Learning vector fields through persistent exploration for robot motion control[⋆]

Wilson S. F. Júnior[*] **Neemias S. Monteiro**[*]
Vinicius M. Gonçalves[*]

[*] *Graduate Program in Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil (e-mails: salomao4@ufmg.br, neemias@ufmg.br, mariano@cpdee.ufmg.br).*

Abstract:
Vector field is a well-established technique for performing mobile robot navigation. However, in environments with obstacles, vector fields may have their performance compromised, inducing trajectories that lead the robot to be trapped in a certain region. Therefore, this article presents a new approach, based on reinforcement learning, for interactive learning of vector fields. As a result, this approach provides a vector field: (i) free of spurious equilibria, and (ii) optimal concerning the length of the path traveled. Due to an appropriate initialization of the vector field, the approach can be used to solve tasks in environments with few obstacles even without having too much learning time. Simulations are implemented to validate the proposed methodology.

Keywords: Eikonal equation; Motion planning; Obstacle avoidance; Optimization; Reinforcement learning; Vector fields.

1. INTRODUCTION

The technique of artificial vector fields is widely used in the navigation of mobile robots (Goncalves et al., 2010; Beard and McLain, 2012; Zhao et al., 2018; Rezende et al., 2020). It stands out for integrating the motion planning and control stages into a single structure. Furthermore, approaches based on vector fields are relatively easy to understand and implement. When used, vector fields have the function of providing the speed or acceleration inputs to be applied in a robot to perform a task (Goncalves et al., 2010). For example, for a planar mobile robot, the vector field provides the velocity input $u(q) = \dot{q}$ for a given configuration $q = [q_1\ q_2]^T$.

Despite the mentioned advantages, in some situations, it can be difficult to build appropriate vector fields, especially in environments with the presence of obstacles. A recurrent method for avoiding obstacles in vector fields is the inclusion of repulsive components based on distance functions (Leitmann and Skowronski, 1977; Khatib, 1986). Another alternative to avoid obstacles is using optimization (Kanoun et al., 2011; Gonçalves et al., 2016; Júnior and Gonçalves, 2018). However, the usage of repulsive fields or optimization will often result in the appearance of spurious equilibrium points. Thus, possibly, the vector field can induce trajectories that lead the robot to be trapped in certain regions (Panagou, 2014). Moreover, the generated paths may be inadequate, i.e., too long.

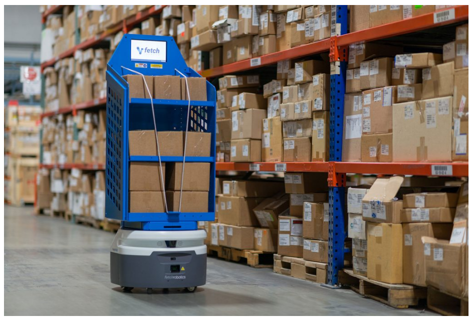

Fig. 1. The technique proposed in this article is applicable in situations where the robot must perform the same task multiple times from different initial configurations, such as shown in the image, in which the robot moves through a warehouse and must take pieces to a storeroom. The image was taken from[1].

In this article, an approach is proposed to build vector fields *on-the-fly* based on the robot's interactions with the environment. The technique of learning through interactions is called *reinforcement learning* (Watkins and Dayan, 1992; Kaelbling et al., 1996), in which an agent, in this case, a robot, seeks to improve its behavior through the acquisition of new knowledge from its interactions with the environment. The agent learns from a series of positive (rewards) or negative (punishments) reinforcements, that is, through the success or failure obtained. Generally, the main objective of this type of learning is to minimize a cumulative cost/maximize a cumulative reward measure obtained by the agent. Successful learning will result in the execution of rational control actions (also known as *policy*) by the agent, given its current state (George and Luger, 2013; Russell and Norvig, 2016). Some works that describe,

[⋆] The authors would like to thank the agencies CNPq, CAPES, FAPEMIG (under Grant APQ-02144-18), and the Graduate Program in Electrical Engineering at UFMG, for their financial support.

[1] `https://www.roboticsbusinessreview.com/supply-chain/autonomous-mobile-robots-changing-logistics-landscape/`

broadly, the use of reinforcement learning in robotic applications can be seen at Kober et al. (2013); Kormushev et al. (2013); Polydoros and Nalpantidis (2017); Nair et al. (2018); Akalin and Loutfi (2021).

Thus, with the use of a technique based on reinforcement learning, it is expected that the constructed vector field, after many repetitions of the task, will be better than the one obtained previously. In this way, the robot will learn the parameters for building the vector field as it explores the environment and performs the tasks. The learning strategy is based and formalized in a particular case of a result presented in Gonçalves (2021).

With the approach proposed in this article, the robot will learn incrementally, while exploring the environment and performing its task, a vector field with two valuable characteristics: (i) free of spurious equilibria, and (ii) optimal concerning the length of the path traveled (although the methodology presented here can be adapted for a different metric).

It is important to emphasize that even without having too much learning time, the field will be able to solve simple tasks in environments with few obstacles because it is initiated appropriately, that is, directed to the target. In this way, the proposed vector field (i.e. the policy) does not start entirely inappropriate and without information, as is common in several techniques/applications with reinforcement learning. This will also speed up the learning process as it will be shown in Subsection 3.3.

The obstacle avoidance of the proposed strategy is done by modifying an initial control action, not necessarily safe, using optimization techniques. This ensures that, even without a very advanced learning process, the modified control action is always safe.

The approach is suitable for applications where the robot performs the same task multiple times from different initial configurations. This situation can be illustrated by a mobile robot, Fig. 1, that moves through a warehouse and needs to take pieces from different initial configurations (positions) to a common configuration (the storeroom). In situations where the robot is required to perform a task a few times, this strategy will not be adequate as the robot will not have an opportunity to learn and thus it may provide bad policies.

