2016 International Conference on
Unmanned Aircraft Systems (ICUAS)
June 7-10, 2016. Arlington, VA USA

A Framework for Optimal Repairing of Vector Field-based Motion Plans

Guilherme A. S. Pereira[1], Sanjiban Choudhury[2], and Sebastian Scherer[2]

***Abstract*— This paper presents a framework that integrates vector field based motion planning techniques with an optimal path planner. The main motivation for this integration is the solution of UAVs' motion planning problems that are easily and intuitively solved using vector fields, but are very difficult to be even posed as optimal motion planning problems, mainly due to the lack of clear cost functions. Examples of such problems include the ones where a goal configuration is not defined, such as circulation of curves, loitering and road following. While several vector field methodologies were proposed to solve these tasks, they are susceptible to failures in the presence of previously unmodeled obstacles, including no-fly zones specified during the flight. Our framework uses a vector field as a high level specification of a task and an optimal motion planner (in our case RRT*) as a local, on-line planner that generates paths that follow the vector field, but also consider the new obstacles encountered by the vehicle during the flight. A series of simulations illustrate and validate the proposed methodology. One of these simulations considers a rotorcraft UAV equipped with a spinning laser patrolling an urban area in the presence of unmodeled obstacles and no-fly zones.**

I. INTRODUCTION

To guide and control robots in environments with obstacles, several vector field methodologies have been proposed in the last three decades. In these methodologies, a velocity or acceleration vector is associated to the robot's free configurations such that the integrals of the resultant vector field are collision-free trajectories that, starting from any possible initial configuration, drives the robot to complete its task. The first vector field methodology was proposed by Khatib in [1], where the vector field was computed as the negated gradient of an artificial potential function with a minimum at the goal configuration. Several methodologies followed this first idea, notably the proposition of Navigation Functions [2] and Harmonic Functions [3] for potential functions without local minima. These methods are used when a robot needs to reach a goal position without colliding with the obstacles in the environment. More recently, different vector fields methodologies, which are not necessarily based on the gradient of functions, have been created to solve different and more complex tasks, such as circulation of curves [4] and tracking [5], among others. The main motivations for the use of vector field techniques are their intuitiveness, simplicity, low computational cost and, since they are closed loop methods, robustness to small localization and actuation errors. Moreover, some methods present mathematical guarantees that the tasks will be completed [5].

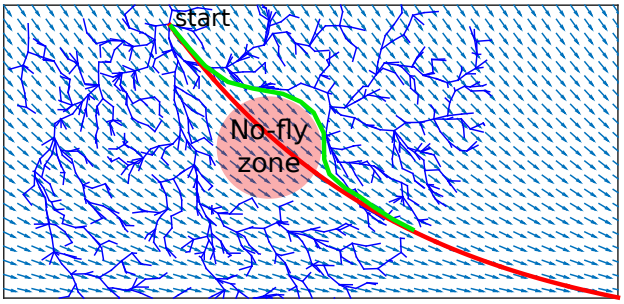

Fig. 1. A UAV uses a vector field, represented by the arrows, to navigate. Assuming that the vector field was created with partial knowledge of the problem, the red path, computed as the integral of the field, would yield a path that crosses a no-fly zone. Instead of creating a new vector field that considers the no-fly zone, an optimal motion planner, such as RRT*, may be used to locally repair the original plan. The resultant path, shown in green, avoids the no-fly zone while still following the vector field. The blue graph represents the optimal tree used to compute the path.

Unfortunately, vector field methodologies present at least two important drawbacks: (i) although the technique was originally created to deal with dynamic environments, global convergence properties are generally lost in the presence of movable or previously unknown obstacles; and (ii) they do not consider the vehicle's differential constraints. The motion planning framework proposed in this paper can be used to overcome these two drawbacks, although only (i) will be addressed here. Regarding, drawback (i), a few works have locally modified the vector field in real time to avoid dynamic obstacles and still maintain convergence properties [6], [7]. However, this modification generally does not consider the quality of the final path.

The novel idea of this paper is to combine vector fields methodologies with an optimal motion planner. More specifically, we tightly couple the optimal motion planner to a vector field by considering the vector field as a high level global plan that must be safely followed by a robot with the aid of the optimal planner. We assume that the vector field does not consider some of the details of the environment, such as small and dynamic obstacles, which usually yields simple and fast field computation. Figure 1 illustrates this idea. In this figure, the red path, obtained as the integral of the vector field, represented by the arrows, crosses a no-fly zone, which was ignored during the field computation. A sampling based optimal planner is then used to find a path,

This work was supported by CNPq/Brazil process 232587/2014-0 and NSF grant #1328930, "NRI: Fast and Accurate Infrastructure Inspection with Low-Flying Robots."

