# Kinodynamic Motion Planning on Vector Fields using RRT*

Guilherme A. S. Pereira, Sanjiban Choudhury and Sebastian Scherer

CMU-RI-TR-16-35

July 14, 2016

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

**Abstract**

This report presents a methodology to integrate vector field based motion planning techniques with optimal, differential constrained trajectory planners. The main motivation for this integration is the solution of robot motion planning problems that are easily and intuitively solved using vector fields, but are very difficult to be even posed as an optimal motion planning problem, mainly due to the lack of a clear cost function. Examples of such problems include the ones where a goal configuration is not defined, such as circulation of curves, loitering, road following, etc. While several vector field methodologies were proposed to solve these tasks, they do not explicitly consider the robot's differential constraints and are susceptible to failures in the presence of previously unmodeled obstacles. To add together the good characteristics of each approach, our methodology uses a vector field as a high level specification of a task and an optimal motion planner (in our case RRT*) as a local planner that generates trajectories that follow the vector field, but also consider the kinematics and the dynamics of the robot, as well as the new obstacles encountered by the robot in its workspace.

**Keywords**

Navigation, motion planning, vector fields, optimal planners, RRT*.

# 1 Introduction

Aiming guidance and control of 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 each robot free configuration 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 follow this first idea, being very important the proposition of Navigation Functions [2] and Harmonic Functions [3], potential functions without local minima. In these works, the robot task was 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 different and more complex tasks, such as circulation of curves [4], deployment [5], tracking [6], 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.

Unfortunately, vector field methodologies also present some important drawbacks. Although vector fields have been used to control several kinds of real world robots, ranging from manipulators [1] to fixed wings unnamed aerial vehicles (UAV's) [6], most of the methodologies cannot be easily extended to multidimensional spaces and complex environments. Also, they do not explicitly consider the robot's differential constraints, what requires the use of non-linear controllers to track the field (what in some cases cannot be achieved) or the combination of the field with other techniques [7]. Moreover, vector fields are better applied for static and previously known workspaces. Although the technique was originally created to deal with dynamic environments, global convergence properties are generally lost in such situations. A few works have locally modified the vector field in real time to avoid dynamical obstacles and still maintain convergence properties [8, 9, 10], but these modifications generally do not consider the quality of the final trajectory.

Another important, and more recent, global motion planning technique is the asymptotically optimal version of The Rapidly-Exploring Random Tree (RRT) [11], called RRT* [12]. RRT* is used to search for optimal trajectories from a given initial configuration to a specific goal configuration or region, in spaces of large dimensions. Besides finding optimal trajectories, RRT* can include differential constraints to guarantee feasible robot trajectories [13, 14]. In fact, "RRT* is a very powerful tool that can be interpreted as an anytime computation framework for optimization problems with complex differential and geometric constraints" [14]. 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" [15]. Although RRT* is used in several robotic problems, it is limited to bounded workspaces with a well defined goal region. Because of this, it is 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 prevent its use for real time operations. Solutions for these problems have been proposed by some authors, generally integrating RRT* with other strategy. One example is [16], 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 constraints. In another work [17], RRT* was used for persistent monitoring and estimation of a time varying random field. Since the cost to be minimized is the covariance of the estimation, no goal position is required.

The idea of this work is to combine vector fields and RRT*. More specifically, we will tightly couple RRT* to a vector field. We consider the vector field as a high level global plan that must be safely followed by a differential constrained robot using RRT*. We assume that the vector field does not consider some of the details of the environment, such as small and dynamic obstacles (furniture, people and other vehicles), what will usually yields in simple and fast field computation. Although RRT* was originally conceived to be a global planner, in our approach it will work as

a local planner that will make the robot to optimally avoid the obstacles not considered by the vector field. RRT* will work on a bounded, small region of the workspace centered at the robot position, while the vector field will be constructed over the whole, possible unbounded, workspace.

In one hand, our main objective is to create a framework that extends the use of RRT* to the large variety of robotic tasks where it is not suitable to be used either by the absence of a specific goal configuration or the large size of the environment. On the other hand, we want to include some of the important properties of RRT*, such as the consideration of the vehicle differential constraints and optimality, into well known vector field methodologies. We do not assume any specific vector field methodology. Therefore, the vector field may be computed as the gradient of a potential or navigation function [1, 2, 3] but also can be one of those that aim, for example, circulation of curves for perimeter surveillance [4], corridor following [18, 19] or multi-robot deployment [5].

The use of vector fields to direct the growth of RRT* trees is considered in [20]. The authors propose a new sampling strategy that, using the vector field, guides the search tree towards the goal, thus speeding up the method. This is the only work published so far that considered both techniques in the same framework. Differently from [20], in the present work RRT* is tightly coupled with the vector field in the sense that the field is used in three steps of the methodology (i) to define a cost function, (ii) to improve sampling and (iii) in the steering procedure of the algorithm. This is only possible because, as it is usual in the RRT* literature [13], although constraints in the robot's state space are considered, the planing is actually done in the lower dimensional task space [21] or even in the configuration space, where the vector field is indeed generated. Next section will formally define our problem.

# 2 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 result in collision. 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 obstacles, and $\mathcal{Q}^m_{\text{obs}}$ is the set of movable and previously unknown obstacles. The free configuration space is defined as $\mathcal{Q}_{\text{free}} = \mathcal{Q} \backslash \mathcal{Q}_{\text{obs}}$. Also, let $\mathbf{u} : \mathcal{Q} \backslash \mathcal{Q}^k_{\text{obs}} \to \mathbb{R}^n$ 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 $\mathcal{Q}^m_{\text{obs}}$. The robot dynamics is specified by constraints of the form $g\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \dots\right) \leq 0$, $\mathbf{q} \in \mathcal{Q}$. Finally, let the trajectory $\xi : [0, 1] \to \mathcal{Q}$ be a smoothing mapping from time to configuration. Our local motion planning problem can then be posed as one of the following problems:

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

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

**Problem 2** Find the collision free and dynamically feasible trajectory of length $r$ that starts at configuration $\mathbf{q}_0$ and follows the vector field as close as possible. This problem can be written as:

$$\begin{aligned} \underset{\xi}{\text{minimize}} \quad & F[\xi, \mathbf{u}] = \int_0^1 f\left(\xi(\tau), \mathbf{u}\left(\xi(\tau)\right)\right) d\tau \\ \text{subject to:} \quad & \\ & \xi(0) = \mathbf{q}_0\,, \\ & \int_0^1 \|\dot{\xi}(\tau)\| d\tau = r\,, \\ & g\left(\xi, \dot{\xi}, \ddot{\xi}, \dots\right) \leq 0\,, \\ & \xi(t) \in \mathcal{Q}_{\text{free}}, \forall t \in [0,1]\,. \end{aligned} \tag{2}$$

It is important to notice that these problems present 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 final limit of the trajectory (Problem 1) or a constraint that enforces the length of the final trajectory (Problem 2). Second, $F[\xi, \mathbf{u}]$, which represents the cost function for the optimization problem, substituted the traditional euclidean distance function used in trajectory planning problems. In our problem, this is a function of both the length of the trajectory, which by itself is a function, and of how close the trajectory is from the vector field, which is also a function. Therefore, $F[\xi, \mathbf{u}]$ is function of functions, i.e. a functional. 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 presented in Section 4.1.

# 3 Background: RRT*

The problems defined in the previous section could be solved in several ways. In this work, we chose to use RRT* as our solver. To make it easier the understanding of the rest of this document, RRT* is summarized in this section.

RRT* basically works in two phases. In the first phase, a tree (graph without loops) with root at the origin is computed. The nodes of the tree are free random samples of the robot's configuration space and the edges represent the existence of a trajectory between two nodes. The edges are chosen so that paths (sequence of nodes) that start at the root represent the best possible trajectories from the root to any node of the tree. In the second phase of the method, called query phase, the trajectory to be followed by the robot is then extracted from the tree.

The algorithm for computing a 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: Finds the node of graph $G$ that is the closest to $\mathbf{q}_{\text{rand}}$ in terms of a given distance function.
- STEER($\mathbf{q}_\text{i}$, $\mathbf{q}_\text{j}$, $\eta$): Compute a configuration $\mathbf{q}_\text{k}$ that minimizes $\|\mathbf{q}_\text{k} - \mathbf{q}_\text{j}\|$ while at the same time maintain $\|\mathbf{q}_\text{k} - \mathbf{q}_\text{i}\| \leq \eta$.
- 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 $\mathbf{q}_\text{i}$ and radius 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 $\zeta_d$ is the volume of the unit ball in the $d$-dimensional Euclidean space [12]. This is a $O(n)$ operation.
- COST($\mathbf{q}_\text{i}$): Returns the cost of the trajectory that starts in $\mathbf{q}_0$ and finishes in $\mathbf{q}_\text{i}$. This information is generally stored in the tree so that this operation is $O(1)$.

