Online learning control for path-aware global optimization with nonlinear mobile robots

Tudor Sântejudean, Lucian Buşoniu

Technical University of Cluj-Napoca, Memorandumului 28, 400114 Cluj-Napoca, Romania
tudorsantejudean@gmail.com, lucian@busoniu.net

Abstract

Consider a robot with nonlinear dynamics that must quickly find a global optimum of an objective function defined over its operating area, e.g., a chemical concentration, physical measurement, quantity of material etc. The function is initially unknown and must be learned online from samples acquired in a single trajectory. Applying classical optimization methods in this scenario would be highly suboptimal, since they would place the next sample arbitrarily far, without taking into account robot motion constraints, and would not revise the path based on new information accumulated along it. To address these limitations, we propose a novel algorithm called Path-Aware Optimistic Optimization (OOPA). We formulate the decision of which robot action to apply as an optimal control problem in which the rewards are refinements of the upper bound on the objective, weighted by bound and objective values to focus the search around optima. OOPA is evaluated in extensive simulations where it is compared to path-unaware optimization baselines, and in a a real experiment in which a ROBOTIS TurtleBot3 successfully searches for the lowest grayscale location on a 2D surface.

Keywords: optimization, learning control, mobile robots.

1. Introduction

Consider a problem where a mobile robot must find the maximum of an objective function defined over its operating area or volume. The function is initially unknown, so it must be learned from samples collected along a single trajectory of the robot. Different from classical optimization, the trajectory is important due to energy and time considerations. This scenario, which we call "path-aware global optimization", can be applied to find for instance the maximal density of sea bottom, surface, or water-column ocean litter (see `http://seaclear-project.eu`), the maximal or minimal location of pollutant concentration, temperature, humidity etc. (Essa et al., 2020; Lilienthal and Duckett, 2004), the strongest-signal location for radio transmission (Fink and Kumar, 2010; Busoniu et al., 2020), the largest sand height on the seabed for dredging, or the maximal or minimal forest density (Sankey et al., 2017), among others.

Local optimization methods like gradient ascent (Nocedal and Wright, 2006), which iteratively update a single point, can solve path-aware optimization after being modified to approximate derivatives from samples, as done e.g. in zeroth-order optimization (Liu

et al., 2020). Our first baseline is such an approximate version of gradient ascent. All these methods however share a common limitation in that they only find a local optimum.

Global optimization techniques (Horst and Tuy, 1996) like branch-and-bound, see Lawler and Woods (1966), Chapter IV of Horst and Tuy (1996), or Bayesian optimization (Frazier, 2018; Shahriari et al., 2016) usually choose the next sample at a location arbitrarily far away from the current robot position. We focus here on an algorithm from the branch-and-bound class: deterministic optimistic optimization (DOO) (Munos, 2011, 2014), which makes a Lipschitz continuity assumption on the function. We consider a variant of DOO that at each step samples a point associated with the maximum of an upper bound computed based on the Lipschitz property and on the information from previously observed samples. We pick DOO because of its soundness: in its original variant, it guarantees global convergence at a tightly-characterized rate towards the global optimum (Munos, 2011).

However, in our path-aware setting, such a global optimizer would *overcommit*: from the information accumulated along the path, it may become clear that this next sample is now suboptimal, but the algorithm would still navigate there even though it no longer makes sense to invest the energy and time, see also Figure 1 on page 5 for an intuition. Designing an optimization algorithm that finds a global optimum without overcommitting is a nontrivial challenge.

Our paper tackles this challenge by making the following key contributions. First, we design a path-aware extension of DOO, by formulating the decision of which direction to go at each time step as an optimal *control* problem. The robot can have general nonlinear dynamics. To avoid overcommitment, unlike DOO, we do not travel the full path to the largest-upper bound point. Instead, we pick a nearby position, reachable by the robot in one step. To represent the aim of refining the upper bound around the optima, the reward in the control problem includes the volume by which each action is expected to lower the upper bound, i.e. the difference between the bound before, and after measuring the new sample. Since exact refinements would require knowing the future function samples, we must resort to approximations of these samples from known position-function value pairs. We use nearest-neighbor approximation for this purpose, leading to a *predicted* volume refinement. This refinement is weighted by the value of the bound and of the function itself at the current point, to focus the algorithm around the optima.

To solve the control problem of maximizing the cumulative rewards (weighted predicted refinements) along the trajectory, we use an online version of value iteration (Bertsekas, 2019) combined with interpolation to handle the general robot dynamics (Buşoniu et al., 2010), and using a small number of updates to prevent extrapolating too much from the available information (Busoniu et al., 2020). The overall algorithm obtained is called *path-aware optimistic optimization*, OOPA.

The second key contribution is to validate OOPA in extensive experiments, in which the key performance criterion is the path length taken to reach (close to) an optimum. We begin with simulations for a mobile robot with unicycle-like nonlinear dynamics, in which we study the impact of the tuning parameters of the algorithm and its robustness to errors in the Lipschitz constant used to compute the upper bounds. The accuracy of volume refinement predictions is studied. All of the above is done for a smooth, deterministic function with multiple optima. Then, the algorithm is validated for noisy and non-differentiable functions. Throughout the simulations, OOPA is compared with

two baselines: gradient ascent and the classical DOO variant explained above.

Finally, to show the feasibility of the method in real life, it is applied to a TurtleBot3 ground robot for a physical embodiment of the smooth objective function from the simulations, realized as the black level of a printed grayscale surface. The behavior of the real robot is similar to those seen in simulations, but interestingly, due to the limitations of the camera, which "flattens out" the region of grayscale values observed, the Lipschitz constant must be taken smaller in the real experiments.