[1]G. A. S. Pereira is with Department of Electrical Engineering, Universidade Federal de Minas Gerais, Brazil. gpereira@ufmg.br

[2]S. Choudhury and S. Scherer are with the Robotics Institute, Carnegie Mellon University, USA. sanjibac@andrew.cmu.edu, basti@andrew.cmu.edu

shown in green, which follows the vector field as close as possible, but also avoids the no-fly zone.

Optimal motion planners have been the focus of attention of several researchers nowadays, since they can deliver global optimal and differential constrained trajectories computed in state spaces of large dimensions. Among these planners, a recent and efficient one is the asymptotically optimal version of the Rapidly-Exploring Random Tree (RRT), called RRT* [8]. Although RRT* has been used to solve several robotic problems, it is so far limited to compact workspaces with a well defined goal region. It is then difficult to use this tool in some tasks that include persistent monitoring of ground areas, tracking of moving targets, and navigation in urban environments, where the environment is unbounded or a goal configuration is not specified. Moreover, for this kind of task, where the workspace can be very large, the computational time of RRT* may rule it out for use in real-time operations. Solutions for these problems have been proposed by some authors, generally integrating RRT* with another strategy. One example is [9], which uses a higher level planner to define a sequence of waypoints and use RRT* to reach each waypoint in minimum time avoiding obstacles and respecting the vehicle's constraints.

Although several optimal planners can be used in the proposed framework, in this paper we use RRT* as the motion planner to be combined with a vector field. Even though RRT* was originally conceived to be a global planner, in our approach it will work as a local planner, which will make the robot to optimally avoid the obstacles and no-fly regions not considered by the vector field. RRT* will work on a bounded, small region of the workspace while the vector field will be constructed over the whole, possible unbounded, workspace. We do not assume any specific vector field methodology. Therefore, the vector field may be computed as the gradient of a potential function [1], [2], [3] but, more interesting, it can be a vector field that aims for, for example, circulation of curves for perimeter surveillance [4], corridor following [10] or multi-robot deployment [11]. These last applications are very hard to be posed as optimal motion planning problems due to the lack of clear cost functions.

The use of vector fields to direct the growth of RRT* trees is considered in [12]. The authors propose a new sampling strategy that, using the vector field, guides the search tree towards the goal, thus speeding up the method. To the best of the authors' knowledge, [12] is the only work published so far that considered both techniques in the same framework. Differently from [12], in this paper RRT* is tightly coupled with the vector field in the sense that the field is used to define a cost function and also to guide the sampling process.

After the formal definition of our problem in the next section, the proposed methodology is presented in Section III. Illustrative simulations are presented in Section IV while conclusions and future work are in Section V.

II. PROBLEM DEFINITION

Let $\mathcal{Q} \subset \mathbb{R}^n$ be the configuration space of a robot and $\mathcal{Q}_{\text{obs}} \subset \mathcal{Q}$ be an invalid set of configurations that represent obstacles and/or no-fly zones. We assume that $\mathcal{Q}_{\text{obs}} = \mathcal{Q}^k_{\text{obs}} \cup \mathcal{Q}^m_{\text{obs}}$, where $\mathcal{Q}^k_{\text{obs}}$ is the set of previously known invalid configurations, and $\mathcal{Q}^m_{\text{obs}}$ is the set of movable and previously unknown obstacles or no-fly zones. The free configuration space is defined as $\mathcal{Q}_{\text{free}} = \mathcal{Q} \backslash \mathcal{Q}_{\text{obs}}$. Also, let $\mathbf{u} : \mathcal{Q} \backslash \mathcal{Q}^k_{\text{obs}} \rightarrow \mathbb{R}^n$ be a continuous vector field that assigns a vector $\mathbf{u}(\mathbf{q})$ to each configuration $\mathbf{q} \in \mathcal{Q}_{\text{free}} \cup \mathcal{Q}^m_{\text{obs}}$. This vector field is responsible for the specification of the robot task and is computed by a global planner that has no knowledge about $\mathcal{Q}^m_{\text{obs}}$. Finally, let the path be the continuous function $\xi : [0, 1] \rightarrow \mathcal{Q}$. Our local motion planning problem can then be posed as:

Problem 1: Find, inside the ball $\mathcal{B}_r \subset \mathcal{Q}$ of radius r centered at the initial configuration $\mathbf{q}_0 \in \mathcal{Q}_{\text{free}}$, the smallest collision free path that starts at $\mathbf{q}_0$ and follows the vector field as close as possible. This problem can be written as:

$$\begin{aligned} \underset{\xi}{\text{minimize}} \quad & F[\xi, \mathbf{u}] = \int_0^1 f\left(\xi(s), \mathbf{u}\left(\xi(s)\right)\right) ds \\ \text{subject to:} \quad & \\ & \xi(0) = \mathbf{q}_0 \,, \\ & \|\xi(1) - \xi(0)\| = r \,, \\ & \xi(s) \in \mathcal{Q}_{\text{free}}, \forall s \in [0, 1] \,. \end{aligned} \tag{1}$$