The article is organized as follows: in section 2, a general explanation is made about the problem to be addressed and the proposed solution for learning target-oriented and collision-free policies. In section 3, the proposed approach is validated in a simulated environment in MATLAB software. In section 4, conclusions and suggestions for future work are presented.

2. THEORETICAL DESCRIPTION OF THE SOLUTION

2.1 Problem proposal

Let $\mathcal{Q} \subseteq \mathbb{R}^n$ be the configuration space of a robot, and $q \in \mathcal{Q}$ its configuration. We assume this robot is represented by a first-order kinematical system ($\dot{q} = u$) and that it initiates at configuration q_0, with the goal being to control this robot to a target set $\mathcal{Q}_{tg} \subset \mathcal{Q}$. We also assume[2] a constant Euclidean norm for u, $\|u\| = 1$, and we define $c : \mathcal{Q} \longmapsto \mathbb{R}^+$ as the cost per movement at configuration q. Therefore, we can formulate an optimal control problem as:

Problem 1.

$$\begin{aligned} \min_{u} \quad & \int_0^{\infty} c(q(t))\|dq(t)\| \\ \text{subject to} \quad & \dot{q}(t) = u(t),\ \|u(t)\| = 1 \\ & q(0) = q_0,\ q(\infty) \in \mathcal{Q}_{tg}. \end{aligned} \tag{1}$$

This formulation expects that q will reach $\mathcal{Q}_{tg}$ sometime, not defining a time constraint to do the task.

Considering an environment with obstacles in which we want to obtain the shortest path (in the Euclidean metric), we define $\mathcal{Q}_{free} \subseteq \mathcal{Q}$ as the collision-free sub-space. In this article we will be focusing on environments in which the obstacles are previously known, considering as future work the evaluation regarding the capability of the algorithm in scenarios where obstacles are added during the learning phase. We may use a special case of $c(q)$ to model this environment as:

$$c(q) = \begin{cases} 1, & \text{for } q \in \mathcal{Q}_{free} \\ \infty, & \text{for } q \notin \mathcal{Q}_{free}. \end{cases}$$

In this way, with the solution of Problem 1, the robot will follow the shortest path (in the Euclidean metric) from q_0 to $\mathcal{Q}_{tg}$ moving only inside $\mathcal{Q}_{free}$. It is guaranteed that Problem 1 induces a control action that is free of spurious equilibrium points. Let $u^*(q)$ be the optimal control action obtained from Problem 1, also known as optimal *policy*, $\pi(q)$. We define $q^*(t, q_0)$ as the robot evolution over the time t, starting from its initial configuration q_0, using the optimal policy $\pi(q)$. Thus, we can define $V(q_0) : \mathcal{Q}_{free} \longmapsto \mathbb{R}^+$ as the *value function* of configuration q_0. $V(q_0)$ can be calculated with $V(q_0) = \int_0^{\infty} c(q^*(t, q_0))\|dq^*(t, q_0)\|$, assigning to each configuration q_0 its optimal cost. Using the Hamilton-Jacobi-Bellman equation, it is possible to show that $V(q_0)$ satisfies the so-called *Eikonal equation* (Bardi and Capuzzo-Dolcetta, 2008):

$$\begin{cases} \|\nabla V(q_0)\|^2 = c(q_0)^2 \\ V(q_0) = 0, q_0 \in \mathcal{Q}_{tg} \end{cases} . \tag{2}$$

Solving the Eikonal equation (2), it is possible to get the policy $\pi(q)$, given by $\pi(q) = u^*(q) = -\frac{\nabla V(q)}{c(q)}$. As this policy has to be executed in a general configuration q, henceforth, we will use only the $V(q)$ notation instead of $V(q_0)$.

Unfortunately, equation (2) is not easy to solve analytically (Vavryčuk, 2012; Jahanandish, 2010), therefore, we will propose a framework to estimate both $V(q)$ and $\pi(q)$ while the robot moves inside $\mathcal{Q}_{free}$. This framework has a connection with machine learning techniques, more specifically in the context of reinforcement learning. Therefore,

[2] This constraint is used in the problem formulation for the sake of simplicity of the resulting control action, however, once we obtain the optimal solution, it keeps being optimal for the problem obtained by dropping out this constraint. Even if we scale the optimal solution u^* (with a positive factor), it is optimal for the modified problem.