**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$(\mathbf{q}_{\text{nearest}}, \mathbf{q}_{\text{rand}}, \eta)$;
  **if** CollisionFree$(\mathbf{q}_{\text{nearest}}, \mathbf{q}_{\text{new}})$ **then**
    $\mathcal{Q}_{\text{near}} \leftarrow$ Near$(G = (V, E), \mathbf{q}_{\text{new}}, \eta)$;
    $V \leftarrow V \cup \{\mathbf{q}_{\text{new}}\}$;
    $\mathbf{q}_{\min} \leftarrow \mathbf{q}_{\text{nearest}}$; $c_{\min} \leftarrow$ 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$(\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}) < c_{\min}$ **then**
        $\mathbf{q}_{\min} \leftarrow \mathbf{q}_{\text{near}}$; $c_{\min} \leftarrow$ Cost$(\mathbf{q}_{\text{near}})$ + PathCost$(\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}})$;
      **end if**
    **end for**
    $E \leftarrow E \cup \{(\mathbf{q}_{\min}, \mathbf{q}_{\text{new}})\}$;
    **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$(\mathbf{q}_{\text{near}})$;
        $E \leftarrow (E \backslash \{(\mathbf{q}_{\text{parent}}, \mathbf{q}_{\text{near}})\}) \cup \{(\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{near}})\}$
      **end if**
    **end for**
  **end if**
**end for**
**return** $G = (V, E)$

- 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. Generally, this operation divides the path into segments an compute the sum of the interval cost. Therefore, it is usually a $O(k)$ operation, where $k$ is the number of divisions of the path.
- Parent$(\mathbf{q}_{\text{i}})$: Returns the parent node of $\mathbf{q}_{\text{i}}$ the tree $G$. By convention, if $\mathbf{q}_{\text{i}}$ is the root of $G$, the function will return $\mathbf{q}_{\text{i}}$. This is an $O(1)$ operation.

Once the tree is computed for a given time or for a fixed number of operations, either one of their nodes is chosen to be the end of the trajectory (what will be done in this work), or the goal position is connected to tree using the same procedure used to connect $\mathbf{q}_{\text{new}}$. Since each node has a reference to its parent, a path is then computed by simply following this references from the goal to the start position.

# 4 Methodology

This section discusses our solution for the optimization problems in Section 2 using RRT*.

## 4.1 Cost functional

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

$$F[\xi, \mathbf{u}] = \int_0^1 \left( a - b \, \frac{\dot{\xi}(\tau)}{\|\dot{\xi}(\tau)\|} \cdot \frac{\mathbf{u}(\xi(\tau))}{\|\mathbf{u}(\xi(\tau))\|} \right) \|\dot{\xi}(\tau)\| d\tau \,, \tag{3}$$

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

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

where $a, b \in \mathbb{R}_+$ and $a > b$. The values of $a$ and $b$ are chosen so that the cost is small when the trajectory is parallel to the vector field (the inner product between the normalized field vector and the unit vector tangent to the trajectory is one), and increases when the trajectory 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 trajectory 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 that

$$c(i, j) = F[\xi, \mathbf{u}]\,,$$

represents the cost of a path that connects $i$ and $j$, $i, j \in \mathcal{Q}$, an important observation is that Cost Functional (3) does not represent a metric or distance function since it does not satisfy the triangular inequality, $c(x, y) \leq c(x, z) + c(z, y)$, $x, y, z \in \mathcal{Q}$, and is not symmetric, i.e. $c(x, y) \neq c(y, x)$. On the other hand, the cost functional 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 possible paths between $i$ and $j$.

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 than transformed into the sum of the costs of the segments. One observation regarding Algorithm 2 is that, frequently, the path length is not multiple of the given step. Therefore, a new step, which is close to the nominal one, needs to be computed as shown in lines 7 and 8 of Algorithm 2. Another observation is, since $v$ and step are constant inside the loop for straight line paths, they can be removed from the loop and taken into account after it, what will speedup the cost computation. We chose not to do that in our algorithm for the sake of clearer presentation.

## 4.2 Sampling Strategy

Since the problem in (1) specifies that a trajectory 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[1]. A way to do this is to represent the ball in spherical coordinates and sample for each coordinate. In 2D, the spheric coordinates are the radius, $d$, and azimuthal angle, $\theta$. Since the probability that

[1]The same observation is in place for Problem 2 represented by Equation (2).

a given sample is inside the ball of radius $R$ is proportional to $R^2$, we can generate samples for the radius as $d = r\sqrt{v_1}$, where $v_1$ is a real random number sampled with uniform distribution over the interval $[0, 1]$. The other coordinate, $\theta$, can be generated as $\theta = 2\pi v_2$, where $v_2$ is a random variable similar to $v_1$, but independent of $v_1$. Each 2D sample inside the circle is then generated as:

$$\begin{aligned} x &= d\cos(\theta) \\ y &= d\sin(\theta)\,. \end{aligned} \tag{4}$$

For 3D, we compute $d = r\,v_1^{\frac{1}{3}}$, $\theta = 2\pi v_2$, and the polar angle, as $\phi = \arccos(2v_3 - 1)$, where $v_1$, $v_2$, and $v_3$ are random and independent samplings over $[0, 1]$. With these coordinates each sample inside the sphere is computed as:

$$\begin{aligned} x &= d\cos(\theta)\sin(\phi) \\ y &= d\sin(\theta)\sin(\phi) \\ z &= d\cos(\phi)\,. \end{aligned} \tag{5}$$

By using the proposed cost functional (3), RRT* will find the shortest trajectories that will asymptotically converge to the integrals of the vector field. 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. In order to speedup this process, we propose a strategy that will generate samples that, more likely, will be part of the final solution. In order words, we generate samples in a way we try to force that the new edges have low cost in the sense of one of the posed objectives, which is to be parallel to the field.

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 tree, as it is done in the standard RRT* algorithm. As it is usual, collision samples are automatically rejected in the sampling process. Configurations $\mathbf{q}_{\text{rand}}$ and $\mathbf{q}_{\text{nearest}}$ form the normalized vector $\mathbf{v}$ as:

$$\mathbf{v} = \frac{\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}}{\|\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}\|}\,. \tag{6}$$

If we assume that the connection between two nodes of the tree is a straight line, $\mathbf{v}$ represents the direction of the edge between $\mathbf{q}_{\text{nearest}}$ and $\mathbf{q}_{\text{rand}}$ and, therefore, represents a vector that is tangent to the trajectory the passes by these nodes. With this in mind, we propose a 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. The algorithm for this strategy is shown in Algorithm 3. In this algorithm, $p_r$ represents the probability of rejection and $\theta$ the minimum acceptable angle between the vectors. Function RAND returns a random number sampled with uniform distribution over $[0, 1]$.

The proposed strategy rejects samples that are not likely to be part of the final trajectory when only modeled obstacles are considered. However, since we want to also 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, what 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.

It is important to observe that this strategy requires the computation of the nearest neighbor of $\mathbf{q}_{\text{rand}}$ before deciding to reject it or not. This indicates that, for a fixed number of iterations of RRT*, we reduce the number of nodes in the tree but we may not reduce the computational time in the same proportion, once finding the nearest neighbor is a very time consuming operation. On the other hand, one can think on this strategy as a way, for the same interval of time (not the same number of iterations), to generate more samples close to the final solution, what will probably cause a faster minimization of the cost function. In conclusion, this method may be quicker than the original to obtain a good trajectory.

**Algorithm 3** Vector field based sample evaluation

**function** EVALUATE_VECTOR($\mathbf{q}_{\text{nearest}}$, $\mathbf{q}_{\text{rand}}$, $p_r$, $\theta$)
  $\mathbf{v} = \frac{\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}}{\|\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}\|}$
  $\mathbf{u} = \frac{\mathbf{u}(\mathbf{q}_{\text{nearest}})}{\|\mathbf{u}(\mathbf{q}_{\text{nearest}})\|}$
  **if** $\mathbf{v} \cdot \mathbf{u} \geq \cos(\theta)$ **then**
    reject_sample=False
  **else**
    **if** RAND $\leq p_r$ **then**
      reject_sample=True
    **else**
      reject_sample=False
    **end if**
  **end if**
  **return** reject_sample
**end function**

**Algorithm 4** Vector field based steering

**function** FIELD_STEERING($\mathbf{q}_{\text{nearest}}$, $\mathbf{q}_{\text{rand}}$, $\eta$, $p_f$)
  module=$\|\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}\|$
  direction=$\frac{\mathbf{q}_{\text{rand}} - \mathbf{q}_{\text{nearest}}}{\text{module}}$
  **if** module $> \eta$ **then**
    module $= \eta$
  **end if**
  **if** RAND $\leq p_f$ **then**
    direction= $\frac{\mathbf{u}(\mathbf{q}_{\text{nearest}})}{\|\mathbf{u}(\mathbf{q}_{\text{nearest}})\|}$
  **end if**
  $\mathbf{q}_{\text{new}} = \mathbf{q}_{\text{nearest}} + \text{module} \times \text{direction}$
  **return** $\mathbf{q}_{\text{new}}$
**end function**

## 4.3 Local Steering

Given configurations $\mathbf{q}_{\text{nearest}}$ and $\mathbf{q}_{\text{rand}}$, the steering function generates a third configuration $\mathbf{q}_{\text{new}}$ that minimizes $\|\mathbf{q}_{\text{new}} - \mathbf{q}_{\text{rand}}\|$ and maintain $\|\mathbf{q}_{\text{new}} - \mathbf{q}_{\text{nearest}}\| \leq \eta$, for $\eta > 0$. Notice that $\eta$ specifies a ball centered in $\mathbf{q}_{\text{nearest}}$[2]. When no differential constraints are considered, $\mathbf{q}_{\text{new}}$ is given by a vector that starts at $\mathbf{q}_{\text{nearest}}$, points to $\mathbf{q}_{\text{rand}}$ and has maximum length given by $\eta$.