It is important to notice that Problem 1 presents two major differences in relation to the standard motion planning problem. First, there is no definition of a goal configuration. This is replaced by a constraint that enforces the distance between the initial configuration and the end limit of the path. Second, $F[\xi, \mathbf{u}]$, which is a cost functional, substitutes the traditional euclidean distance function used in path planning problems. This is a function of both the length of the path and of how "close" the path is from the vector field. Since no goal configuration is defined, it is the role of this functional to dictate the direction of the robot's movement. The definition of the functional is discussed in Subsection III-B. Our methodology to solve Problem 1 is discussed in next section.

III. METHODOLOGY

In this work we chose to use RRT* as the method to solve Problem 1. RRT* is an anytime computation framework for optimization problems with complex constraints [13]. In this context, "anytime" means that the method quickly finds a feasible, but not necessarily optimal solution for the problem, then incrementally improve it over time toward optimality [14]. This characteristic fits well to our problem, which is computed on-the-fly. In our case, it means that even with limited computational resources and a small interval of time, a solution may be found. With more resources and time, this solution is then improved. Additionally, another important characteristic of RRT* is the fact that it does not require an explicit geometric representation of the obstacles, which is very important in the case of unknown obstacles detected during the motion. Because our methodology depends on

RRT*, for the sake of easier understanding, this method will be reviewed in the next subsection. The following subsections will present how RRT* was adapted/used to solve the problem presented in the previous section.

A. RRT*

The RRT* motion planning algorithm is computed in two steps. In the first step, a tree (graph without loops) with the root at the initial configuration is constructed using samples of the configuration space. In this tree, the nodes (samples) are free configurations while the edges represent the robot's paths between two configurations. In the second step of the algorithm, the goal configuration is connected to the tree and a path is found from the initial configuration to the goal as a sequence of nodes. If the goal configuration cannot be connected to the tree, either there is no solution to the problem, or more nodes (samples) would be necessary. One interesting point is that RRT* constructs an optimal tree in the sense that any path from the initial configuration to a node of the tree is the best possible, given a cost function and the current number of samples. A more in-deep discussion about RRT* is presented in [15].

In this work, since we do not specify a goal configuration *a priori*, we are mostly interested in the first step of the method, which is the optimal tree construction. Once the tree is found, our solution will be the path between the root of the tree and one of the nodes that are approximately at distance r from it. Among the equidistant nodes from the root, the node chosen to be the end of the path is the one associated to the smallest path cost. The algorithm for computing a search tree using RRT* is shown in Algorithm 1. The basic functions called by this algorithms are:

- SAMPLEFREE: Generate a random configuration in $\mathcal{Q}_{\text{free}}$.
- NEAREST($G = (V, E)$, $\mathbf{q}_\text{i}$): Finds the node of graph G that is the closest to $\mathbf{q}_\text{i}$ in terms of a given distance function.
- STEER($\mathbf{q}_\text{i}$, $\mathbf{q}_\text{j}$, η): Returns a new configuration $\mathbf{q}_\text{k}$, which is obtained by steering $\mathbf{q}_\text{i}$ towards $\mathbf{q}_\text{j}$ in a straight line such that $\|\mathbf{q}_\text{k} - \mathbf{q}_\text{j}\|$ is minimized and $\|\mathbf{q}_\text{k} - \mathbf{q}_\text{i}\| \leq \eta$.
- COLLISIONFREE($\mathbf{q}_\text{i}$, $\mathbf{q}_\text{j}$): Returns TRUE if the path from $\mathbf{q}_\text{i}$ to $\mathbf{q}_\text{j}$ lies in $\mathcal{Q}_{\text{free}}$ and FALSE otherwise.
- NEAR($G = (V, E), \mathbf{q}_\text{i}, \eta$): Computes the set of nodes that are inside the ball centered in $\mathbf{q}_\text{i}$. If the radius of the ball is given by $\min\{\gamma(log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\}$, where $\gamma > (2(1 + 1/d))^{1/d}(\mu(\mathcal{Q}_{\text{free}})/\zeta_d)^{1/d}$, d is the dimension of $\mathcal{Q}$, $\mu(\mathcal{Q}_{\text{free}})$ is the volume of $\mathcal{Q}_{\text{free}}$ and ζ_d is the volume of the unit ball in the d-dimensional Euclidean space, RRT* is asymptotically optimal, as showed in [8].
- COST($\mathbf{q}_\text{i}$): Returns the cost of the path that starts in $\mathbf{q}_0$ and finishes in $\mathbf{q}_\text{i}$.
- PATHCOST($\mathbf{q}_\text{i}$, $\mathbf{q}_\text{j}$): Computes the cost of the path between $\mathbf{q}_\text{i}$ and $\mathbf{q}_\text{j}$ using the specified cost function.
- PARENT($\mathbf{q}_\text{i}$): Returns the parent node of $\mathbf{q}_\text{i}$.