Related work can also be found in other fields than classical optimization. For example, in artificial intelligence, bandit algorithms are a class of sample-based optimization of stochastic functions (Lattimore and Szepesvári, 2020). They also overcommit by sampling at arbitrary distances, and must typically sample at least once everywhere to start building their estimates. In robotics, mapping requires building a map of the environment, and leads to the famous SLAM problem (Durrant-Whyte and Bailey, 2006) when the location of the robot is also unknown. Informative path planning chooses the path of a robot so as to find a map or other quantity of interest in as few steps as possible (Binney et al., 2010; Binney and Sukhatme, 2012; Lim et al., 2015). Other variants, like coverage (Choset, 2001), aim to find a shortest path that examines the entire space using the robot sensors. Different from all these robotics paradigms, the aim here is not to build or sense the entire function, but rather just to find the optimum as quickly as possible.

Compared to our preliminary work in the conference paper (Santejudean and Busoniu, 2021)[1], the present paper provides both algorithmic and experimental novelty. Algorithmically, by exploiting interpolative value iteration, OOPA is generalized to nonlinear robot dynamics, whereas before the robot had to obey simple-integrator dynamics and its position had to stay on a grid so that exact value iteration could be run. Experimentally, all the simulations are performed for unicycle-like, nonlinear dynamics. More importantly, fully novel experiments (i.e. that were not given before even for linear dynamics) are reported: for noisy samples and non-differentiable functions, together with – crucially – the real TurtleBot experiments. Compared to the approach in (Busoniu et al., 2020), which aimed to maximize the amount of transmitted data over a wireless network, the robot model is similar but the problem is different: finding the maximal function value. This leads to a more involved optimal control problem.

In Section 2 we introduce the background on DOO followed, in Section 3, by the formal definition of the optimization problem as optimal control. Section 4 presents the OOPA algorithm used to solve this problem. Empirical results, both in simulations and with a TurtleBot3 robot are provided in the next section. Lastly, Section 6 summarizes the main points and gives further extensions for the OOPA method.

2. Preliminaries

DOO is an algorithm belonging to the branch-and-bound class that aims to estimate the optimum of a function $f : P \to \mathbb{R}$ from successive function evaluations. It sequentially splits the search space P into finer partitions and samples to expand further only those

[1] In turn, (Santejudean and Busoniu, 2021) is a synthesis of the BSc thesis of the first author, available at `http://rocon.utcluj.ro/files/santejudeanbsc.pdf`.

sets associated with the highest upper bound values on the objective function. After a numerical budget has been exhausted, the algorithm approximates the maximum as the location p with the highest f value evaluated so far. An assumption made by DOO is that there exists a (semi) metric over P, denoted by l, and that f is Lipschitz continuous w.r.t. this metric at least around its optima, in the sense:

$$f(p^*) - f(p) \leq l(p^*, p), \forall p \in P \tag{1}$$

where $p^* \in \text{argmax}_{p \in P} f(p)$. Note that for convenience, we will require here the property to hold for any pair $(p_1, p_2) \in P^2$:

$$|f(p_1) - f(p_2)| \leq l(p_1, p_2), \tag{2}$$

and the Euclidean norm weighted by the Lipschitz constant will be chosen as the metric l over P:

$$l(p_1, p_2) = M||p_1 - p_2||, \tag{3}$$

where M is the Lipschitz constant. Our method can be extended to any metric l.

We will use the Lipschitz continuity described in equations (1)-(3) and give a different approach to the partition splitting in DOO: the construction of a so-called "saw-tooth" upper bound (Munos, 2014). This upper bound function is defined as $B : P \to \mathbb{R}$ so that:

$$f(p) \leq B(p) \coloneqq \min_{(p_s, f(p_s)) \in S} [f(p_s) + l(p, p_s)], \ \forall p \in P \tag{4}$$

where p_s is a sampled point and $(p_s, f(p_s)) \in S$, denoting with S the set of samples (function evaluations) considered while building B. See Figure 2 for an example. At each iteration, the next state to sample is given by the formula:

$$p_+ \coloneqq \text{argmax}_{p \in P} B(p). \tag{5}$$

The algorithm iteratively samples points selected with equation (5). Note that B is lowered (refined) with each new sample gathered by the robot, implicitly via (4).

Beyond this classical DOO algorithm, the bound (4) will drive our entire method, and its implications in the problem definition and algorithm will be detailed in the following sections.

3. Problem definition and formulation using optimal control

Consider a mobile robot described by the n_p-dimensional positions $p \in P \subset \mathbb{R}^{n_p}$ ($n_p \geq 1$), auxiliary states $y \in Y \subset \mathbb{R}^{n_y}$ ($n_y \geq 0$) and control inputs $u \in U \subset \mathbb{R}^{n_u}$ ($n_u \geq 1$). Differently from our preliminary paper (Santejudean and Busoniu, 2021), where dynamics were linear and first-order, here we consider general nonlinear dynamics of the robot. To define these dynamics, we denote the overall system state by $x \coloneqq [p^T, y^T]^T$, where $x \in X \coloneqq P \times Y$. Note that y can add velocities, headings etc. to the system state x; if $n_y = 0$, the state signal consists of only robot positions and the agent moves with first order dynamics. We define next the dynamics, $g : P \times Y \times U \to P \times Y$, in the discrete-time setting:

$$[p_{k+1}^T, y_{k+1}^T]^T =: g([p_k^T, y_k^T]^T, u_k) \tag{6}$$

where k denotes the time step of the trajectory followed by the robot.