To favor the minimization of the cost function while steering, we propose a simple modification to the computation of $\mathbf{q}_{\text{new}}$. The idea is that, instead of the vector that points to $\mathbf{q}_{\text{rand}}$, with some probability, the vector field at $\mathbf{q}_{\text{nearest}}$, $\mathbf{u}(\mathbf{q}_{\text{nearest}})$, is used to compute $\mathbf{q}_{\text{new}}$. The proposed steering algorithm is shown in Algorithm 4. It is important to mention that this algorithm tries to generate a low cost edge between $\mathbf{q}_{\text{nearest}}$ and $\mathbf{q}_{\text{new}}$. However, since we may have several low cost nodes in the neighborhood of $\mathbf{q}_{\text{new}}$ we cannot guarantee that $\mathbf{q}_{\text{nearest}}$ will be the parent of $\mathbf{q}_{\text{new}}$ in the final tree, although we increase the probability that this connection occurs.

## 4.4 Differential Constraints

So far, we have assumed that the robot had no differential constraints. If this is the case, the connection between two configurations, which until now have been done using straight line trajectories, needs to consider the vehicle constraints. This is done by modifying the steering function so that the connection between two nodes of the three is the optimal, constrained trajectory between them. For nonholonomic systems, a popular dynamic model, which can represent several vehicles,

[2]In the RRT* algorithm $\eta$ also specifies the neighborhood region of a given node.

is the 2D Dubin's model [13]:

$$\dot{x} = v\,\cos\theta \tag{7}$$

$$\dot{y} = v\,\sin\theta \tag{8}$$

$$\dot{\theta} = \omega\,, \qquad \|\omega\| \leq \frac{v}{\rho}\,, \tag{9}$$

where $[x, y, \theta]$ is the configuration vector of the vehicle, $v$ and $\omega$ are its control inputs given by linear and angular velocities respectively and $\rho$ is the vehicle minimum turning radius. For the Dubin's model, the optimal trajectory between any pair of configurations is one among six types of trajectories that either go straight or turn the vehicle with its maximum steering, which is limited by $\rho$.

The RRT* implementation used in this work to solve the Dubin's model trajectory planning, as suggested by [13], samples in the configuration space of the vehicle, considering its position and orientation, but not the derivatives of these variables. A difference here is that the sampling region is given by a ball in 2D centered at the initial position of the vehicle. Similarly, the trajectory length relies on simple euclidean distance that ignores the orientation of the vehicle. This makes sense if we consider that the vector field used in this case is 2D, computed only for $x$ and $y$.

The steering procedure used in this work computes $\mathbf{q}_{\text{new}}$ as explained Section 4.2 and returns one of the six possible trajectories between $\mathbf{q}_{\text{nearest}}$ and $\mathbf{q}_{\text{new}}$. The cost for this trajectory is computed by dividing the trajectory in several waypoints and computing the cost function between each pair of waypoints assuming they are connected by straight lines. For a good approximation, a large number of waypoints is necessary. It is important to remember that the steering function and the computation of the trajectory cost is also used to choose the minimum cost parent for $\mathbf{q}_{\text{new}}$ and to rewire the tree. Therefore, it is expected that the computational cost of RRT* when this kind of trajectory is used is highly increased when compared to straight line trajectories.

### 4.5 Real-time Planning

In our approach, RRT* is used as the local planner that tries to find the best trajectory inside a ball shaped search region. For real time planning in actual environments, we then use an idea similar to the one proposed in [15]. In this approach, given a trajectory $\xi : [0, 1] \to \mathcal{Q}_{\text{free}}$ generated by RRT*, the robot executes an initial portion of $\xi$ until a given commit time $t_{\text{com}} < 1$. This initial trajectory is called *committed trajectory* [15]. Once the robot starts following the committed trajectory, RRT* is executed again with the final configuration of the committed trajectory, $\xi(t_{\text{com}})$, as the new root of the tree. To avoid changing the committed trajectory, the authors of [15] remove the branches of the tree that starts at the nodes of the committed trajectory, but $\xi(t_{\text{com}})$. When the robot reaches $\xi(t_{\text{com}})$, a new committed trajectory is selected as the best current trajectory in the tree. Since in our case there is no definition of a goal position, the process repeats until some stopping criteria is reached. It is important to notice that this process is only effective because, in general, the robot can compute the trajectories much faster than it can move.

An important observation is relative to the choice of $t_{\text{com}}$. If the robot does not have a complete knowledge about the environment, what is the case assumed here, it does not know the shape and size of the detected obstacles. Therefore, given an estimate of the environment obtained by the robot's noisy and limited sensors, it is not possible to guarantee that the initial trajectory $\xi : [0, 1]$ is indeed in $Q_{\text{free}}$. However, given the characteristics of the sensors, it is normally possible to determine with some degree of certainty the free region inside the sensors field of view. The size of this region and the velocity of the robot may then be used to compute $t_{\text{com}}$.

## 5 Simulations

We have implemented the proposed methodology both in Matlab and in C++ using OMPL [22] and ROS [23] for $\mathcal{Q} \subset \mathbb{R}^2$ and $\mathcal{Q} \subset SE(2)$.

In our first set of simulations, tested in Matlab, we have constructed a continuous vector field inside a 40 m wide corridor. The objective of the field is to navigate a holonomic robot with

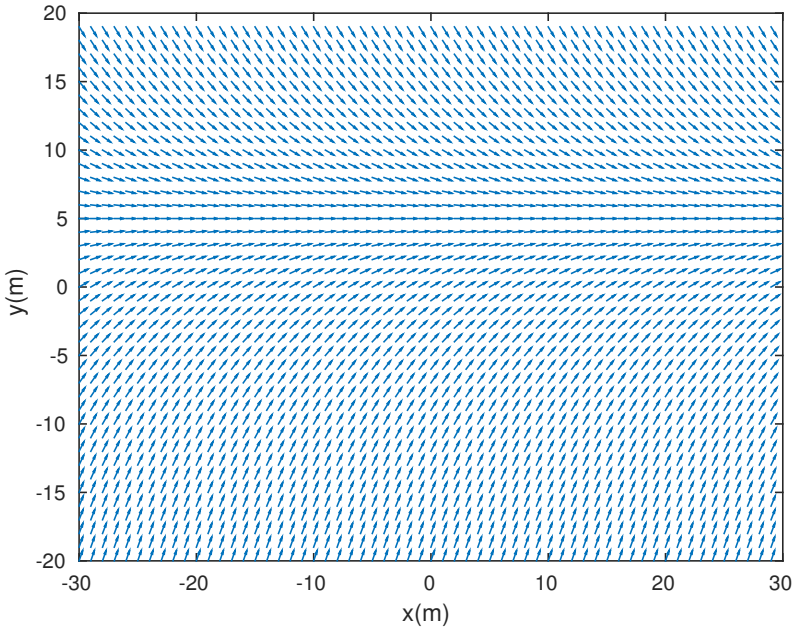



Figure 1: Vector field used in our first set of simulations.

$\mathbf{q} = [x, y]^T$ along a longitudinal line positioned 5 m from the center of the corridor to the left. This vector field, which is shown in Figure 1, can be computed as:

$$\mathbf{u}(\mathbf{q}) = \begin{bmatrix} 1 \\ k\,(d_0 - y) \end{bmatrix}, \tag{10}$$

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

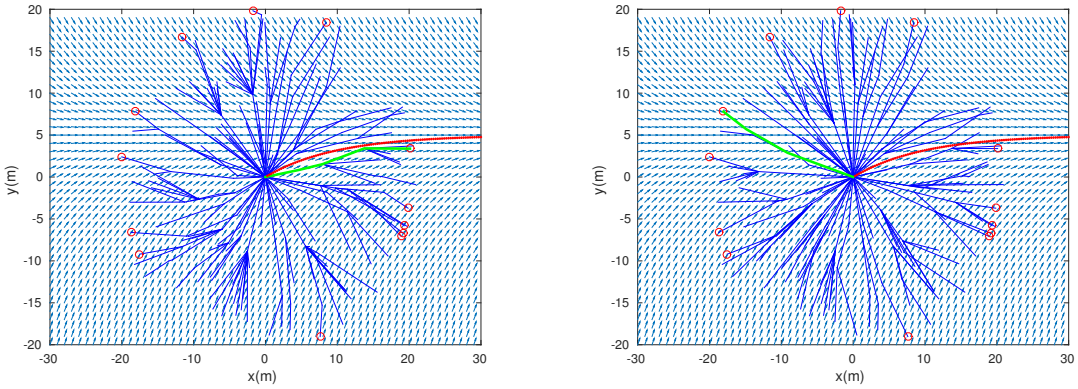
## 5.1 Cost Function

We have considered the robot initial configuration to be $\mathbf{q}_0 = [0, 0]^T$ and the radius of the planning region to be $r = 20$ m. Assuming that we are solving Problem 1, the trajectory must start at $\mathbf{q}_0$ and finish at a configuration that is 20 m from $\mathbf{q}_0$. To guarantee that, we grow an optimal 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. Also, in this set of simulations, no differential constraints were considered. For sampling, we used an uniform distribution inside the ball of radius $r + 1$ m centered at $\mathbf{q}_0$. No samples were rejected. A linear steering function with $\eta = 10$ was applied to generate new configurations from the sampled ones [12].