To solve Problem 1 using the RRT* tree, function PATHCOST($\cdot$) needs to be defined in agreement with the cost functional $F[\xi, \mathbf{u}]$. Also, to improve the efficiency of the method, function SAMPLEFREE may take into account the vector field. The definition of these functions is presented in the next subsections.

Algorithm 1 RRT*

$V \leftarrow \{\mathbf{q}_0\}$; $E \leftarrow \emptyset$;
for $i = 1 : n$ **do**
  $\mathbf{q}_{\text{rand}} \leftarrow$ SAMPLEFREE;
  $\mathbf{q}_{\text{nearest}} \leftarrow$ NEAREST($G = (V, E)$, $\mathbf{q}_{\text{rand}}$);
  $\mathbf{q}_{\text{new}} \leftarrow$ STEER($\mathbf{q}_{\text{nearest}}$, $\mathbf{q}_{\text{rand}}$, η);
  if COLLISIONFREE($\mathbf{q}_{\text{nearest}}$, $\mathbf{q}_{\text{new}}$) **then**
    $\mathcal{Q}_{\text{near}} \leftarrow$ NEAR($G = (V, E), \mathbf{q}_{\text{new}}, \eta$);
    $V \leftarrow V \cup \{\mathbf{q}_{\text{new}}\}$;
    $\mathbf{q}_{\text{min}} \leftarrow \mathbf{q}_{\text{nearest}}$;
    $c_{\text{min}} \leftarrow$ COST($\mathbf{q}_{\text{nearest}}$) + PATHCOST($\mathbf{q}_{\text{nearest}}, \mathbf{q}_{\text{new}}$);
    for all $\mathbf{q}_{\text{near}} \in \mathcal{Q}_{\text{near}}$ **do**
      if COLLISIONFREE($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}$) $\wedge$
       COST($\mathbf{q}_{\text{near}}$) + PATHCOST($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}$) $< c_{\text{min}}$ **then**
        $\mathbf{q}_{\text{min}} \leftarrow \mathbf{q}_{\text{near}}$;
        $c_{\text{min}} \leftarrow$ COST($\mathbf{q}_{\text{near}}$) + PATHCOST($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}$);
    $E \leftarrow E \cup \{(\mathbf{q}_{\text{min}}, \mathbf{q}_{\text{new}})\}$;
    for all $\mathbf{q}_{\text{near}} \in \mathcal{Q}_{\text{near}}$ **do**
      if COLLISIONFREE($\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{near}}$) $\wedge$
       COST($\mathbf{q}_{\text{new}}$) + PATHCOST($\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{near}}$) $<$ COST($\mathbf{q}_{\text{near}}$) **then**
        $\mathbf{q}_{\text{parent}} \leftarrow$ PARENT($\mathbf{q}_{\text{near}}$);
        $E \leftarrow (E \backslash \{(\mathbf{q}_{\text{parent}}, \mathbf{q}_{\text{near}})\}) \cup \{(\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{near}})\}$
return $G = (V, E)$

B. Cost functional

To consider both the vector field and the length of the path, we propose a cost functional of the form:

$$F[\xi, \mathbf{u}] = \int_0^1 \left(a - b \, \frac{\xi'(s)}{\|\xi'(s)\|} \cdot \frac{\mathbf{u}(\xi(s))}{\|\mathbf{u}(\xi(s))\|} \right) \|\xi'(s)\| ds \, . \tag{2}$$

where $\xi'(s) = d\xi/ds$ is the first derivative of the path with respect to the spacial parameterization variable s, and $a, b \in \mathbb{R}_+$ are constants such that $a > b$. The values of a and b are chosen so that the cost is small when the path is parallel to the vector field (the inner product between the normalized field vector and the unit vector tangent to the path is one), and increases when the path is not parallel to the field (inner product is smaller than one). If $a = 2$ and $b = 1$, for example, the cost for the case in which the path is anti-parallel to the field (inner product is -1) will be three times higher than the one in which it is parallel to field, considering the same length of the tangent vector given by $\|\xi'(s)\|$.