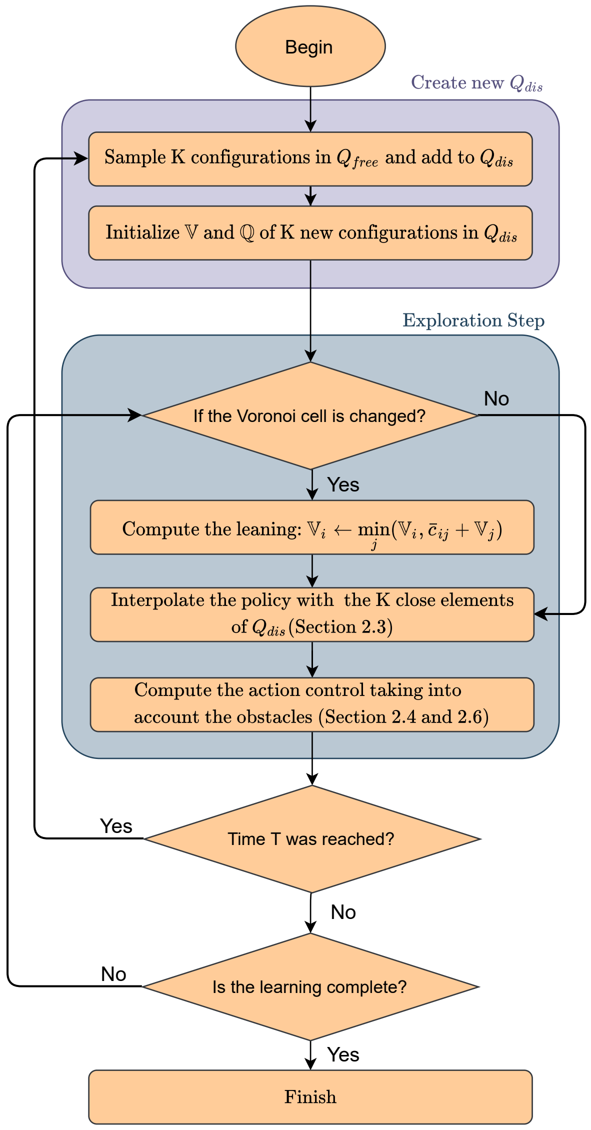
Begin
Create new Q_{dis}
Sample K configurations in Q_{free} and add to Q_{dis}
Initialize $\mathbb{V}$ and $\mathbb{Q}$ of K new configurations in Q_{dis}
Exploration Step
If the Voronoi cell is changed?
No
Yes
Compute the leaning: $\mathbb{V}_i \leftarrow \min_j(\mathbb{V}_i, \bar{c}_{ij} + \mathbb{V}_j)$
Interpolate the policy with the K close elements of Q_{dis} (Section 2.3)
Compute the action control taking into account the obstacles (Section 2.4 and 2.6)
Yes
Time T was reached?
No
No
Is the learning complete?
Yes
Finish

Fig. 2. Flowchart diagram for the framework presented in this article.

we expect the robot to *learn* the optimal path as it moves in the environment.

2.2 Learning strategy

We will now present a framework to estimate the policy $\pi(q)$ and the value function $V(q)$ while the robot moves inside $\mathcal{Q}_{free}$.

Figure 2 shows the flowchart representing the framework. Each subsection ahead will explain and depict the steps inside this flowchart.

First of all, it is necessary to sample $\mathcal{Q}_{free}$ in a finite set of configurations which we will define as $\mathcal{Q}_{dis}$. Any sampling strategy can be used as long as there is at least one sampled point in $\mathcal{Q}_{tg}$, but we will discuss in detail the sampling strategy in section 2.5. Let $\mathbb{q}_i$ be an element of $\mathcal{Q}_{dis}$, we will estimate the value function of this configuration, $\mathbb{V}_i \approx V(\mathbb{q}_i)$, and the policy of this configuration, $\mathbb{\pi}_i \approx \pi(\mathbb{q}_i)$.

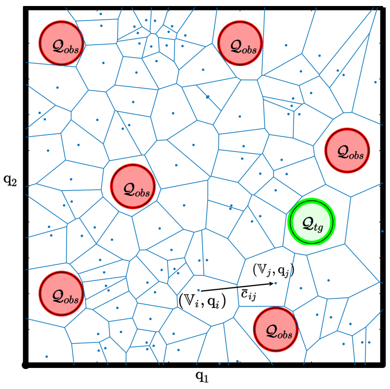

Fig. 3. Representation of the Voronoi diagram in an environment with obstacles ($\mathcal{Q}_{obs} = \mathcal{Q} - \mathcal{Q}_{free}$). The blue dots represent the sites denoted by the triple ($\mathbb{V}_i$, $\mathbb{q}_i$, $\mathbb{\pi}_i$).

Generally speaking, this sampling strategy will create a Voronoi diagram in the configuration space, where the site of each Voronoi cell is an element of $\mathcal{Q}_{dis}$. It is important to stress that it is not necessary to calculate the Voronoi regions explicitly. It is only necessary that the robot can identify which configuration of $\mathcal{Q}_{dis}$ is the closest one to the current configuration $q(t)$. Fig. 3 shows an example of this diagram.

Since the robot will be in a Voronoi cell at each instant, as it moves, with a strategy that will be presented in section 2.6, sometimes it will change its current Voronoi cell. This can be identified by simply checking if the closest configuration in $\mathcal{Q}_{dis}$ to q has changed.

Therefore, every time the robot moves from a Voronoi cell in which the site is $\mathbb{q}_i$ to one in which the site is $\mathbb{q}_j$, the equation (3) will be used to improve the estimate of $\mathbb{V}_i$:

$$\mathbb{V}_i \leftarrow \min_j(\mathbb{V}_i, \bar{c}_{ij} + \mathbb{V}_j), \tag{3}$$

in which $\bar{c}_{ij}$ is an estimate of the optimal cost in the path beginning at $\mathbb{q}_i$ and ending at $\mathbb{q}_j$, i.e. an estimate of $V(\mathbb{q}_i) - V(\mathbb{q}_j)$. In our case, since $c(q) = 1$ in the $\mathcal{Q}_{free}$, we can use $\bar{c}_{ij} = \|\mathbb{q}_i - \mathbb{q}_j\|$. A particular case of the results obtained in Gonçalves (2021) shows that this iteration converges to the solution of the Bellman equation $\mathbb{V}_i = \min_j(\bar{c}_{ij} + \mathbb{V}_j)$, as long as the initial value of $\mathbb{V}_i$ overestimates the real value. The convergence is monotonic because the $\mathbb{V}_i$'s never increase. This discrete dynamic programming problem is a discrete approximation of our continuous problem.