In the first test we compare the standard Euclidean cost function with cost functional (3), where we set $a = 5$ and $b = 4$. This result is in Figure 2. Notice that the best path computed with the proposed cost function, shown in Figure 2(a), approximates the integral of the vector field while the one that simply minimizes the length of the path, shown in Figure 2(b), approaches a straight line that connects $\mathbf{q}_0$ to one of the configurations at the limits of the planning ball.

It is important to mention that both random trees in Figure 2 were generated with the same seed. This means that both trees share the same 200 vertices. In spite of this, it can be observed that they are not identical, once the cost function changes the way connections are made.

In our second test we evaluate both the quality of the path as the number of samples increase and the behavior of the method when an unmodeled obstacle is introduced in the workspace. In this simulation, we have changed the initial position to $\mathbf{q}_0 = [-25, -15]^T$ and the radius of the




(a) (b)

Figure 2: Comparison between the proposed field-based cost function (a) and the standard Euclidean one that minimizes only the path length (b). Both figures show a vector field (pink arrows) overlayed by the tree (blue), the integral of the field (red dots), the vertices of tree at $r = 20\,\text{m}$ from the start position (red circles) and the best trajectory encountered by RRT* (green). In (a), the computed trajectory approaches the integral of the field. In (b), the computed trajectory approaches a straight line that connects the initial position to one of the configurations at the border of the planning region.

planning region to $r = 50\,\text{m}$. All other parameters are the same. Our simulations are shown in Figure 3. In the figures on the left column it is possible to see that the computed path (when found) approaches the integral of the field as the number of samples increases. This illustrates the "anytime" characteristic of the method. Since most of robots take more time to follow a trajectory than to compute it, a robot could start following a "not so good" trajectory while this trajectory is improved towards the optimal one.

In the second column of Figure 3 it is shown the behavior of the method in the presence of an obstacle positioned at the desired longitudinal line. This obstacle was unknown during the field computation. Therefore, for our initial configuration, the integral of the field would lead to a collision between the robot and the obstacle. Notice that the RRT* based local planner was able to compute a path that avoids the obstacle and, simultaneously, try to follow the vector field as close as possible.

## 5.2 Sampling

We also performed a series of Matlab simulations to evaluate the proposed sampling strategy. In this set of simulations, we fixed the maximum processing time and measure the number of nodes in the tree for different values of probability of rejections, $p_r$. In all simulations of this section we chose $\theta = 60\,\text{degrees}$. Figure 4 shows the result of our simulations. In the left column of this figure, where no obstacles are considered, it is possible to see an improvement on the computed trajectory (in green), which approaches the field integral (in red), as the probability of rejection in Algorithm 3 increases. Another important observation is that, as shown in Table 1, this improvement is followed by a reduction in the number of samples in the tree.

In the right column of Figure 4 we see the results obtained when we have introduced a large non-convex obstacle in the environment. Notice that the solution for the problem in this case would be a trajectory that is not always tangent to field. In this case, our results show that once the probability of rejection increases, the probability of finding a solution decreases. In the limit, when $p_r = 100\%$ no solutions are found.

Table 1: Number of nodes in the trees for the simulations in Figure 4.

| $p_r$ | free | obstacle |
|---|---|---|
| 0% | 532 | 698 |
| 50% | 509 | 664 |
| 90% | 454 | 605 |
| 100% | 329 | 333 |

### 5.3 Steering

All previous simulations used a standard steering function with $\eta = 10$. In this section we evaluate the effect of different values of $\eta$. It is important to remember that $\eta$ is used both in the steering function and to define the ball of near neighbors of a new sample. Because of this, while large values of $\eta$ increase the probability of quickly finding a solution, since it allows the new nodes to be far from the closest node already in the tree, they also increase the computational time, due to a larger number of near neighbors. Figure 5 shows the results for three values of $\eta$ and 2000 iterations. In this simulations we have used standard uniform sampling inside the circular searching region. Notice that, the size of the graph edges tend to increase as $\eta$ increases. Also, observe that, for this specific number of iterations, the final trajectories are very similar for all cases, what suggests that the steering value should not be chosen based on the quality of the final trajectory, but as a compromise between how fast a feasible solution is found and the computational time necessary to optimize this trajectory.

We have also evaluated the proposed steering function in Algorithm 4, in which, with some probability, chooses to follow the field to generate a new node to the tree. The results for four values of the probability of following the field are in Figure 6. Notice at left that as $p_f$ increases, the quality of the trajectory also increases. However, a large $p_f$ may prevent the tree to reach the borders of the search region in the presence of obstacles, as shown in the right column of Figure 6.

### 5.4 Differential Constraints

Using OMPL and ROS we have also tested the ability of the method to deal with different steering functions, in our case, Dubins' functions. In this simulation we then consider the robot orientation, making $\mathcal{Q} \subset SE(2)$. It is assumed that the vector field is computed over a reduced version of the robot's configuration space composed by position only. Therefore, for a planar robot with configuration given by $\mathbf{q} = [x, y, \theta]^T$, we assume that, given a position $(x, y)$, the vector field is constant for all values of orientation $\theta$. Orientation is also ignored by functions $\textsc{Near}(\cdot)$ and $\textsc{Nearest}(\cdot)$, which we assume to apply an Euclidean distance function over the linear components of the configuration. However, it is important to observe that the complete configuration (position and orientation) is used in all other steps of the method since $\mathbf{q}_{\text{rand}}$ is sampled over a region of the complete configuration space.

In Figure 7 we show a simulation where a robot is in a large environment with a single unmodeled obstacle (blue box). We assumed that the robot can be modeled as a Dubins' vehicle with turning radius $\rho = 2\,\text{m}$. The robot starts at configuration $\mathbf{q}_0 = [-25, 15, 0]^T$ and must follow the vector field in Figure 1. Since the obstacle was not known during the field computation, the integral of the vector field (black trajectory) would lead the robot to a collision. Using the proposed methodology, the robot was able to construct a tree (shown by the blue lines) inside its sensor radius, $r = 50\,\text{m}$ (shown in yellow), and find a feasible trajectory (shown in red) that both avoids the obstacle and follow the vector field as close as possible. Notice that local paths between two nodes can be now be straight lines or arcs.