Assuming that $c(i, j) = F[\xi, \mathbf{u}]$ represents the cost of a path that connects i and j, for $i, j \in \mathcal{Q}$, it is important to say that Cost Functional (2) is additive and, as long as $a - b > 0$, it is also monotonic, in the sense that, $c(x, y) + c(y, z) \geq c(x, y)$ and $c(x, y) + c(y, z) \geq c(y, z)$. Additionally, notice that the cost functional is strictly positive, since $c(i, j) = 0$ only if $i = j$, and bounded, since there exist a constant k such that $c(i, j) < k$ for all $i, j \in \mathcal{Q}$ and all existing paths between i and j. On the other hand, the functional is

Algorithm 2 Cost computation between two configurations connected by a straight line path.

1: **function** PATHCOST($\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{end}}$, step)
2: **if** COLLISIONFREE($\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{end}}$) **then**
3: path_length $= \|\mathbf{q}_{\text{end}} - \mathbf{q}_{\text{start}}\|$
4: $\mathbf{v} = \frac{\mathbf{q}_{\text{end}} - \mathbf{q}_{\text{start}}}{\text{path\_length}}$
5: number_of_segments = round($\frac{\text{path\_length}}{\text{step}}$)
6: step $= \frac{\text{path\_length}}{\text{number\_of\_segments}}$
7: cost $= 0$
8: **for** length_from_start=0 : step : path_length $-$ step **do**
9: $\mathbf{q}_{\text{i}} = \mathbf{q}_{\text{start}} +$ lenght_from_start $\cdot\, \mathbf{v}$
10: $\mathbf{u} = \frac{\mathbf{u}(\mathbf{q}_{\text{i}})}{\|\mathbf{u}(\mathbf{q}_{\text{i}})\|}$
11: cost $=$ cost $+ (a - b\,(\mathbf{v} \cdot \mathbf{u})) \cdot$ step
12: **else**
13: cost$= \infty$
14: **return** cost

not symmetric, i.e. $c(x, y) \neq c(y, x)$, which is an important information for some RRT* implementations.

Assuming a straight line path, the computation of the cost between two configurations can be done using Algorithm 2. In this algorithm, the path is discretized using a sequence of segments. The number of segments will depend on the size of the original path and is determined by a nominal segment size, which is an input of the algorithm (step). The integral of the cost functional is then transformed into the sum of the costs of the segments. One observation regarding Algorithm 2 is that, frequently, the path length is not a multiple of the given step. Therefore, a new step, which is close to the nominal one, needs to be computed as shown in lines 5 and 6 of Algorithm 2.

C. Sampling Strategy

Since Problem 1 specifies that a path must start at $\mathbf{q}_0$ and finish at the surface of a ball of radius r centered at $\mathbf{q}_0$, it makes sense to uniformly generate samples only inside this ball. A way to do this is to represent the ball in spherical coordinates and sample for each coordinate. Another possibility is to sample uniformly inside the box that circumscribes the ball and to reject the samples outside the ball. Notice that this process, which would be implemented by function SAMPLEFREE used in Algorithm 1, creates a compact set of configurations, thus allowing RRT* to be used to compute paths in the original, possibly unbounded, configuration space.

By sampling inside the ball and by using cost functional (2), RRT* will find the shortest path that asymptotically converge to the integrals of the vector field in the absence of unmodeled obstacles. However, depending on the volume of the spheric search space and the number of obstacles, it may be necessary a very large number of samples to obtain a good solution. To speedup this process, we propose a strategy that eliminates samples that, likely, will not be part of the final solution.

Given a new sample $\mathbf{q}_{\text{rand}} \in Q_{\text{free}}$ we compute its nearest vertex, $\mathbf{q}_{\text{nearest}}$, among all nodes of the current search tree, as is done in the standard RRT* algorithm. If we assume that the connection between two nodes of the tree is a straight line, the normalized vector $\mathbf{v}$, that points from $\mathbf{q}_{\text{nearest}}$ to $\mathbf{q}_{\text{rand}}$, represents a vector that is tangent to a path that contains these nodes. With this in mind, we propose an acceptance/rejection test that accepts $\mathbf{q}_{\text{rand}}$ if $\mathbf{v}$ is "similar" to the vector field at $\mathbf{q}_{\text{nearest}}$, given by $\mathbf{u}(\mathbf{q}_{\text{nearest}})$, or reject $\mathbf{q}_{\text{rand}}$ with some probability if $\mathbf{v}$ and $\mathbf{u}(\mathbf{q}_{\text{nearest}})$ are not similar. The measure of similarity is the angle between the vectors, which can be easily computed by the inner product between them, provided that the field is normalized. An illustration of this strategy is shown in Fig. 2. On the left is shown a situation where $\mathbf{q}_{\text{rand}}$ would be accepted and on the right is an example where this sample would be rejected with some probability. A function that implements the strategy is shown in Algorithm 3. In this function, RAND returns a random number sampled with uniform distribution over $[0, 1]$, p_r represents the probability of rejection and θ the maximum acceptable angle between the vectors. This function must be used right after line 4 of Algorithm 1 to decide if the current $\mathbf{q}_{\text{rand}}$ will be used to generate $\mathbf{q}_{\text{new}}$ or not.