Denote as in Section 2 with $f : P \to \mathbb{R}$ the function our robot needs to optimize. Initially, f is unknown to the agent and must be learned online by sampling the states visited during a single trajectory run. Due to dynamics constraints (e.g. limited velocity), our robot can only sample at the next, $k + 1^{th}$-iteration the states reachable within one step from its current position p_k. These constraints, along with the goal of finding a maximum p^* while minimizing the trajectory length (which for constant robot velocity also translates to minimizing the travelling time) define our *path-aware global optimization* setting. Even though we aim to quickly find a global maximum, the algorithm naturally explores near-optimal regions around the global optima. Such behavior can be useful in problems often encountered in practice, when a mobile robot needs to find the optimum of a physical quantity such as the highest density of surface or underwater ocean litter, the maximal bandwidth location of a radio transmitter, areas with large pollutant concentration throughout a city, etc.

The solution we propose applies DOO to build and refine with each new sample gathered by the robot an upper bound approximator of f. Different from the classical DOO approach, the goal here is not to always sample the maximal B-valued positions at next iterations (steps), but rather exploit the underlying objective of refining the bound around the optimum. The reasoning is as follows: due to dynamics constraints, the agent cannot perform large steps throughout the search space to directly sample the maximal-B states. Moreover, by committing to such trajectories without changing direction until the maximal-B states are reached, the robot would not be using the information (samples) gathered across the committed path, likely leading at some point to suboptimal trajectories. We call this the *danger of overcommitment* and provide some intuition in Figure 1.

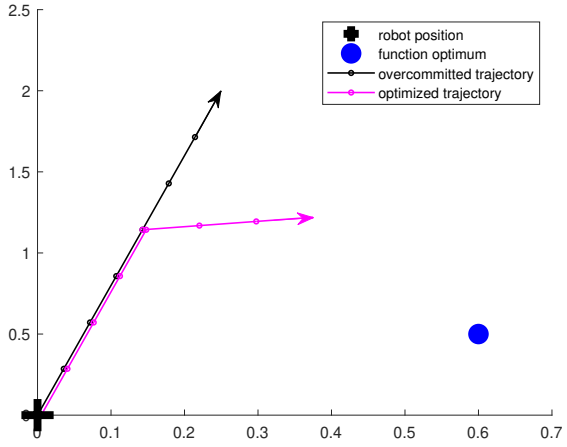

Figure 1: Based on the information available to the robot when it is at the black cross, a classical algorithm decides to check the point at the black arrow. However, samples (dots) accumulated along the black trajectory give more information about the optimum (blue disk), so it becomes apparent that continuing along this trajectory would waste energy and time. Thus, the robot should change heading to the magenta trajectory.

To address overcommitment, we make DOO aware of its path by formulating the task of choosing a decision at each step (e.g. the next waypoint to be followed, acceleration and torque of the robot, etc.) as an optimal control problem (OCP). As stated above, we do not aim to always sample the highest B-valued states; instead, the goal here is to lower (refine) the upper bound of f around optima to implicitly find p^*. Thus, we aim to maximize the refinements of the upper bound around points of interest that either: a) have high upper bounds and optimistically can "hide" maxima points, or b) have high function values that can lead to a maximum.

As the agent learns the function while searching for the optimum, our problem exhibits the exploration-exploitation tradeoff present in reinforcement learning. Specifically, overly encouraging a) will lead to excessive refinements in untraveled regions, less focused around high-value function points (too much exploration), while by overly encouraging b), the robot will overly refine areas where high function values were sampled and visit less untraveled regions that can possibly contain maxima (too much exploitation).

The OCP reward function defined next addresses this tradeoff. We consider the volume between the function upper bound and the horizontal plane (we use the term volume generically for any n_p; for $n_p = 1$ it reduces to an area). With each new sample the upper bound is lowered according to (4). The difference between the old and the new volumes is called volume refinement. Note that future samples are unknown and cannot be used to compute exactly this refinement, instead we must rely on approximations to predict it; see the example in Figure 2 for more intuition.

Formally, we define the reward as the volume predicted to be refined by taking action u in position p of state x, weighted by the average of the function value and its upper bound, both evaluated at p:

$$\rho(x, u) = \frac{\widehat{f}(p) + B(p)}{2} r(p, u) \tag{7}$$

where $\rho(x, u)$ is the reward function, $\widehat{f}$ is a function estimate obtained using a sample-based approximator, B is the upper bound function defined in (4), and $r(p, u)$ represents the volume predicted to be refined. The refinement is computed in the following way. First, denote with $S = \{(p_j, f(p_j)) | \, p_j = f(p_j),\ j = 1, \ldots, k\}$ the set of samples acquired up to step k. Compute next the upper bounds B_1 and B_2 using (4) and two slightly different sets of samples: B_1 with $S_1 = S \cup \{(p, \widehat{f}(p))\}$ and B_2 with $S_2 = S_1 \cup \{(p_+, \widehat{f}(p_+))\}$, where p_+ is the position of a possible next state $x_+ = g(x, u)$. The volume refined is determined through numerical integration over the difference in bounds $(B_1 - B_2)$ across the n_p dimensions of P.

A grid of points is used to determine approximately B_1 and B_2 and apply the integration over the bounds difference. This evaluation grid should not be taken too fine, as its values are mostly estimations of their true upper bound quantities. In most cases a rough approximation of B will suffice.