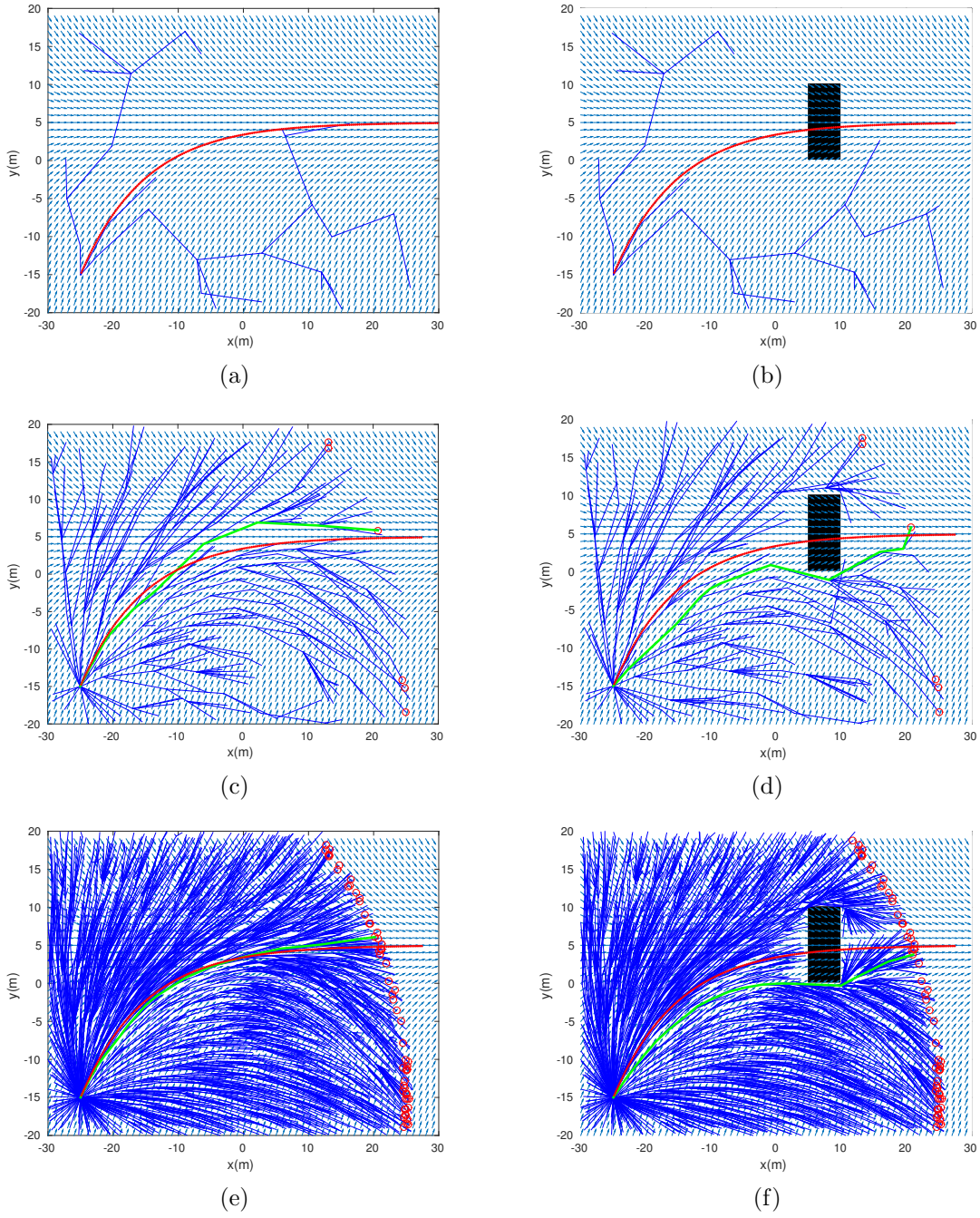



Figure 3: Paths found (green) as the number of samples increase: (a)-(b) – 30 samples (no paths found), (c)-(d) – 300 samples and (e)-(f) – 3000 samples. In the left column the corridor is free and in the right column there is rectangular obstacle (black) centered at the required path. This obstacle was unknown during the field computation. In all figures, the red paths represent the integral of the vector field and the red circles the vertices of the tree localized at the border of the planning region.

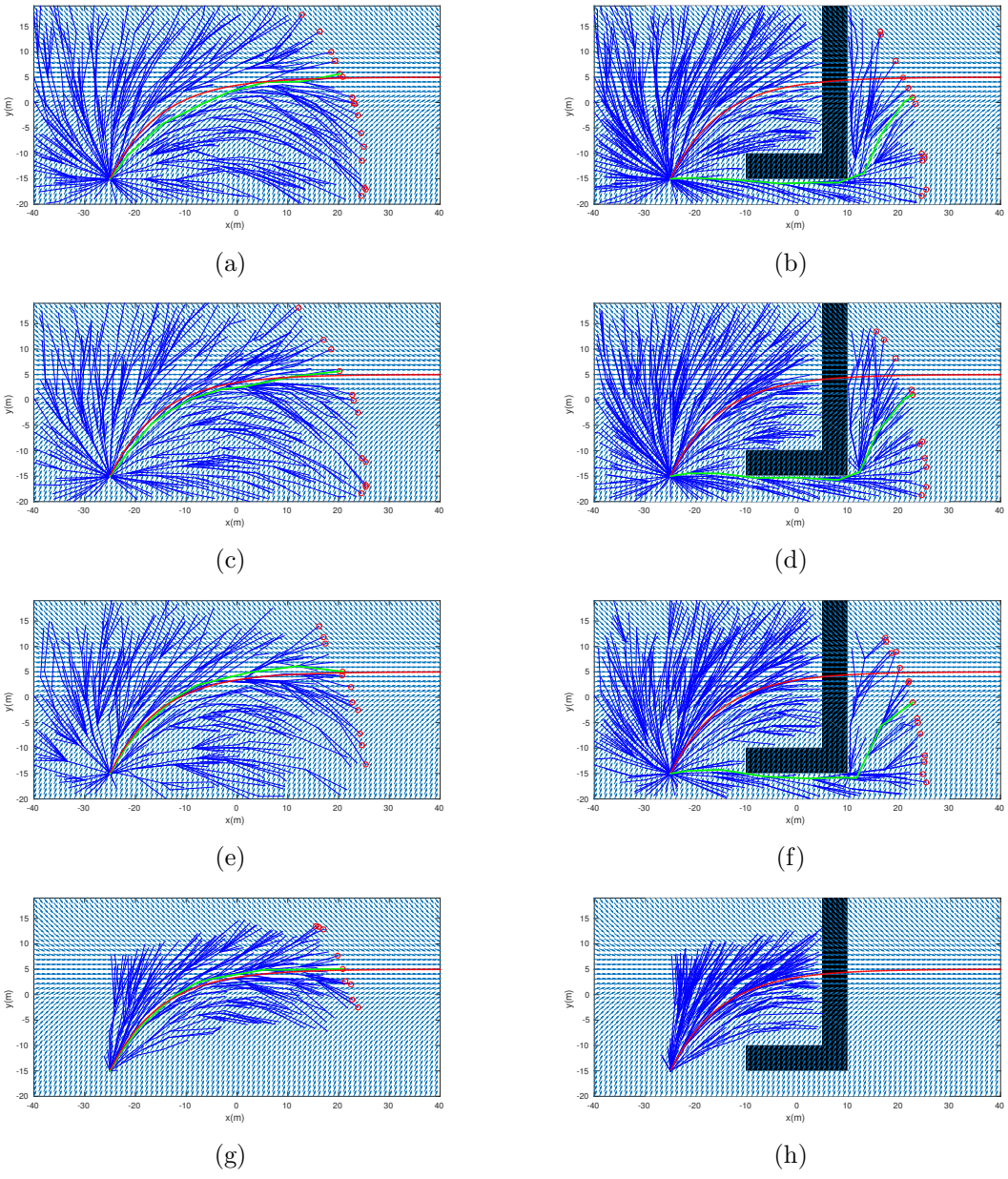



Figure 4: Effect of the probability of sampling rejection. From top to bottom, each line of figures show the search results for $p_r = 0\%$, $p_r = 50\%$, $p_r = 90\%$ and $p_r = 100\%$. The trees were computed for a fixed amount of time. The number of nodes in each tree is shown in Table 1.

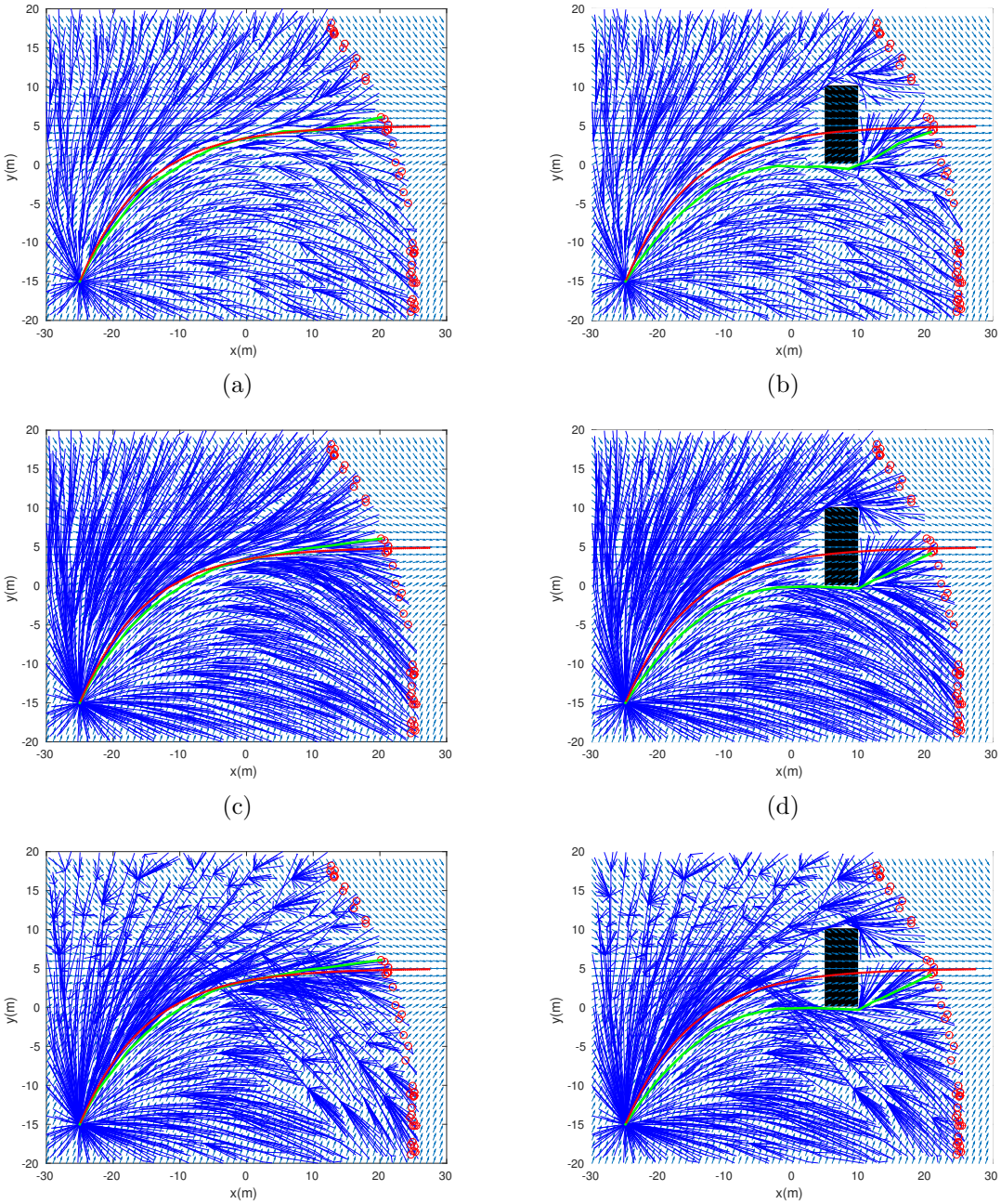



Figure 5: Effect of $\eta$. From top to bottom, each line of figures show the search results for $\eta = 5$, $\eta = 10$ and $\eta = 20$. The trees were computed with 2000 iterations.