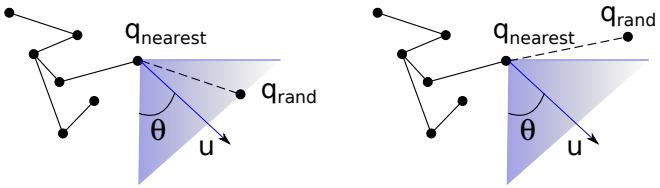

Fig. 2. Sample evaluation strategy. Given a specified angle θ and vector $\mathbf{u}(\mathbf{q}_{\text{nearest}})$, represented by the blue arrow, $\mathbf{q}_{\text{rand}}$ would be accepted on the left and rejected with some probability at the right.

Algorithm 3 Vector field based sample evaluation.

1: **function** REJECTSAMPLE($\mathbf{q}_{\text{nearest}}$, $\mathbf{q}_{\text{rand}}$, p_r, θ)
2: $\mathbf{v} = \frac{\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}}{\|\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}\|}$
3: $\mathbf{u} = \frac{\mathbf{u}(\mathbf{q}_{\text{nearest}})}{\|\mathbf{u}(\mathbf{q}_{\text{nearest}})\|}$
4: **if** $\mathbf{v} \cdot \mathbf{u} \geq \cos(\theta)$ **then**
5: reject_sample=False
6: **else**
7: **if** RAND $\leq p_r$ **then**
8: reject_sample=True
9: **else**
10: reject_sample=False
11: **return** reject_sample

The proposed strategy rejects samples that are not likely to be part of the final path when only modeled obstacles are considered. However, since we also want to avoid obstacles that were not considered during the computation of the vector field, it is important to keep some samples whose $\mathbf{v}$ vectors points against the field, which requires the probability of rejection of $\mathbf{q}_{\text{rand}}$ to be smaller than 1 and the angle between $\mathbf{v}$ and $\mathbf{u}(\mathbf{q}_{\text{nearest}})$ to be larger than 0.

Next section presents simulations that illustrates and evaluates our complete methodology.

IV. SIMULATIONS

We have implemented the proposed methodology both in Matlab and in C++ with OMPL [16] and ROS [17] for $\mathcal{Q} \subset$

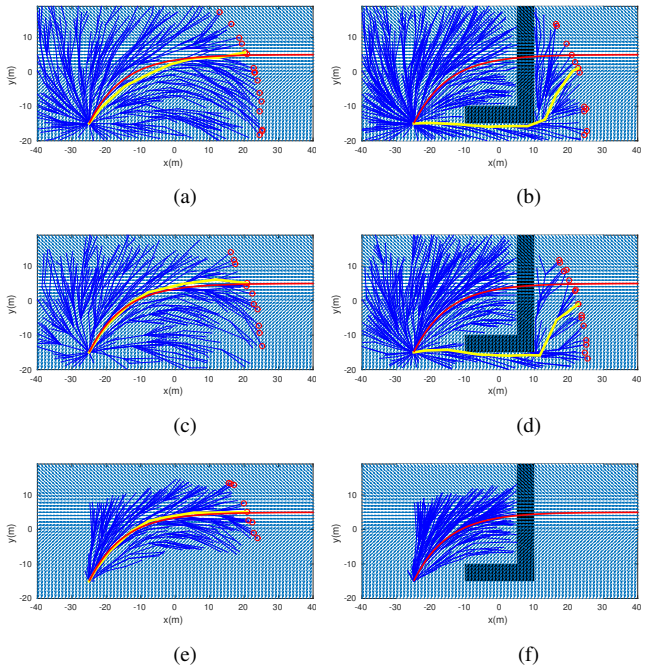

Fig. 3. Effect of the probability of sampling rejection. From top to bottom, each line of figures show the search results for $p_r = 0\%$, $p_r = 90\%$ and $p_r = 100\%$. The trees were computed for a fixed amount of time.

$\mathbb{R}^2$. Therefore, $\mathbf{q} = [x, y]^T$. In our first set of simulations, we have constructed a continuous vector field inside a 40 m wide corridor. The objective of the field is to navigate a robot along a longitudinal line positioned at distance d_0 from the center of the corridor. Assuming that the corridor is aligned with the x-axis of the world coordinate frame, this vector field can be computed as:

$$\mathbf{u}(\mathbf{q}) = \begin{bmatrix} 1 \\ k\,(d_0 - y) \end{bmatrix}, \qquad (3)$$

where d_0 is the position of the longitudinal line and k is a positive gain that determines how fast the robot moves towards the line. In our simulations we made $d_0 = 5$ and $k = 0.1$. For the sake of clear presentation, in our figures we show normalized versions of this field.