The terms $B(p)$ and $r(p, u)$ direct the refinements to locations with high upper bounds where optimistically a maximum might be situated, and respectively where the robot has the potential to significantly lower B (encouraging exploration). Factor $\widehat{f}(p)$ in (7) tells the robot to visit states closer to high function values (encouraging exploitation). As we do not know the function at future positions, $\widehat{f}(p)$ will be in most cases a prediction.

We compute a nearest-neighbor prediction, by taking the function value of the closest point to p that was already sampled. If state p was sampled before, its corresponding function value is directly taken (this should rarely happen in practice). We will use this simple approximator in experiments due to its light computational load, however, any sample-based approximator could be used for f.

Figure 2 gives an example of the reward calculation for a simple 1D case, where the evaluation point is denoted by p_k and the next point (corresponding to $g(x_k, u_k)$) is p_{k+1}. As p_k was already sampled, its function value $f(p_k)$ is chosen as the prediction of $\widehat{f}(p_k)$ and $\widehat{f}(p_{k+1})$ (since we are using the nearest-neighbor approximator and p_k is the closest sampled point to p_{k+1}). Note that $B(p_k) = f(p_k)$, as the bound evaluated in a sampled position equals the function value according to (4).

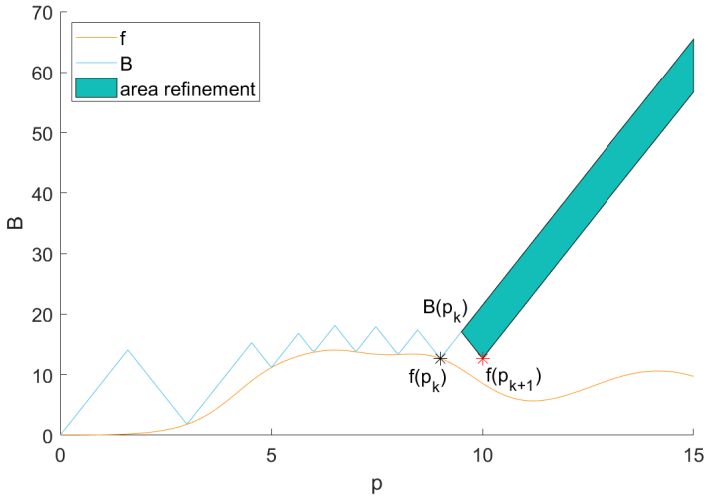

Figure 2: The sampled function is drawn with an orange line and the saw-tooth envelope represents its upper bound. Since p_k is the closest, sampled position to p_{k+1}, $f(p_k)$ (black star) is used to approximate $\widehat{f}(p_k)$, $\widehat{f}(p_{k+1})$ (red star) and to determine $B(p_k)$. The area refinement is colored in green.

More generally, note that the ideal reward function would use the true function values and updated B-values at the next steps. However, this is impossible in practice because it would require advance knowledge of the function samples. This is why we must resort to approximations of f and of the refinement.

The OCP objective is to maximize the long-term value function $V : X \to \mathbb{R}$. As the horizon is unknown to the robot (there is no telling how many steps it will take to reach the optimum), we define V in the infinite horizon setting:

$$V^h(x) = \sum_{k=0}^{\infty} \rho(x, u) \tag{8}$$

where $h : X \to U$ represents the control law and $u = h(x)$. This objective aims to maximize the upper bound based rewards, and we will experimentally investigate the

resulting travelled distance until the optimum.[2] We do not use a discount factor because upper bounds should eventually get close to the function, making refinements approach zero, and due to this long-term values are expected to be bounded. We are searching for an optimal policy, denoted by h^*, such that:

$$V^*(x) = V^{h^*}(x) \geq V^h(x), \forall x, h. \tag{9}$$

Define also the optimal Q-function $Q^*(x,u) = \rho(x,u) + V^*(g(x,u))$, which satisfies the Bellman equation:

$$Q^*(x,u) = \rho(x,u) + \max_{u'} Q^*(g(x,u),u') \tag{10}$$

This equation is usually solved by iterative Q-value iteration updates:

$$Q^+(x,u) = \rho(x,u) + \max_{u'} Q(g(x,u),u') \tag{11}$$

which converges to Q^*. Then, the optimal policy is given by $h^*(x) = \arg\max_u Q^*(x,u)$.

Due to the accumulation of new information, the OCP is a time-varying problem in which $\rho(x,u)$ changes with each new sample gathered. More specifically, the volume refinements r, function estimate $\hat{f}$ and upper bound B change as the agent follows its path. The changes in ρ are expected to be limited (since we only add one sample per step) and thus the solution of one problem should offer useful insight about the next. We exploit this feature in the algorithm from the next section.

A formal convergence analysis would be difficult to achieve due to the heuristics used. Nevertheless, intuitively, we expect the algorithm to asymptotically fill the entire search space with samples up to the resolution afforded by the discretized actions, which would mean that the optimum is also found with this resolution. The more interesting question is however how fast this optimum is found, which requires understanding the small-sample, nonasymptotic regime of the method.

In the following sections we will focus on exhaustive simulations and a real-robot experiment to validate the method.

4. Algorithm

In this section we will first introduce an approximator for the computation of the Q-function, then describe the optimal control algorithm and finally highlight the impact of its tuning parameters.

Unlike the method presented in Santejudean and Busoniu (2021), the dynamics in (6) allow the system state x to be continuous and include extra states in addition to p; in particular p can take any arbitrary position in P (it is not limited to a grid of points anymore). The action space remains a finite, discrete set and we will denote it by $U_d \subset U$. Due to the continuity of x, we must in this paper resort to approximations of the Bellman updates based on (10).