Every time $\mathbb{V}_i$ updates through equation (3), if $\mathbb{V}_j + \bar{c}_{ij} < \mathbb{V}_i$, the policy $\mathbb{\pi}_i$ will be updated according to equation (4)

$$\mathbb{\pi}_i \leftarrow \frac{\mathbb{q}_j - \mathbb{q}_i}{\|\mathbb{q}_j - \mathbb{q}_i\|}, \tag{4}$$

see also Fig. 3 for a geometric explanation. These simple equations will be used to estimate $\mathbb{V}$ and $\mathbb{\pi}$. To perform this estimation properly, we propose to initialize both sets of values as follows:

- $\mathbb{V}_i = Kd(\mathbb{q}_i)$, where $d(\mathbb{q}_i)$ is the Euclidean distance of $\mathbb{q}_i$ to $\mathcal{Q}_{tg}$ disregarding the obstacles (naive) and $K \in \mathbb{R}^+$ is a scale factor ($K > 1$). K aims to overestimate the value function $\mathbb{V}_i$ initially, in order to ensure that when using the equation (3), it will be learned accordingly.

- $\mathbb{\pi}_i = \frac{\mathbb{q}_{tg} - \mathbb{q}_i}{\|\mathbb{q}_{tg} - \mathbb{q}_i\|}$, being q_{tg} any element of set $\mathcal{Q}_{dis}$ in $\mathcal{Q}_{tg}$. In this way, the initial policy will guide the robot towards the target.

It is important to mention that the parameter K can be chosen depending on the number of obstacles present in the environment. With a high number of obstacles, the overestimation of $\mathbb{V}_i$'s initial values needs to be higher than the scenario with none or few obstacles. For example, in the best case scenario, one without obstacles, $K = 1$ is sufficient.

This initialization has the advantage that the initial control action is sufficient in various scenarios, for example in scenarios without obstacles, or even, for simple environments with some obstacles. As mentioned, it may also speed up the learning process as it will be shown in Subsection 3.3. Thus, this initialization can be seen as an important quick start.

2.3 Continuous control action proposal

Up to now, even considering we have both sets $\mathbb{\pi}$ and $\mathbb{V}$ well estimated, they are discrete values, and to control the robot motion, we need a continuous and smooth control action. In order to solve this, we define the function $\tilde{V} : \mathcal{Q}_{free} \longmapsto \mathbb{R}^+$ as the estimate for $V(q)$. Let $\mathcal{P}(N, q)$ be the subset of the N closest elements of $\mathcal{Q}_{dis}$ to q. We propose to compute $\tilde{V}(q)$ as the average of the elements from this subset weighted by the inverse of the distance from each element of the subset to q:

$$\tilde{V}(q; \mathbb{V}, \mathbb{q}) = \frac{\sum_{i \in \mathcal{P}(N,q)} \frac{1}{\|q - \mathbb{q}_i\|} \mathbb{V}_i}{\sum_{i \in \mathcal{P}(N,q)} \frac{1}{\|q - \mathbb{q}_i\|}}. \tag{5}$$

With the $\tilde{V}(q)$ properly defined, we can calculate the control action as $u(q) = -\frac{\nabla \tilde{V}(q)}{c(q)}$. We consider that when $q = \mathbb{q}_i$ for some i, $\tilde{V}(q)$ is calculated using the limit, which is $\lim_{q \to \mathbb{q}_i} \tilde{V}(q) = \mathbb{V}_i$.

However, this calculation can have a problem if $\tilde{V}(q)$ has some noise. In this case, the gradient will not represent the real gradient of $V(q)$ correctly. Hence, we will propose a solution to estimate the policy directly. We define $\tilde{\pi} : \mathcal{Q}_{free} \longmapsto \mathbb{R}^n$ as the estimated policy $\pi(q)$, and similarly to equation (5), we will compute $\tilde{\pi}(q)$ as:

$$\tilde{\pi}(q; \mathbb{\pi}, \mathbb{q}) = \frac{\sum_{i \in \mathcal{P}(N,q)} \frac{1}{\|q - \mathbb{q}_i\|} \mathbb{\pi}_i}{\| \sum_{i \in \mathcal{P}(N,q)} \frac{1}{\|q - \mathbb{q}_i\|} \mathbb{\pi}_i \|} \tag{6}$$

and, at least at first glance, we can use it directly as control action, i.e. $u(q) = \tilde{\pi}(q)$. We again use the limit when $q = \mathbb{q}_i$, in which case $\lim_{q \to \mathbb{q}_i} \tilde{\pi}(q) = \mathbb{\pi}_i$.

In the scenario where $\mathcal{Q}_{dis}$ covers the entire space $\mathcal{Q}_{free}$, the function $\tilde{\pi}(q)$ converges to $\pi(q)$, when the values of set $\mathbb{\pi}$ are estimated satisfactorily. Thus, using this policy it would be possible to take the robot to $\mathcal{Q}_{tg}$ moving only inside $\mathcal{Q}_{free}$ while traveling the shortest distance. However, it is unfeasible in practice to sample the entire space $\mathcal{Q}_{free}$ and to wait for the perfect learning of the $\mathbb{V}$'s. In that case, the function $\tilde{\pi}(q)$ loses the collision avoidance guarantee and an alternative needs to be proposed.

2.4 Obstacles avoidance strategy

Until this moment, we estimated a policy $\tilde{\pi}(q)$ which can move the robot from q_0 to $\mathcal{Q}_{tg}$. As it was informed before, in practice this policy by itself is not enough to avoid obstacles in configuration space since it does not equate perfectly to the real $\pi(q)$. Aiming to solve this problem, we propose as an alternative the following optimization problem to compute a $\tilde{\pi}(q)$ that also avoids obstacles:

$$\begin{aligned} \tilde{\pi}_c(q) = \underset{u}{\text{argmin}} \quad & \|u - \tilde{\pi}(q)\|^2 \\ \text{subject to} \quad & A(q)u \leq b(q), \end{aligned} \tag{7}$$

in which $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}_+^{m \times 1}$ are part of the constraints that will be used to avoid the obstacles.