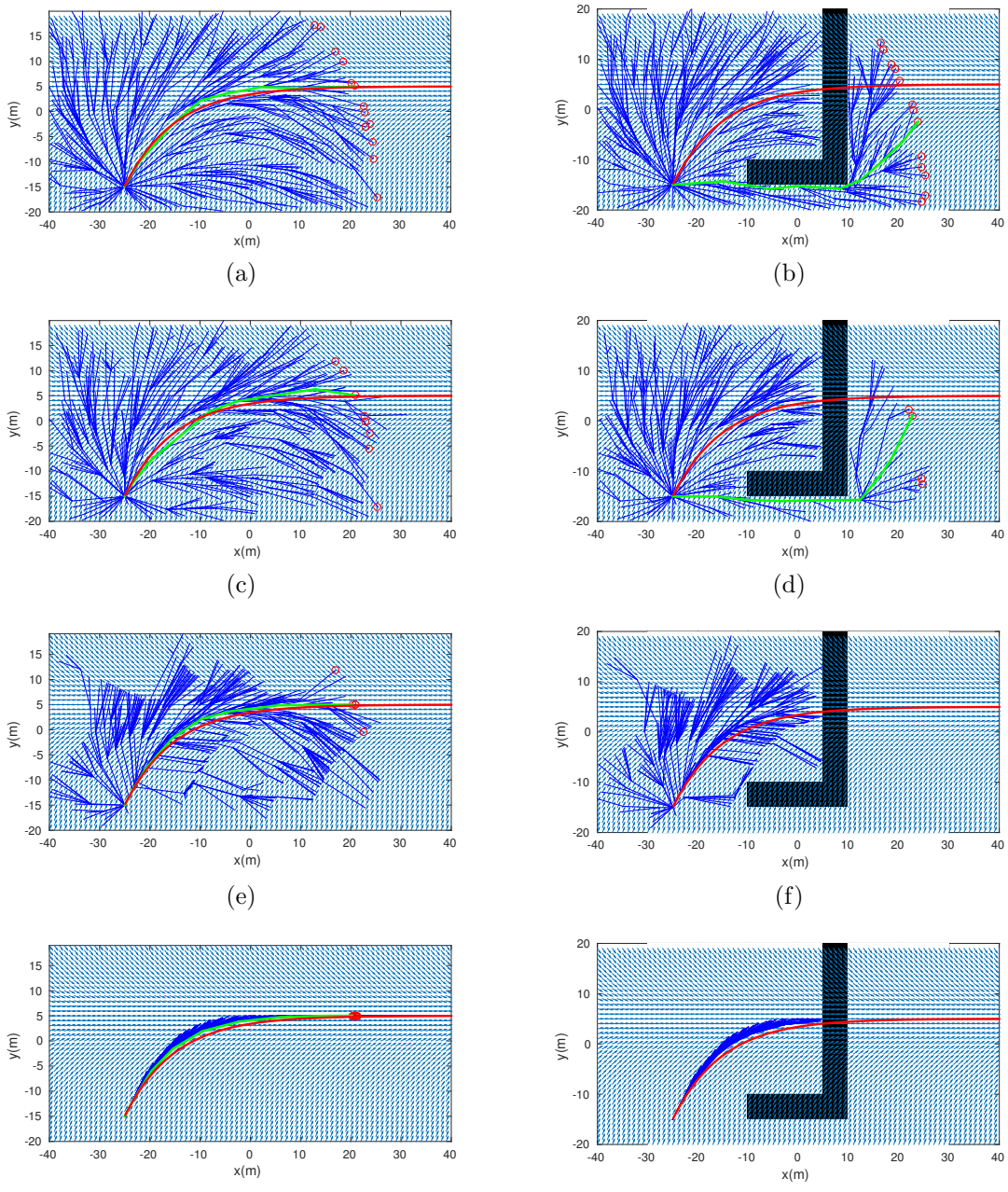



Figure 6: Simulations with the proposed steering function. From top to bottom, each line of figures show the search results for $p_f = 0\%$, $p_f = 50\%$, $p_f = 90\%$ and $p_f = 100\%$. The trees were computed with 500 iterations.

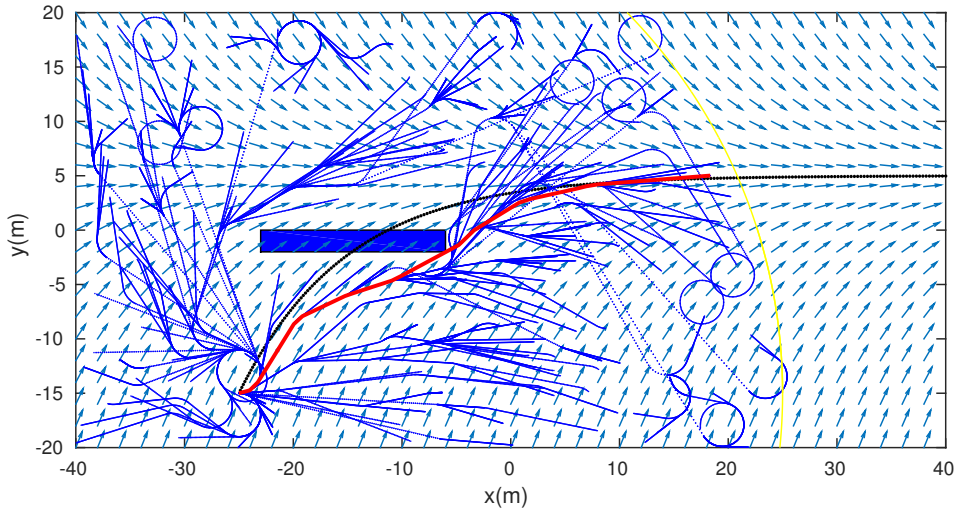



Figure 7: Illustrative example of the method with Dubins' steering functions. A vector field (arrows) was computed without the knowledge of the obstacle (blue box). Thus, the integral of the vector field (black) would lead the robot to a collision. The proposed methodology uses a RRT* tree (blue lines) to find a dynamically feasible, and asymptotically optimal trajectory (red) that avoids the obstacle and follows the vector field.

## 5.5 Applications

In this section we use the proposed methodology to guide mobile robots during the execution of their tasks.

### 5.5.1 Corridor Following

In our first simulations of this section, executed in Matlab, we assume the same corridor used previously, but now with an extension of 190 m. A holonomic robot must follow this corridor in the longitudinal line represented by $y = 5$ m. The vector field used to accomplish this task is the one in Equation 10. To apply our method, the robot computes, for a fixed amount of time, a trajectory that starts at $\mathbf{q}_0$ and finishes at the borders of the sampling region. Then, it moves for up to five waypoints of the trajectory and compute, for the same fixed amount of time, a new trajectory that starts at its current configuration and finishes at the border of the new sampling region centered at the new start location. This process is repeated until the end of the simulation. We chose the starting configuration to be $\mathbf{q}_0 = [-30, 15]^T$ and the radius of the search region to be $r = 40$ m. Also, we have used our sampling strategy with $p_r = 60$ and $\theta = 60$ degrees. We used the standard linear steering with $\eta = 10$ m, since the proposed strategy proved to be a wrong choice in environments with non-convex obstacles, even with low values of $p_f$.

The results obtained are shown in Figure 8. In this figure, the red circles represent the initial configurations for the intermediate trajectories computed by the robot. In Figure 8(a), the corridor is empty and the final trajectory follows the integral of the vector field very closely. This trajectory was obtained with five intermediate search trees, being the number of nodes on these trees 500, 591, 599, 608, and 610. In Figure 8(b) we show the result obtained when several unmodeled obstacles were included in the corridor. Notice that the robot is able to avoid the obstacles and follow the field when this is possible. The trajectory in this case was computed using seven search trees with number of nodes equal to 559, 669, 725, 731, 704, 749, and 757. As a comparison, if the standard sampling strategy is used, as shown in Figure 8(c), the number of nodes on the tree are equal to 593, 673, 776, 742, 763, 808, and 793. Notice that even with a reduced number of nodes, the trajectory found with the proposed sampling strategy is slightly better then the one obtained with the standard one.