Multilinear interpolation is used further to represent $Q(x,u)$. To give some intuition, Figure 3 presents a simple interpolation example in which the discrete states $x_1, \ldots, x_N$

[2] We also tried to include an explicit travel cost term in the reward, but this did not work better.

comprise the interpolation grid. The points $x_1, \ldots, x_N$ are the centers of the N pyramidal basis functions (PBFs) $\phi_1, \ldots, \phi_N$ – the triangles in the 1D case of the figure. The action space is composed of D discrete controls $u_1, \ldots, u_D$. For each pair of i^{th}-PBF and j^{th}-discrete action we have a different parameter $\theta_{i,j}$, with a total of $N \cdot D$ such parameters. Note that the PBFs represent the weights with which each point on the grid participates to the approximation.

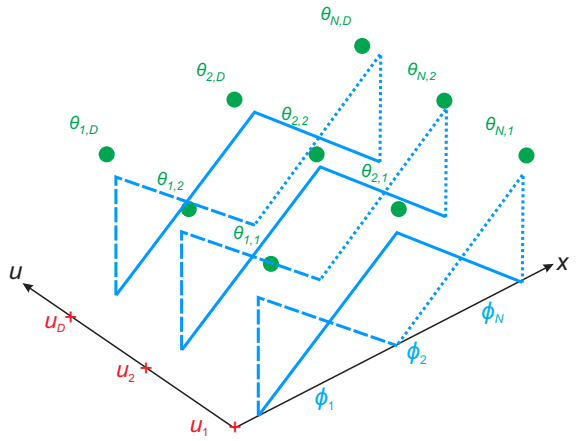

Figure 3: Illustration of a 2D interpolation example where the discrete states $x_1, \ldots, x_N$ make up the interpolation grid. The action space $U_d = \{u_1, \ldots, u_D\}$. For each pair (x_i, u_j) a parameter $\theta_{i,j}$ is considered. The pyramidal basis functions $\phi_1(x), \ldots, \phi_N(x)$, with the centers in $x_1, \ldots, x_N$, give the weights with which each point on the grid participates to the approximation.

We will expand the interpolation example to the $n_p + n_y$ dimensions of X and apply it to our method. Define first the state grid X_{grid}, composed of grids of points taken across each interval domain of the state variables in x. Part of X_{grid} is P_{grid}, which represents the discretized space of positions across P used for the B-related computations (lines $11 - 14$ of Algorithm 1). A linearly parameterized approximation of Q is obtained:

$$\widehat{Q}(x, u_j; \theta) = \phi(x)^T \theta_{\cdot,j} \tag{12}$$

where $\phi : X \to \mathbb{R}^{n_{\text{grid}}}$ represent the pyramidal basis functions and $\theta \in \mathbb{R}^{n_{\text{grid}}}$ their corresponding parameters, with n_{grid} denoting the total number of points taken on the state grid X_{grid}. The parameters $\theta_{i,j}$ are associated with the centers $x_i \in X_{\text{grid}}$ and actions $u_j \in U_d$ and act as weights to the PBFs $\phi_i(x)$. Note that x_i is the center of the PBF ϕ_i and thus the parameter $\theta_{i,j}$ can be seen as $\widehat{Q}(x_i, u_j)$, since $\phi_i(x_i) = 1$ and $\phi_{i'}(x_i) = 0, \ \forall\, i' \neq i$.

Finally, an approximate version of (11), also known as interpolative Q-iteration (Buşoniu et al., 2010), can be given:

$$\widehat{Q}(x_i, u_j; \theta_+) = \theta_{+,i,j} = \rho(x_i, u_j) + \max_{j'} \widehat{Q}(g(x_i, u_j), u_{j'}; \theta), \quad \forall i, j \tag{13}$$

where θ_+ represents the updated version of the current parameters θ.

The method in Algorithm 1 is called Path-Aware Optimistic Optimization (OOPA) and extends the previous version presented in Santejudean and Busoniu (2021) by gen-

eralizing g to arbitrary robot dynamics. Moreover, the OCP is approximately solved through the interpolative Q-iteration of (13) instead of the discrete-state Q-iteration.

The algorithm workflow is as follows. The robot starts in a state x_0 with no previous samples available, i.e. $S = \emptyset$. At each trajectory step k, the agent takes a sample $f(p_k)$ and adds it to the set S. We apply then an online version of the interpolative value iteration (13) to solve the OCP defined in Section 3. We do not fully solve the OCP at each step but rather approximate it by running a series of Q-iteration updates. The reasoning will be given later. Note that the Q function is estimated across an interpolation grid X_{grid} (line 16) using the rewards computed in the algorithm lines $10 - 15$, previously described in page 6.

Next, the agent uses explicit enumeration over U_d (line 20) to choose the action maximizing the approximated Q-values:

$$u_k = \arg\max_{u \in U_d} \widehat{Q}(x_k, u; \theta_k), \tag{14}$$

applies it following the dynamics in (6) and measures the next state x_{k+1}. This procedure continues until the total number of trajectory steps n is exhausted.

Algorithm 1 Path-Aware Optimistic Optimization (OOPA).