$\tilde{\pi}_c(q)$ is a "corrected" version of $\tilde{\pi}(q)$, ensuring the obstacles avoidance. Note that the problem is a strictly convex quadratic program, therefore it can be solved efficiently. Further, we will soon guarantee, by construction, that the feasible space is always non-empty. Thus, a solution u always exists and is unique, and $\tilde{\pi}_c(q)$ is a well-defined function.

The objective function tries to equate $\tilde{\pi}_c(q)$ and $\tilde{\pi}(q)$. Indeed, the optimization problem solution when the constraints are not active is exactly, $\tilde{\pi}_c(q) = \tilde{\pi}(q)$. With them active, the solution is the closest one to $\tilde{\pi}(q)$ respecting the constraints.

To formulate the obstacles avoidance as $A(q)u \leq b(q)$, we define $G(q) : \mathcal{Q} \mapsto \mathbb{R}_-^m$ as a non-positive function that will capture the opposite of the Euclidean distance to the obstacles. It does not need to be an accurate distance, but if it decreases when the robot departs from the obstacle and it is zero if and only if the robot is colliding with an obstacle, it can be used in this context. As shown by Kanoun et al. (2011), if $\dot{G}(q) \leq -\eta G(q)$, with η a positive scalar, and $G(q_0) \leq 0$, then $G(q(t)) \leq 0$ for $t > 0$, so the robot will not collide with the obstacles. Since $\dot{G}(q) = \frac{\partial G}{\partial q}(q)\dot{q}$ and $\dot{q} = u$, we conclude that $A(q) = \frac{\partial G}{\partial q}(q)$ and $b(q) = -\eta G(q)$. Note that $b(q) = -\eta G(q) \geq 0$, thus the feasible set $A(q)u \leq b(q)$ is always non-empty ($u = 0$ is always a solution).

Is important to stress that, another possibility also evaluated during the development of this work, is to add classic repulsive terms (Khatib, 1986) instead of using quadratic programming to compute $\tilde{\pi}_c(q)$, but they led to spurious equilibrium points more often than using the optimization method given in equation (7) (Panagou, 2014).

2.5 Sampling strategy

To perform the learning strategy, as mentioned in section 2.2, we need to create the set $\mathcal{Q}_{dis}$ by sampling $\mathcal{Q}_{free}$. We propose to repeat this procedure as the robot moves inside the environment by starting a sampling procedure inside the ball with radius D centered in its current configuration q at each T seconds. Thus, this proposal considers that the robot has a limited perception of the environment.

A problem with the approach of sampling points $\mathcal{Q}_{dis}$ of the set $\mathcal{Q}_{free}$ indiscriminately is the *curse of dimensionality*, in which, the problem can have a huge computational burden since it will have to deal with many points in the computation, especially for configuration spaces in higher dimensions. Thus, to make the problem computationally treatable and ensure a fair sampling of the space, this sampling procedure will make the following check: only sample if the number of points of $\mathcal{Q}_{dis}$ inside the ball is under a fixed number M and then we sample P points, assuring they have a minimum distance h among each other and among all other points in $\mathcal{Q}_{dis}$. Doing this procedure repeatedly ensures it will cover the set $\mathcal{Q}_{free}$.

Note that each time that new elements are sampled, the sets $\mathbb{V}$ and $\mathbb{\pi}$ must be increased, initializing their new elements as defined in section 2.2.

2.6 Exploration proposal

The entire framework presented until now requires the robot to move inside the environment to learn the set of values $\mathbb{\pi}_i$ correctly. This movement can be performed in many different ways, but we propose the following approach. Let $w \in \mathbb{R}^n$ be a disturbance input that will be added to equation (7). This disturbance aims to move the robot around the scenario, adding an *exploration* component to the robot motion control. Thus, the $\tilde{\pi}_c(q)$ has to be changed to:

$$\begin{aligned} \tilde{\pi}_c(q) = \underset{u}{\text{argmin}} \quad & \|u - (\tilde{\pi}(q) + w)\|^2 \\ \text{subject to} \quad & A(q)u \leq b(q). \end{aligned} \tag{8}$$

As explained for equation (7), the objective function of (8) will equate $\tilde{\pi}_c$ to $\tilde{\pi} + w(t)$ when there is no constraint active. By doing that, the robot will explore the environment, depending on the value of w and not only relying on the policy which can be not mature enough at the beginning.

There are some features that w must have. First of all, the maximum norm of w needs to decrease over time, since there is an expectation that the robot will learn the optimal policy as it moves inside the environment. Secondly, it has to reset to the maximum norm if the robot reaches a spurious equilibrium point. This disturbance will be the only way that the robot will have to move out from the spurious equilibrium points. Third, it will be constant for a period since our intent is not to make the robot move completely erratically inside the environment, which would happen if w changes very often, but ensure that it will move in a direction where it can learn the value function accordingly.

By using equation (8), once the robot reaches the target $\mathcal{Q}_{tg}$, probably the robot did not learn the policy well yet. We must make the robot move more in $\mathcal{Q}_{free}$ to ensure that the learning will continue. Thus, we must induce a movement that makes the robot go to another configuration, and then continue running the learning algorithm again. In practice, this could be done by making the robot perform another task or we can artificially induce this movement by considering only random movements.

3. SIMULATION RESULTS

We will present three simulations with common parameters to illustrate the proposed methodology.

3.1 Parameters and settings

We simulated the framework using a mobile robot with configuration $q = [q_1 \quad q_2]^T$ for Simulations 1 and 2 and $q = [q_1 \quad q_2 \quad q_3]^T$ for Simulation 3. This configuration is the robot position based on a fixed reference frame. The simulation was performed in MATLAB R2020b and we used a first-order kinematic model to comply with Problem 1. We ran the simulations into an Intel Core i3-2310M CPU @ 2.10GHz×4, with 8GB of RAM memory and running Ubuntu 18.04.5 LTS.