We set the radius of the planning region to be $r = 50$ m. Therefore, the path computed by RRT* must start at $\mathbf{q}_0$ and finish at a configuration that is 50 m from $\mathbf{q}_0$. To guarantee that, we grow an optimal search tree that may slightly exceeds the limits of the planning region and, during the query phase, choose, among the configurations within distances $(r - \delta, r + \delta)$, the one with the smallest cost to represent the final configuration. In our simulations, $\delta = 0.5$ m. Other parameters of the simulations are $a = 10$, $b = 9$ (Equation 2) and $\eta = 1$ m (Algorithm 1).

Figure 3 shows the behavior of the method in different situations and with different parameters. In this figure, the RRT* tree is shown in blue and the final path is shown in yellow. The red circles are the samples at distant r from $\mathbf{q}_0 = [-25, -15]^T$. These simulations were performed in Matlab for a fixed and constant interval of time. Figures 3(a) and 3(b) present simulations with no sample rejection. In (a), no unmodeled obstacles are considered while in (b) a L-shaped obstacle is introduced. Observe that the path in (a) approaches the integral of the vector field (shown in red), while the one in (b) deviate from the field to avoid the obstacle. Figures 3(c) and 3(d) show similar situations but now with the use of function REJECTSAMPLE with $\theta = 60°$ and $p_r = 90\%$. Notice that the path in (c) is closer to the integral of the field than the one in (a). Figures 3(e) and 3(f) were obtained by setting $\theta = 60°$ and $p_r = 100\%$. Notice that the path in (e) is even closer to the integral of the field than the one in (c), since more samples are "around" the optimal path. However, notice that the tree was "trapped" by the obstacle and no paths were found in (f). This indicates that, although biasing the sampling can lead to better trajectories within a fixed amount of time, a careful choice of the parameters is necessary to allow the convergence of the method in presence of obstacles.

Figure 4 shows a complete simulation of a robot starting at $\mathbf{q}_0 = [-30, 15]^T$ following a corridor with several unmodeled obstacles. In this simulation, the robot computes, for a fixed amount of time, a path that starts at $\mathbf{q}_0$ and finishes at the borders of the sampling region. Then, it follows 50% of the path and compute, for the same fixed amount of time, a new path that starts at its current configuration and finishes at borders of the new sampling region centered at the new start configuration. This is repeated until the end of the simulation. In Fig. 4, the red circles represent the initial configurations for each intermediate path computed by the robot. Sampling rejection was performed with $p_r = 60\%$ and $\theta = 60°$.

In our second set of simulations we used Gazebo [18] to test the method in a more realistic situation, where the obstacles are discovered by a UAV on-the-fly. The simulated UAV is an octo-rotor vehicle equipped with a spinning laser, such as the one presented in [19]. To detect obstacles, a local occupancy grid centered on the current robot configuration is constructed during the flight. The robot may also receive messages with no-fly zones information. The task of the UAV is to patrol a neighborhood by circulating a planar curve at a fixed height. To specify this task, we used the methodology proposed in [4] to generate a vector field that makes the robot to circulate along an implicit curve of the form $x^4 + y^4 = c^4$, where $c = 20$ m. Regarding RRT*, we set the local planning radius to be $r = 12$ m. During the flight, the UAV follows the current path for 1.8 s and, in parallel, computes the next path, which starts at the end of the current one. Its forward speed was set to be 2.0 m/s. A snapshot of the simulation along plots of the robot's paths without and with obstacles is shown in Fig. 5. A video of the simulation can be found at `http://www.cpdee.ufmg.br/~coro/movies/icuas16`.

V. CONCLUSIONS

We have presented a motion planning methodology that uses intuitive vector field approaches to define the main behavior of a robot and an optimal motion planner as local strategy to avoid previously unknown obstacles and no-fly zones. Our simulated results indicate that the proposed

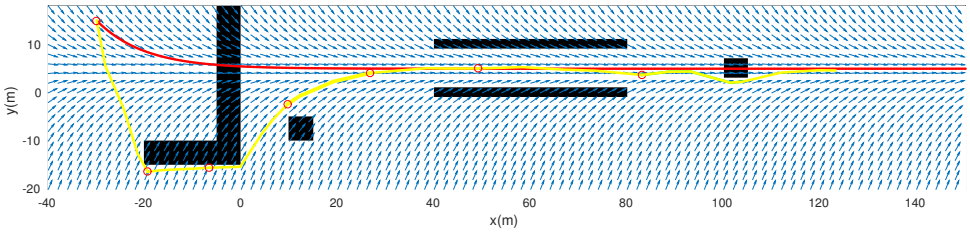

Fig. 4. Simulation showing a robot following a corridor with obstacles. The red path is the original path computed as the integral of the vector field. The yellow path is the path computed by the proposed methodology. The red circles indicate the start point of the intermediate paths computed by the robot.