Input: motion dynamics g, discretized action set U_d, state interpolation grid X_{grid} (containing also P_{grid}), Lipschitz constant M, number of Q-iteration sweeps m, total trajectory steps n

1: initialize sample set $S \leftarrow \emptyset$
2: initialize $\theta_{0,i,j} \leftarrow 0$, $\forall x_i \in X_{\text{grid}}, u_j \in U_d$
3: measure initial state x_0
4: **for** each step $k = 0, \ldots, n-1$ **do**
5: sample $f(p_k)$, add pair $(p_k, f(p_k))$ to S
6: $\theta_0 = \theta_k$
7: **for** each sweep $l = 0, \ldots, m-1$ **do**
8: **for** all states $x_i \in X_{\text{grid}}$, actions $u_j \in U_d$ **do**
9: $x_{+,i} \leftarrow g(x_i, u_j)$
10: find $\widehat{f}(p_i)$, $\widehat{f}(p_{+,i})$ and $B(p_i)$ using M, (4) and S
11: compute B_1, B_2 across P_{grid} using M, (4), $S_1 \leftarrow [S, (p_i, \widehat{f}(p_i))]$ and
12: $S_2 \leftarrow [S_1, (p_{+,i}, \widehat{f}(p_{+,i}))]$, respectively
13: predict volume refinement $r(p_i, u_j)$ by applying numerical integration
14: over $(B_1 - B_2)$ across P_{grid}
15: find $\rho(x_i, u_j)$ using $r(p_i, u_j)$ and (7)
16: $\theta_{l+1,i,j} = \rho(x_i, u_j) + \max_{j'} \widehat{Q}(g(x_i, u_j), u_{j'}; \theta_l)$
17: **end for**
18: **end for**
19: $\theta_k = \theta_m$
20: $u_k = \arg\max_{u \in U_d} \widehat{Q}(x_k, u; \theta_k)$
21: apply action u_k, measure next state x_{k+1}
22: **end for**
23: return $\widehat{p^*} = \arg\max_{p_k} f(p_k)$, where $(p_k, f(p_k)) \in S, k = 1, \ldots, n$.

The tuning parameters of OOPA are as follows: the number m of Q-iteration sweeps that are applied at each step of the algorithm, the Lipschitz constant M, and the state grid X_{grid}.

Parameter m impacts the propagation of the upper bound-based rewards across the state grid considered while approximating Q. As mentioned above, we do not fully solve the OCP due (as a first reason) to the prohibitively expensive computations, but run m iterative updates of the form (13) on the interpolation grid X_{grid}. Fully converging to the true solution would require the true rewards to be known in advance, as B-values change (lower) with each new sample gathered by the robot, meaning that refinement gains at later steps will be smaller. Not knowing in advance the evolution of B at future steps will lead to overestimations of ρ in terms of refinement gains. This represents a second reason why the number of Q updates needs to be limited. We suggest taking $m \leq 5$.

Since the Lipschitz constant is generally unknown in practical scenarios, we must resort to empirical tuning to approximate it. The recommended approach is to start with a high M and decrease its value based on the feedback provided by the experiments. By taking a lower M compared to its true value, the bound inequality $f(p) < B(p)$ would not hold anymore, creating incorrect upper bounds (and volume refinements in (7)) that might break the algorithm.

Lastly, finer state grids (and discretized action spaces) are expected to lead the robot closer to the optimum, thus approximating p^* with better accuracy. We will study the impact of the state space discretization in the next section.

5. Results and discussion

Consider a robot with unicycle-like dynamics in (6):

$$p_{k+1} = p_k + T_s \cdot u_{k,1} \cdot [\cos(u_{k,2}), \sin(u_{k,2})]^T \tag{15}$$

where $T_s = 1\,\text{s}$ is the sampling period, while $u_{k,1}$ and $u_{k,2}$ respectively represent the velocity and robot heading, with $u_k = [u_{k,1}, u_{k,2}]^T$. The discretized set of actions is built using the velocities $u_{k,1} \in \{0, 1\}\,\text{m/s}$, where the nonzero velocity is along one of the headings $u_{k,2} \in \{0, \pi/4, \ldots, 7\pi/4\}\,\text{rad}$. Note that we have simplified equation (6) by taking no additional motion related states y in x, since such first-order dynamics help with the readability and are enough to illustrate our ideas. More accurate models are, of course, possible.

The nonlinearity in (15) comes from the usage of trigonometric functions in the unicycle dynamics of the robot. An example of linear dynamics would be the simple choice of up/down/left/right directions previously used in Santejudean and Busoniu (2021). Note that our method could easily accommodate other dynamics, for instance "true" unicycle dynamics with second order models (Lalish and Morgansen, 2008; Plaku et al., 2010).

The robot looks for an optimum in the space $P = [0, 2] \times [0, 2]\,\text{m}^2$. We define the grid P_{grid} of 21×21 equidistant positions (the points on the grid are spaced at $0.1\,\text{m}$ on both axes). Since $x = p$, $X_{\text{grid}} = P_{\text{grid}}$, and we will determine B and estimate Q on the same grid P_{grid}. Note that the dynamics are nonlinear and the next states can fall anywhere in X (they are not limited to X_{grid}). Thus, we must resort to the interpolative Q-iteration in (13) to estimate the action-value function.

The function sampled by the agent is represented by a sum of three radial-basis functions (RBFs) with the following coefficients: widths $b_i = [0.65; 0.65], [0.3; 0.3], [0.5; 0.5]$, respectively heights $h_i = [148.75, 255.0, 212.5]$ and centers $c_i = [0.375; 0.75]$, $[1.375; 1.75]$, $[1.625;\ 0.375]$ (see Figure 10 on page 17 for a contour plot). The global optimum is at $p^* = [1.375; 1.75]$ and has the value $f^* = 255.0$. The Lipschitz constant was chosen empirically as $M = 730$, so that it creates close to true upper bounds for f.