In all cases, we consider a punctual robot because we assume that the obstacles are already enlarged. All the obstacles have a circular/spherical shape, but with different sizes. The distance function to the obstacles were simply calculated as distances to the circles and lines (for the boundaries of the scenario). Finally all the scenarios contain only one target $\mathcal{Q}_{tg}$ represented by a circular/spherical shape.

We used $K = 10$ (subsection 2.2), $N = 5$ (subsection 2.3), $\eta = 0.1\ s^{-1}$ for the circular/spherical obstacles and $\eta = 1$ s^{-1} for the outer boundary (subsection 2.4). The sampling procedure is performed by choosing $D = 0.5\ m$, $T = 1\ s$, $M = 1$, $P = 1$ and $h = 0.8\ m$ (subsection 2.5).

The disturbance w is generated by sampling a vector from a multi-dimensional normal distribution in $\mathbb{R}^2/\mathbb{R}^3$ with mean equal to the zero vector and covariance matrix equal to the identity matrix. Then, this vector is normalized and scaled by a factor that initially is $2\ m/s$ and decreases linearly, reducing 20% at each $5\ s$ (subsection 2.6). Our control action $\tilde{\pi}(q)$ is always normalized, and thus it needs to be converted to a real speed. We choose that this speed is $1\ m/s$. Finally, we integrate using the explicit first-order Euler integration with $dt = 0.05\ s$.

Every time that the robot reaches $\mathcal{Q}_{tg}$ we "teletransport" the robot to a random configuration, using an uniform distribution. In a real application, it would be necessary to have a strategy to guarantee that the robot will have the opportunity to explore $\mathcal{Q}_{free}$. For each scenario, we left the computer simulating for the same amount of time.

3.2 Validation methodology

In order to validate the methodology and show that the robot indeed is learning, for each scenario, we sample one hundred points uniformly in $\mathcal{Q}_{free}$ and with them, we periodically perform the following *validation* procedure: we simulate the trajectory traveled by the robot using the policy $\mathbb{\pi}$ of different timestamps of the simulation and considering the initial configuration as each of these one hundred points. These trajectories also consider $w(t) = 0$. With these trajectories, we compute the distance traveled towards the target set and store these values. If the robot does not reach the target set (i.e the control action of that specific timestamp fails to guide the robot towards the $\mathcal{Q}_{tg}$), we consider this distance as ∞. With these one hundred distances, we compute two metrics: the *success rate* versus time, which is the percentage of them which is not ∞ (thus it measures how often the control action was able to solve the task) and the *log-average distance* of these numbers (thus it measures how optimal are these distances). We choose the log-average because we also considered the "infinite distances" in this average, and then the regular average would become infinite. The log-average is given by:

$$L = -\ln\left(\frac{\sum_i^{100} e^{-d_i}}{100}\right)$$

in which d_i is the i-th distance in meters, considered dimensionless when used in the average. We show these two metrics over time, but it is important to highlight that this time is *not* the time spent by the computer to simulate, but the *simulated "real" time*, which would mirror the time that a real robot would take to produce these results.

3.3 Results

Scenario 1: the scenario is shown in Fig. 4, along with the points $\mathbb{q}$ and vectors $\mathbb{\pi}_i$ [3] after two hours of simulated time ("real" robot time). This scenario aims to validate the proposal in a bi-dimensional environment with several obstacles, which can be challenging for the robot, especially using traditional vector field controllers. Fig. 5 shows the evolution of the previously mentioned metrics: the success rate and the log-average distance. The success rate is over 90% after 820 s, and it is even better, over 95%, after 1860 s. Thus, it takes less than 15 minutes to reach a good success rate and it takes only 31 minutes to be able to go from almost any configuration to the target. The time spent to calculate the control action is, on average, 5 ms. To highlight the importance of the proposed initialization (section 2.2), Fig. 6 shows the performance using a random initialization of the sets $\mathbb{V}_i$, $\mathbb{\pi}_i$. It shows that the robot takes much more time to achieve good results since only after approximately 3500 s (almost one hour) we have 90% of success rate.

Scenario 2: this scenario is also bi-dimensional, but it has only two obstacles, representing an easier environment. This scenario is represented by Fig. 7, along with the policies $\mathbb{\pi}$'s in the elements of $\mathcal{Q}_{dis}$, shown after two hours of simulated time ("real" robot time). From Fig. 8, for this easier scenario, the success rate achieves 100% after 100 s, i.e. it only requires 1 minute and 40 s to be able to go from any configuration to the target. The log-average distance required some more time to reduce and reaches the optimal time approximately after 200 s, or 3 minutes. The time spent to calculate the control action is, on average, 5 ms. With the random initialization of the sets $\mathbb{V}_i$, $\mathbb{\pi}_i$, the Fig. 9 shows that for this scenario the time spent to achieve 90% of success rate is almost 7000 s (almost two hours), much more than what is spent for the one with the proposed initialization.

Scenario 3: this scenario is shown in Fig. 10. It is the three-dimensional version of **Scenario 1**. Looking at Fig. 11, we can see that the success rate achieves 100% after 50 s, which is a good performance. However, the log-average distance shows that it required 1200 s (20 minutes) to stabilize. The time spent to calculate the control action is, on average, 5 ms.

[3] It can be seen that some arrows point to the obstacles, because the algorithm has not learned that policy correctly yet.