### 5.5.2 Circulation of curves

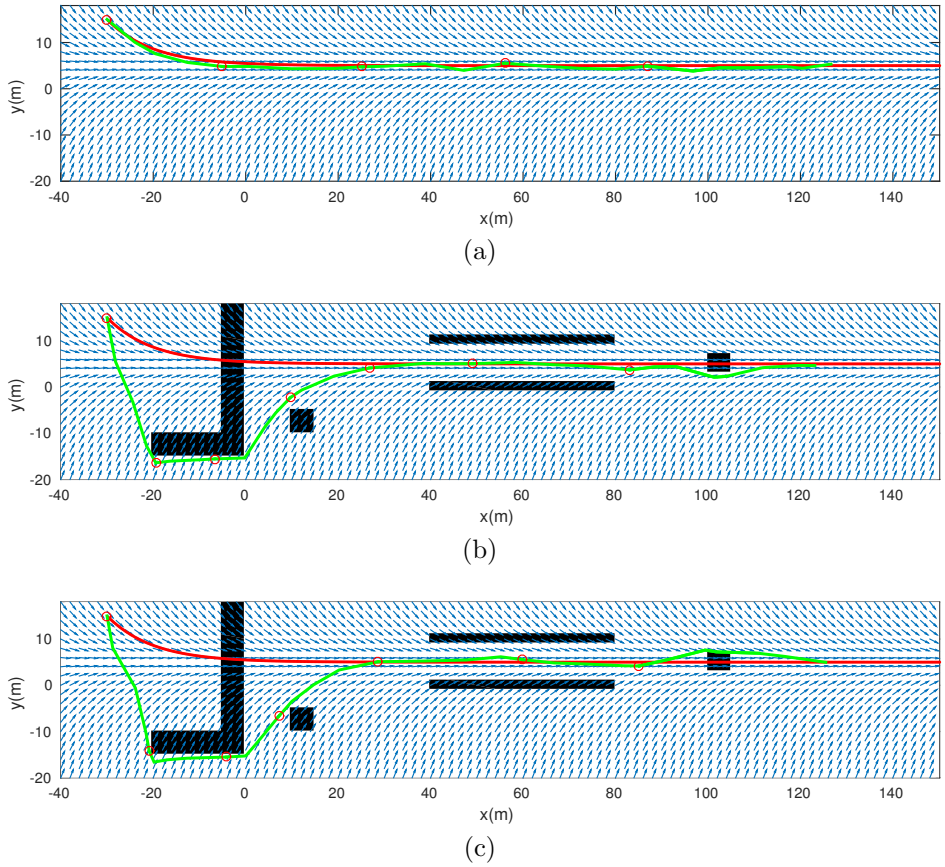
In our second application, a robot must circulate a curve in $\mathbb{R}^2$. We have used the methodology proposed in [4] to generate a vector field that makes the robot converge to and circulate along a implicit curve of the form:

$$x^4 + y^4 = a^4\,, \tag{11}$$

where we chose $a = 20$ m. Since we are dealing with a smaller environment, we set the local planning radius to be $r = 20$ and the steering parameter to be $\eta = 5$. Similar to the previous simulation, the robot plans a trajectory for a fixed amount of time, moves along up to five waypoints of this trajectory and start another planning process inside the local planning circle. Because the orientation of the field may present several changes inside the planning circle, in this simulation we do not use the proposed field based sampling or steering functions. The results are in Figure 9. Notice in Figure 9(a) that the resultant trajectory circulates the target curve, as desirable. In Figure 9(b), it is possible to see that the introduction of a large obstacle locally changes the final path, so the obstacle is avoided.

For this application, we also tested the approach for non-holonomic robots using our OMPL/ROS implementation. We assumed that the robot can be modeled as a Dubins' vehicle with turning radius $\rho = 2$ m. We set the radius of the planning region to be $r = 10$ m. All simulations run in an Intel Core i7-3.00GHz Linux computer.

Figure 10 shows the results of the method when each search tree is computed in 1 s (figures (a) and (c)) and = 5 s (figures (b) and (d)) with and without obstacles (blue rectangles). Notice that, although all trajectories computed are dynamically feasible, the quality of the trajectory increases if the method has more time to execute. The trajectory in Figure 9(c), for example, has




(a)

(b)

(c)

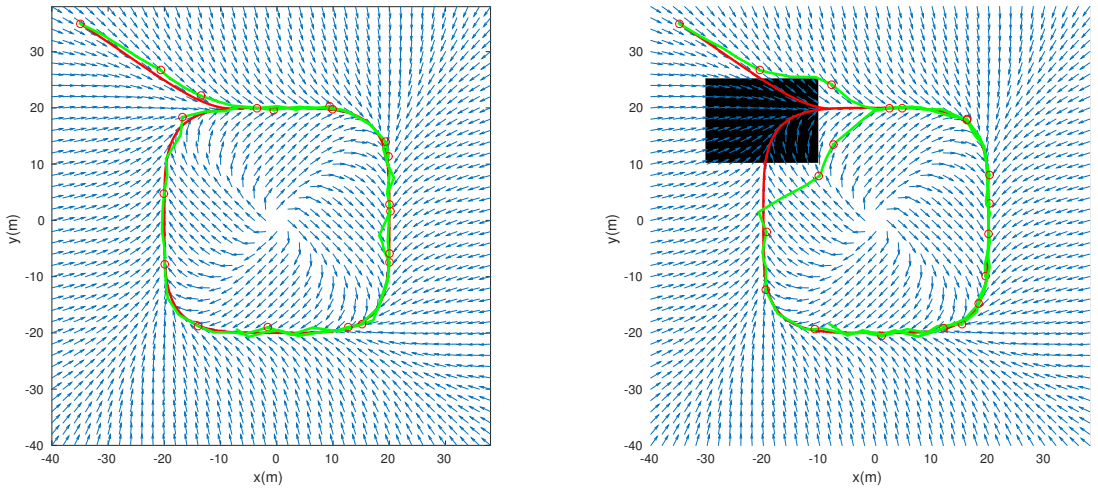
Figure 8: Simulations showing a robot following a corridor. The red circles indicate the intermediate points where search trees were constructed. (a) corridor without obstacles, (b) and (c) corridors with obstacles. In (a) and (b) was used the proposed sampling function and in (c) uniform sampling inside the circular search region.

a loop at the bottom right of the figure, which is removed in the result of Figure 9(d), where more optimization time was spent.

We also tested the method in a more realistic situation, where the obstacles are discovered by a simulated Unnamed Air Vehicle (UAV) on-the-fly. For this, we have used Gazebo [24], a dynamic robot simulator. The simulated UAV is an octo-rotor vehicle equipped with a spinning laser, such as the one presented in [25]. 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 similar to the previous one. Regarding RRT*, we set the local planning radius to be $r = 12\,\mathrm{m}$. 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. 11. A video of the simulation can be found at `http://www.cpdee.ufmg.br/~coro/movies/icuas16`.

### 5.5.3 Navigation Function

To test our method with a different vector field, we have constructed a navigation function in the standard spheric world, as proposed by [2]. The field used in our methodology is the negative of the gradient of the navigation function. Since the robot has a specific goal, when the goal entered the search radius it was added as a node in the search tree, so the robot as able to follow the path from the initial configuration to the goal configuration. In this Matlab simulation we have assumed




(a) (b)

Figure 9: Simulations showing a robot circulating a curve without (a) and with (b) the presence of a previously unmodeled obstacle.

a holonomic robot and used $r = 10$ and $\eta = 2$. In Figure 12 we show the results for situations with and without unmodeled obstacles.

# 6 Conclusions and Future Work

This report presented a motion planning framework that integrates vector field methodologies with optimal motion planners to allow the safe navigation of nonholonomic robots in partially unknown workspaces. Simulated results shown the potentiality of the method to control actual robots in tasks that cannot be solved by optimal motion planners alone.

Next steps of this research include the use of the method to control ground and aerial mobile robots working in urban environments in the presence of obstacles and people. To do so, the current implementation needs to be extended to 3D and sped up to allow better trajectories in less time.

# Acknowledgments

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."