To study the performance of OOPA, throughout the simulations we will use as key criterion the trajectory length required by the agent to reach close to the optimum. We will also consider a *successful* approximation of p^* a trajectory that reaches at least as close as 0.1 m (equal to the grid step size) to p^*.

We will compare OOPA to two different baselines. The first one is classical DOO (CDOO) algorithm that, similarly to OOPA, uses the saw-tooth upper-bound (4). However, it fully commits to visit the maximum-B point (5) and thus changes its trajectory only once this point was reached. CDOO is not using the information gathered along the (possibly overcommited) trajectory until the next target point is reached.

The second baseline is Gradient Ascent. It applies Local Linear Regression on the closest N neighbors to the current position to create a local approximation plane of the sampled function. This plane is differentiated and the gradient direction that results is followed by the robot with unit velocity. This method has the natural limitation of converging only to local maximum points, likely at faster rates than the global methods above.

We organize the empirical results as follows. Section 5.1 provides experiments with a simulated mobile robot. It studies first the impact of the tuning parameters (Section 5.1.1) and the prediction accuracy of the volume refinements (Section 5.1.2). Section 5.1.3 compares the method against the CDOO and Gradient Ascent baselines. In Section 5.1.4 a study is provided to see how OOPA deals with noisy function evaluations, and Section 5.1.5 checks the behavior of OOPA for non-differentiable, pyramidal functions. Finally, we provide a real-life illustration of the method in Section 5.2.

5.1. Results with a simulated mobile robot

5.1.1. Influence of tuning parameters

The first parameter to be tuned is m, the number of Q updates applied at each step taken by the robot. In the following simulations we keep the total number of trajectory steps set to $n = 300$, equivalent to 30 m travel distance. On the setup defined above, we run the algorithm for $m \in \{1, 2, 3, 4, 5\}$ and study the maximum f-value sampled: $\overline{f} = \max_{p_k}(f(p_k))$, as well as the minimum distance until p^*: $\Delta_p = \min_{p_k}(||p^* - p_k||)$, where $(p_k, f(p_k)) \in S$. Another metric of interest is the minimum difference between the optimum and the values of f sampled so far: $\Delta_f = \min_{p_k}(f^* - f(p_k))$, with p_k having the same meaning as above. The robot's initial position is set to $p_0 = [0; 0]$.

Figure 4 shows that higher values of m ($m \in \{3, 4, 5\}$) perform well in finding the maximum with higher accuracy, while lower values ($m \in \{1, 2\}$) are suboptimal. The optimum was found sooner for $m \in \{3, 5\}$ (recall the meaning of "found" – 0.1 m close to p^*): in 10.3 m compared to the 25.1 m when $m = 4$. Among these values we choose $m = 3$, as this both provides better accuracy when searching for p^* and requires less computation (due to a lower number of Q updates). The intuition is that rewards based on the volume refinements need to be propagated across the state space, but not too

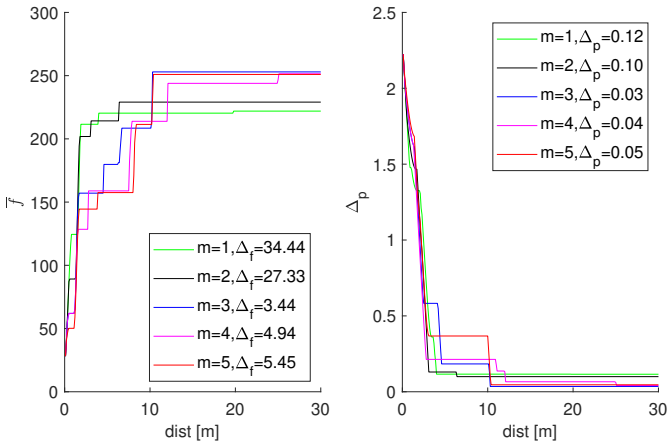

Figure 4: Illustration of the maximum value sampled so far (left) and the minimum distance to the maximum p^* (right), denoted with Δ_p; for $m \in \{1, 2, 3, 4, 5\}$. The legend on the left shows the minimum distance between the samples of f and its maximum value f^*, denoted by Δ_f.

much, since they are mostly approximate predictions and in most cases their values are overestimated, especially in the beginning of the experiment.

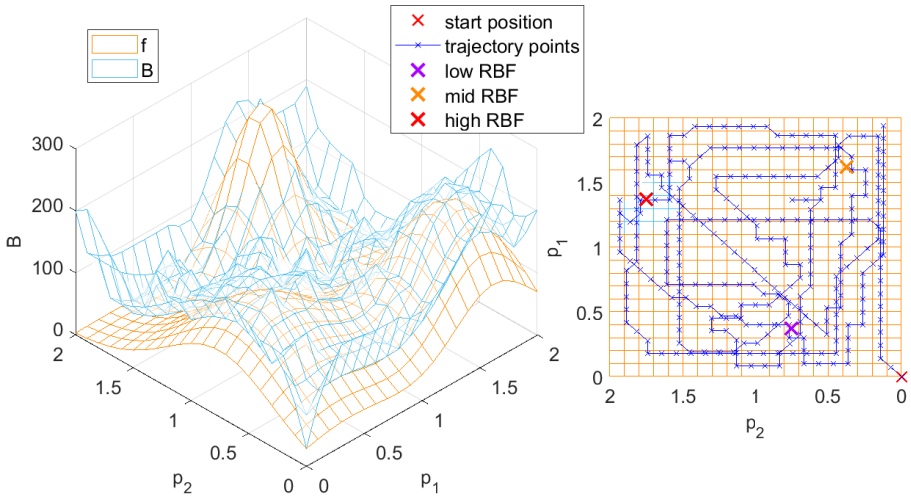

Figure 5: Left: The sampled function (in orange) is bounded from above by the refined B function (in blue), both evaluated on the same grid. Right: The sampling trajectory of the robot, drawn with blue 'x' and starting from the black 'x'.