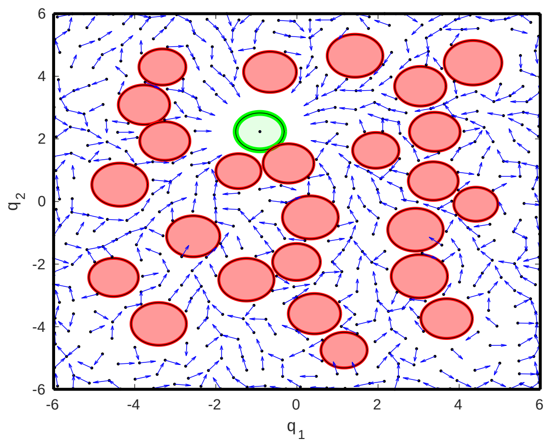

Fig. 4. **Scenario 1**, with several obstacles surrounding the target. $\mathbb{q}_i$, $\mathbb{\pi}_i$ are also shown. There are 496 elements in each set.

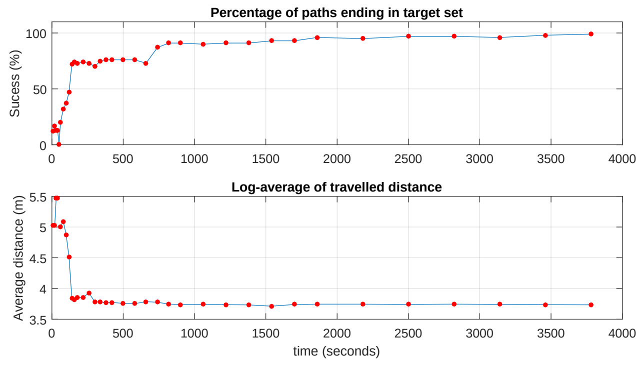

Fig. 5. Performance of **Scenario 1**. The superior subplot shows the success rate while the inferior subplot shows the log-average distance.

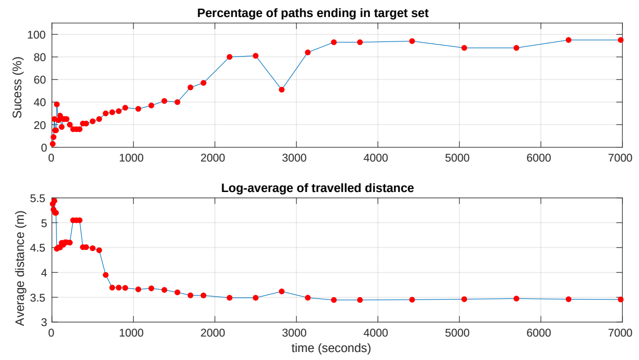

Fig. 6. Performance of **Scenario 1 - Random Initialization**. The superior subplot shows the success rate while the inferior subplot shows the log-average distance.

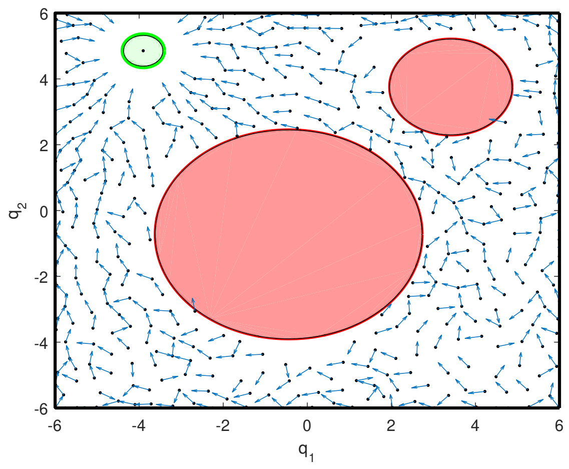

Fig. 7. **Scenario 2**, with several obstacles surrounding the target. $\mathbb{q}_i$, $\mathbb{m}_i$ are also shown. There are 405 elements in each set.

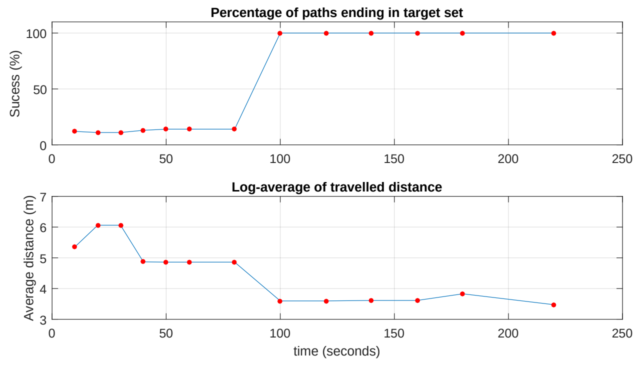

Fig. 8. Performance of **Scenario 2**. The superior subplot shows the success rate while the inferior subplot shows the log-average distance.

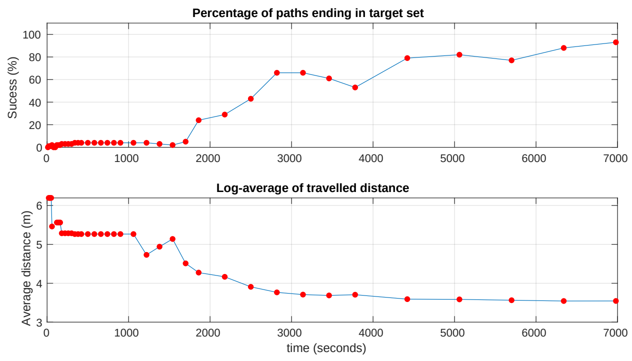

Fig. 9. Performance of **Scenario 2 - Random Initialization**. The superior subplot shows the success rate while the inferior subplot shows the log-average distance.