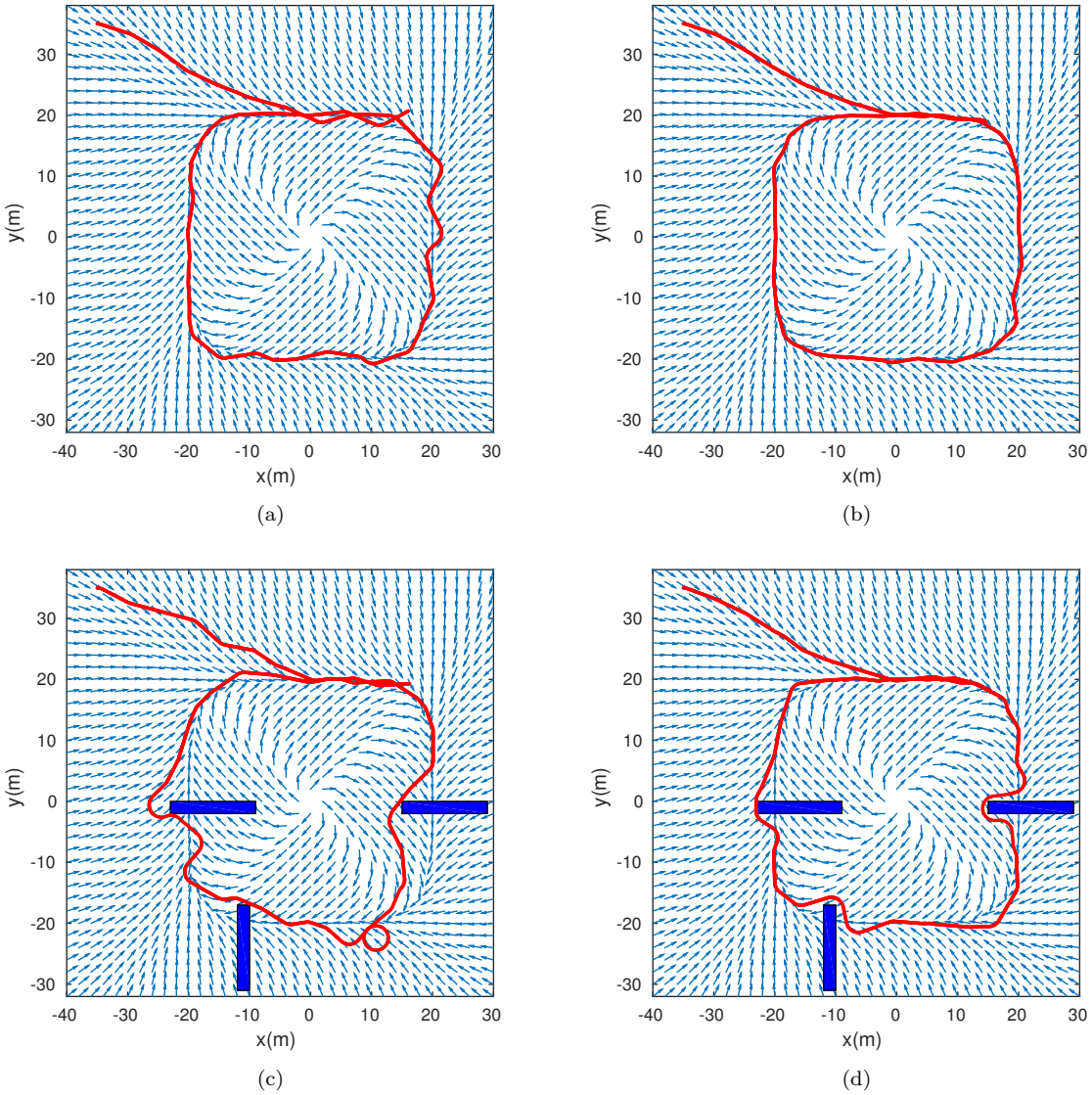



Figure 10: Circulation of curves using a non-holonomic robot. Effect of the search time. On the left search time is 1 s and on the right 5 s. (a)-(b) free workspace; (c)-(d) presence of unmodelled obstacles.

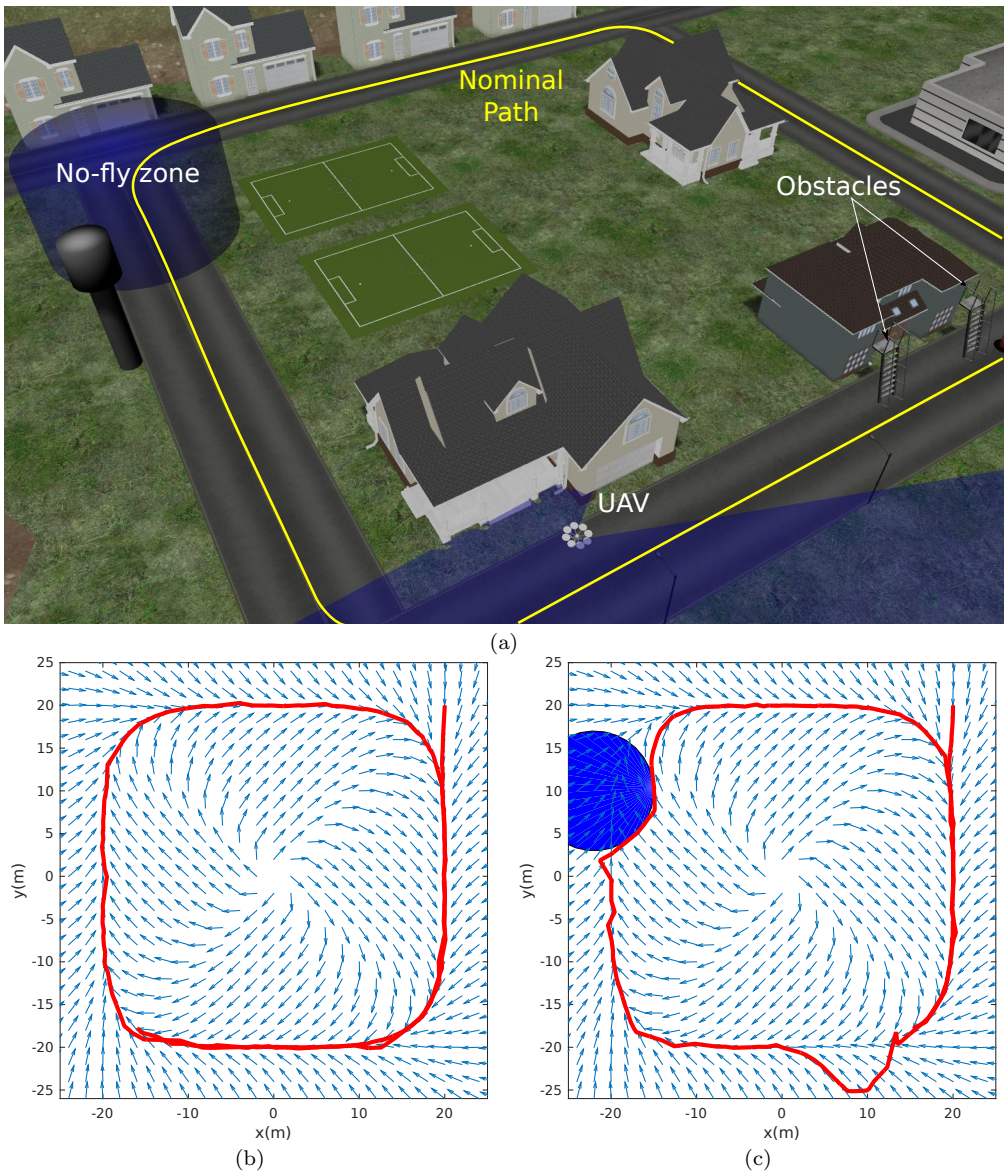
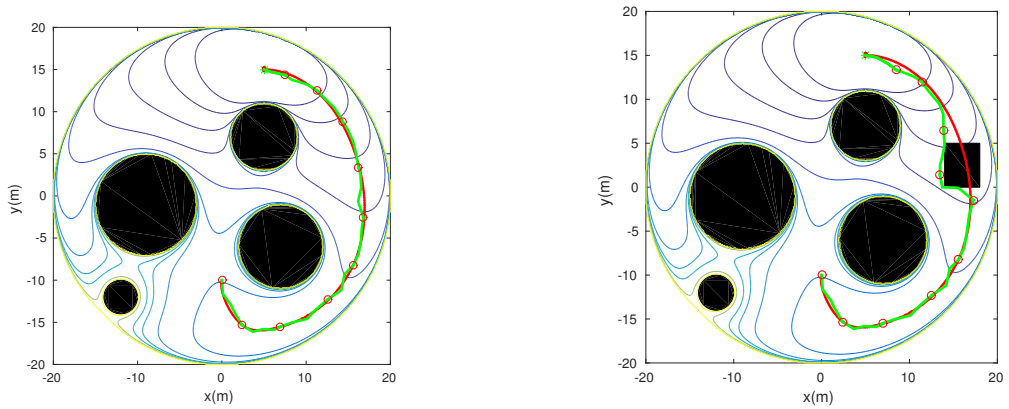



Figure 11: Gazebo simulation of a UAV in a patrolling task: (a) Snapshot of the simulation; (b) planned path without unmodelled obstacles and no-fly zones; and (c) path in the presence of new obstacles and a no-fly zone (blue circle). In (b) and (c) the vector field is represented by the arrows.




(a) (b)

Figure 12: Simulations showing a robot following a navigation function (a) without and (b) with an unmodeled, rectangular shaped obstacle.

# References


[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of 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 *Proceedings of the IEEE International 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, August 2010.

[5] 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.

[6] 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.

[7] M. Owen, R. W. Beard, and T. W. McLain, "Implementing dubins airplane paths on fixed-wing UAVs," in *Handbook of Unmanned Aerial Vehicles*. Springer, 2014, pp. 1677–1701.

[8] J. M. Esposito and V. Kumar, "A method for modifying closed-loop motion plans to satisfy unpredictable dynamic constraints at runtime," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, 2002, pp. 1691–1696.

[9] 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 *Proceedings of the IEEE International Conference on Decision and Control*, vol. 3, 2003, pp. 2926–2931.

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

[11] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

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

[13] ——, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proceedings of the IEEE International Conference on Decision and Control*, 2010, pp. 7681–7687.

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

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

[16] 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.

[17] X. Lan and M. Schwager, "Planning periodic persistent monitoring trajectories for sensing robots in Gaussian random fields," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2013, pp. 2415–2420.

[18] 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 International Journal of Robotics Research*, vol. 28, no. 6, pp. 685–700, June 2009.

[19] C. Belta, V. Isler, and G. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, Oct 2005.

[20] 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 *Proceedings of the IEEE International Conference on Mechatronics and Automation*, 2013, pp. 519–524.

[21] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space voronoi bias," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 2061–2067.

[22] 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.

[23] 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.

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

[25] 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.