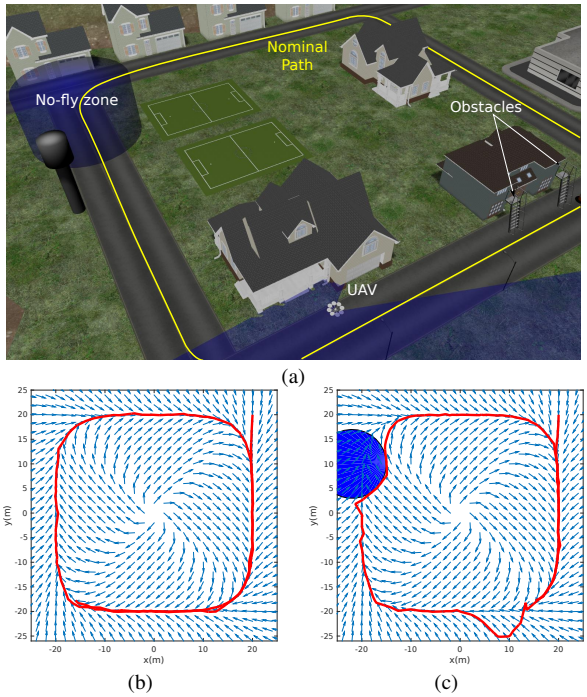

Fig. 5. Gazebo simulation of a UAV in a patrolling task: (a) Snapshot of the simulation; (b) planned path without unmodelled obstacles and no-fly zones; and (c) path in the presence of new obstacles and a no-fly zone (blue circle). In (b) and (c) the vector field is represented by the arrows.

methodology can be used to safely navigate UAVs in different tasks, including curve following and loitering, tasks in which a cost function is not well defined.

To speed up the the path computation, our future work will include the proposition of a steering function that considers the vector field. Also, we will add the vehicle's differential constraints into the optimization problem, which will allow its implementation in different UAVs, even at high speeds.

ACKNOWLEDGMENT

The authors thanks Sezal Jain for the help with the ROS/Gazebo simulations.

REFERENCES

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The Intl. J. Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

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

[3] C. I. Connolly, J. Burns, and R. Weiss, "Path planning using laplace's equation," in *IEEE Intl. Conference on Robotics and Automation*, 1990, pp. 2102–2106.

[4] V. M. Gonçalves, L. C. A. Pimenta, C. A. Maia, B. C. O. 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.

[5] E. W. Frew, D. A. Lawrence, and S. Morris, "Coordinated standoff tracking of moving targets using Lyapunov guidance vector fields," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 2, pp. 290–306, 2008.

[6] J. M. Esposito and V. Kumar, "A method for modifying closed-loop motion plans to satisfy unpredictable dynamic constraints at runtime," in *IEEE Intl. Conf. Robotics and Automation*, 2002, pp. 1691–1696.

[7] S. G. Loizou, H. G. Tanner, V. Kumar, and K. J. Kyriakopoulos, "Closed loop motion planning and control for mobile robots in uncertain environments," in *IEEE Conference on Decision and Control*, vol. 3, 2003, pp. 2926–2931.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The Intl. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[9] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[10] G. A. S. Pereira, L. C. A. Pimenta, L. Chaimowicz, A. R. Fonseca, D. S. C. Almeida, L. Q. Corrêa, R. C. Mesquita, and M. F. M. Campos, "Robot navigation in multi-terrain outdoor environments," *The Intl. Journal of Robotics Research*, vol. 28, no. 6, pp. 685–700, June 2009.

[11] A. Howard, M. J. Matarić, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Distributed Autonomous Robotic Systems 5*. Springer, 2002, pp. 299–308.

[12] A. H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz, and N. Muhammad, "Potential guided directional-RRT* for accelerated motion planning in cluttered environments," in *IEEE Intl. Conference on Mechatronics and Automation*, 2013, pp. 519–524.

[13] J. H. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, 2011, pp. 3276–3282.

[14] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *IEEE Intl. Conference on Robotics and Automation*, 2011, pp. 1478–1483.

[15] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conference on Decision and Control*, 2010, pp. 7681–7687.

[16] 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, 2012, http://ompl.kavrakilab.org.

[17] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[18] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[19] S. Jain, S. T. Nuske, A. D. Chambers, L. Yoder, H. Cover, L. J. Chamberlain, S. Scherer, and S. Singh, "Autonomous river exploration," in *Field and Service Robotics, Brisbane*, December 2013.