignment: $2.52 \pm 0.42\,\mathrm{ms}$, search: $0.12 \pm 0.01\,\mathrm{ms}$)[1].

To compare the efficiency of our method with RRT*, used in [14] to optimize the field-based functional, we run both approaches in a fixed obstacle configuration. We used the RRT* implementation provided by OMPL [30]. Since RRT* is an anytime probabilistic planner, we ran it 100 times for $30\,\mathrm{ms}$ (the time spent by our method in the real-robot experiment) and one time for $3\,\mathrm{s}$, which would represent a path very close to the optimal. In the same conditions, our method spent $21.91\,\mathrm{ms}$ to compute a path. Figure 8 shows the result of this experiment. Notice that the path computed by our approach (blue) is similar to the optimal one (dashed red), while the paths computed by RRT* with a strict deadline (shown in gray) present a large variation. This indicates that our method would be a better choice when a short time is available for path planning.

V. Conclusions and Future Work

This paper presented a path planner that uses a lattice-shaped search tree to represent a set of pre-computed local paths. This lattice, which also represents a discretization of

[1]A video with simulations and experiments can be found at: https://youtu.be/Axn7XRimgFU.



(a) $t = 0$s. (b) $t = 5$s. (c) $t = 10$s.

(d) $t = 15$s. (e) $t = 20$s. (f) $t = 27$s.

Fig. 6. Sequence of snapshots of a real robot experiment. On the top we show the constructed map, real time path planned (in green), and the laser scans (in red). On the bottom, the robot moves to the direction of the reader avoiding obstacles (white cylinders and the WVU bucket).

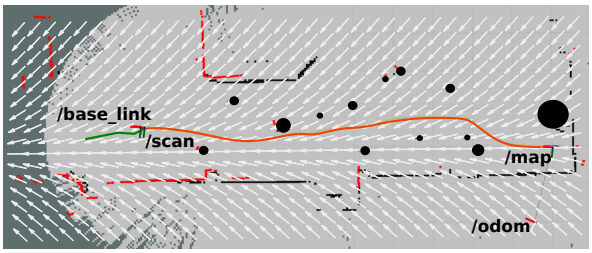


Fig. 7. Real robot experiment scenario. The vector field is given as white arrows; the path executed by the robot, as an orange line; the next planned path, as a green line; instantaneous lase scans, as red dots; obstacle, as black dots. The $/map$ frame is coincident with the starting position of the robot. The $/odom$ frame is initialized aligned with the $/map$, but it drifts during execution. SLAM computes the transformation between them. The $/scan$ frame is at the origin of the sensor space and the $/base\_link$ is at the center of the robot.

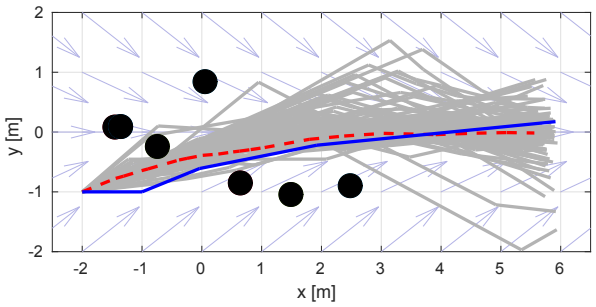


Fig. 8. Comparison with RRT*. In light blue, the vector field. In blue, the planned path using our motion planner. In red, the optimal path generated by RRT* in 3 s. In gray, paths generated by RRT* in 30 ms. Obstacles are shown in black.

the sensor's field-of-view, enables fast collision checking, elimination of untraversable paths, and cost minimization. The approach is based on the idea that pruning a search tree is less expensive computationally than constructing one, which was shown to be true in our experiments. In the simulations, the planner was tested in Poisson forest environments for a range of tree densities. In real-world experiments, the planner was tested with a differential drive robot and proved itself capable of producing good paths in approximately 30 ms for an i3 CPU, indicating it can be used for safe navigation in cluttered environments.

Future work will include expanding the workspace from 2D to 3D. Additionally, we want to take advantage of the independency of the collision checking computations and implement this motion planner using an parallelized architecture using NVidia® CUDA®. During our experiments, we noticed that homotopy of the computed path changes depending on the obstacle configuration. To avoid this problem, that could lead to chattering behaviors, we want to add another layer of complexity in our optimization functional, keeping track of the path homotopy to avoid abrupt changes. We want to test our method with faster robots, because they would probably require smoother trajectories. For that, we want to leverage the capability provided by our lattice of categorizing free and occupied space and use CHOMP [31], for example, to generate better trajectories in the free space.