4. CONCLUSION

In this article, an approach was proposed to build vector fields *on-the-fly* based on the robot's interactions with the

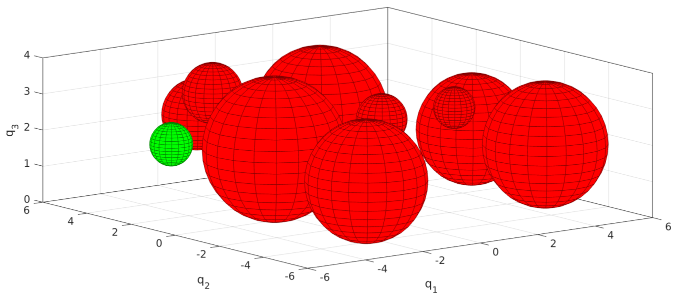

Fig. 10. **Scenario 3**, with several obstacles surrounding the target. The sets $\mathbb{q}_i$ and $\mathbb{m}_i$ are not represented because it is difficult to see them clearly in 3D. There are 1930 elements in each set.

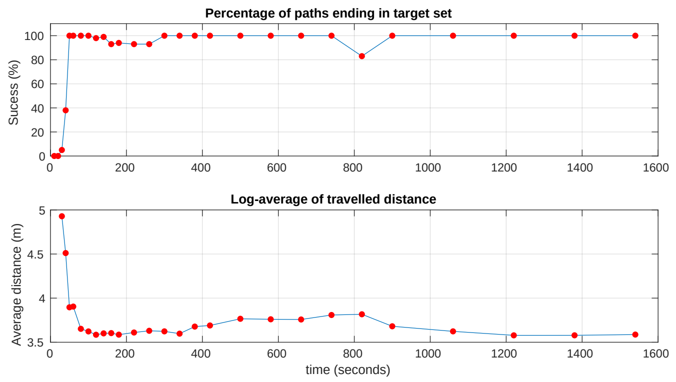

Fig. 11. Performance of **Scenario 3**. The superior subplot shows the success rate while the inferior subplot shows the log-average distance.

environment. As the robot explores the environment, it learns the parameters for building the vector field, the learning is improved, the success rate for reaching the target increases, and the distance traveled towards the target set decreases. The main advantage of using the proposed method is to build vector fields free of spurious equilibria, and optimal concerning the length of the path traveled. Even without having advanced learning, the field will be able to solve simple tasks in the proposed scenario. This is due to a proper initialization of the policy, that is, directed to the target.

As a suggestion for future work, the proposed approach can be applied in a real environment, or a more realistic robot simulator, for example, using CoppeliaSim or Gazebo. Another suggestion is to adapt the approach to environments with multiple targets and test it into robots with more degrees of freedom and different kinematic constraints, such as a mobile manipulator robot. We also aim to study the capability of this algorithm to react in scenarios where new obstacles are added during the learning.

REFERENCES

Akalin, N. and Loutfi, A. (2021). Reinforcement learning approaches in social robotics. *Sensors*, 21(4), 1292.

Bardi, M. and Capuzzo-Dolcetta, I. (2008). *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations.* Springer Science & Business Media.

Beard, R.W. and McLain, T.W. (2012). *Small unmanned aircraft: Theory and practice.* Princeton university press.

George, F. and Luger, S. (2013). *Artificial intelligence: Structures and strategies for complex problem solving.* Addison-Wesley.

Goncalves, V.M., Pimenta, L.C.A., Maia, C.A., Dutra, B.C.O., and Pereira, G.A.S. (2010). Vector fields for robot navigation along time-varying curves in n-dimensions. *IEEE Transactions on Robotics*, 26(4), 647–659.

Gonçalves, V.M., Fraisse, P., Crosnier, A., and Adorno, B.V. (2016). Parsimonious kinematic control of highly redundant robots. *IEEE Robotics and Automation Letters*, 1(1), 65–72.

Gonçalves, V.M. (2021). Max-plus approximation for reinforcement learning. *Automatica*, 129, 109623.

Jahanandish, M. (2010). A geometric-based numerical solution of eikonal equation over a closed level curve. *Iranian Journal of Science and Technology (Sciences)*, 34(1), 47–58.

Júnior, W.S.F. and Gonçalves, V.M. (2018). Formalismo para resoluçao de tarefas em robótica utilizando otimizaçao. *Anais do XXII Congresso Brasileiro de Automática (CBA) - 2018.*

Kaelbling, L.P., Littman, M.L., and Moore, A.W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 237–285.

Kanoun, O., Lamiraux, F., and Wieber, P. (2011). Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4), 785–792.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, 396–404. Springer.

Kober, J., Bagnell, J.A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 1238–1274.

Kormushev, P., Calinon, S., and Caldwell, D.G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3), 122–148.

Leitmann, G. and Skowronski, J. (1977). Avoidance control. *Journal of optimization theory and applications*, 23(4), 581–591.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6292–6299. IEEE.

Panagou, D. (2014). Motion planning and collision avoidance using navigation vector fields. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2513–2518. IEEE.

Polydoros, A.S. and Nalpantidis, L. (2017). Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2), 153–173.

Rezende, A.M., Gonçalves, V.M., Nunes, A.H., and Pimenta, L.C. (2020). Robust quadcopter control with artificial vector fields. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 6381–6387. IEEE.

Russell, S.J. and Norvig, P. (2016). Artificial intelligence: A modern approach.

Vavryčuk, V. (2012). On numerically solving the complex eikonal equation using real ray-tracing methods: A comparison with the exact analytical solution. *Geophysics*, 77(4), T109–T116.

Watkins, C.J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4), 279–292.

Zhao, S., Wang, X., Lin, Z., Zhang, D., and Shen, L. (2018). Integrating vector field approach and input-to-state stability curved path following for unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(8), 2897–2904.