REFERENCES

[1] J. Langelaan and S. Rock, "Towards autonomous UAV flight in forests," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2005, p. 5870.

[2] K. Yang and S. Sukkarieh, "3D smooth path planning for a UAV in cluttered natural environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 794–800.

[3] R. A. Chisholm, J. Cui, S. K. Lum, and B. M. Chen, "UAV LiDAR for below-canopy forest surveys," *Journal of Unmanned Vehicle Systems*, vol. 1, no. 01, pp. 61–68, 2013.

[4] C. Torresan, A. Berton, F. Carotenuto, S. F. Di Gennaro, B. Gioli, A. Matese, F. Miglietta, C. Vagnoli, A. Zaldei, and L. Wallace, "Forestry applications of UAVs in europe: A review," *International Journal of Remote Sensing*, vol. 38, no. 8-10, pp. 2427–2447, 2017.

[5] L. A. Arroyo, C. Pascual, and J. A. Manzanera, "Fire models and methods to map fuel types: the role of remote sensing," *Forest Ecology and Management*, vol. 256, no. 6, pp. 1239–1252, 2008.

[6] M. A. Wulder, C. C. Dymond, J. C. White, D. G. Leckie, and A. L. Carroll, "Surveying mountain pine beetle damage of forests: A review of remote sensing opportunities," *Forest Ecology and Management*, vol. 221, no. 1-3, pp. 27–41, 2006.

[7] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 2899–2906.

[8] S. Choudhury, S. Scherer, and J. A. Bagnell, "Theoretical limits of speed and resolution for kinodynamic planning in a poisson forest," in *Proceedings of Robotics: Science and Systems XI*, 2015.

[9] A. C. Chiella, H. N. Machado, B. O. Teixeira, and G. A. S. Pereira, "GNSS/LiDAR-Based navigation of an aerial robot in sparse forests," *Sensors*, vol. 19, no. 19, p. 4061, 2019.

[10] B. Liu, W. Feng, T. Li, C. Hu, and J. Zhang, "A variable-step RRT* path planning algorithm for quadrotors in below-canopy," *IEEE Access*, vol. 8, pp. 62 980–62 989, 2020.

[11] D. Panagou, "Motion planning and collision avoidance using navigation vector fields," in *IEEE International Conference on Robotics and Automation*, 2014, pp. 2513–2518.

[12] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-CAP: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 8456–8463.

[13] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *IEEE International Conference on Robotics and Automation*, 2019, pp. 732–738.

[14] G. A. S. Pereira, S. Choudhury, and S. Scherer, "A framework for optimal repairing of vector field-based motion plans," in *International Conference on Unmanned Aircraft Systems*, 2016, pp. 261–266.

[15] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT*," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3513–3518.

[16] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[17] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[18] V. M. Gonçalves, L. C. Pimenta, C. A. Maia, B. C. Dutra, and G. A. S. Pereira, "Vector fields for robot navigation along time-varying curves in n-dimensions," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 647–659, 2010.

[19] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[20] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[21] ——, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.

[22] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot motion planning on a chip." in *Robotics: Science and Systems*, 2016.

[23] M. Ostilli, "Cayley trees and bethe lattices: A concise analysis for mathematicians and physicists," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 12, pp. 3417–3423, 2012.

[24] A. Lacaze, Y. Moscovitz, N. DeClaris, and K. Murphy, "Path planning for autonomous vehicles driving over rough terrain," in *IEEE International Symposium on Intelligent Control*, 1998, pp. 50–55.

[25] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[26] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," *arXiv preprint arXiv:2001.04420*, 2020.

[27] V. M. Gonçalves, C. A. Maia, G. A. S. Pereira, and L. C. A. Pimenta, "Navegação de robôs utilizando curvas implícitas," *Sba: Controle & Automação Sociedade Brasileira de Automatica*, vol. 21, no. 1, pp. 43–57, 2010.

[28] I. Ko, B. Kim, and F. C. Park, "Randomized path planning on vector fields," *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1664–1682, 2014.

[29] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings., IEEE International Conference on Robotics and Automation*, 2005, pp. 2432–2437.

[30] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[31] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494.