Figure 5 (left) presents the refined upper bound of the sampled function built using (4) and the $n = 300$ samples acquired, for $m = 3$. The robot trajectory in Figure 5 (right) shows more steps being spent in the higher-valued regions where the three RBFs

(centered in the red, orange and magenta 'x') are situated. Thus, the choice of $m = 3$ gives balanced results in terms of both exploration and exploitation.

Next, we will check the robustness of the algorithm to overestimation or underestimation of the Lipschitz constant, to get some intuition on M. For this, let us take $M' = \lambda \cdot M$, where M is the previous value of 730, and $\lambda \in \{0.01; 0.05; 0.1; 0.5; 1; 2; 5; 10; 50; 100\}$. We run the algorithm with $m = 3$ and study its behavior using the metrics from above.

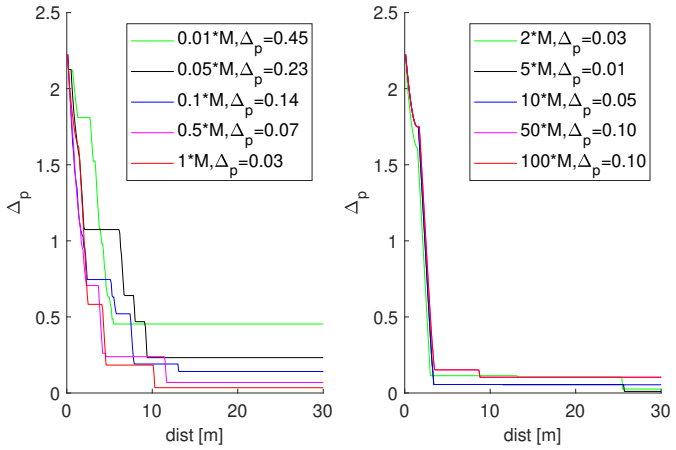

Figure 6: Robustness of the algorithm's performance to underestimation (left) and overestimation (right) of the Lipschitz constant.

Figure 6 shows that larger values of the Lipschitz constant always find p^* and do not break the algorithm. However, taking M' one or two orders of magnitude less than M, while upper bounds are no longer valid for $M' < M$, can lead to finding the optimum with less accuracy or even break the algorithm. Interestingly, even when M' was set to $M/2$ the optimum was still found with good accuracy. This might suggest that the values of the Lipschitz constant which estimate p^* well belong to a larger interval range than initially expected. Finally, we suggest starting with a high M and tune (lower) its value empirically, as this is safer.

It would be instructive to check if finer grids lead to better approximations of p^*. For this experiment the state space is split into grids of size $\{16^2, 21^2, 26^2, 31^2, 36^2\}$ and the parameters of the algorithm are set to $m = 3$ and $M = 730$. The step length of the robot is changed to the grid step size (δ) of each experiment respectively. Then, n will be the (rounded up) ratio between the initial trajectory length of 30 m and the grid step.

Figure 7 reports the results. Except for the case of 36^2, each grid leads the robot to an approximation error of the optimum (Δ_p) lower than the corresponding δ. The intuition for the "unlucky case" (36^2) is as follows. For smaller step sizes the number of m updates should also be increased so that the information spreads further over the grid of states (comparably to the larger discretization steps). However, this would increase the already high computational times (due to a large number of grid points over which the Q updates are run), possibly overly extrapolating the high estimates of the initial rewards. An alternative could be obtained by encouraging more the exploitation behavior of the algorithm, i.e. using a higher weight for $\widehat{f}$ compared to B in (7). Redoing the 36^2

experiment with the weights 0.75 and 0.25 for $\widehat{f}$ and B respectively, i.e. $\rho(x,u) = \frac{3\widehat{f}(p)+B(p)}{4} r(p,u)$, the maximum is found with accuracy of $\Delta_p = 0.02 < \delta_{36^2} = 0.057$ and $\Delta_f = 1.95$. This is a good result, as the approximation accuracy of p^* is lower than the step size of the experiment.

Figure 8 shows the computation times and leads to the conclusion that the average execution time has a quadratic growth with respect to the grid discretization.

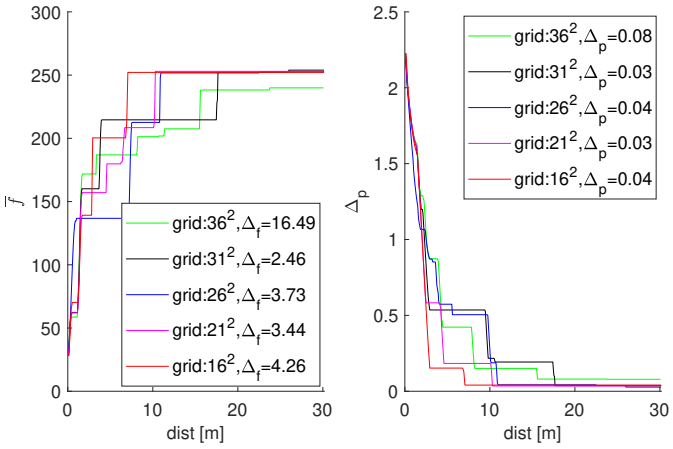

Figure 7: Performance of the algorithm for varying grid size.

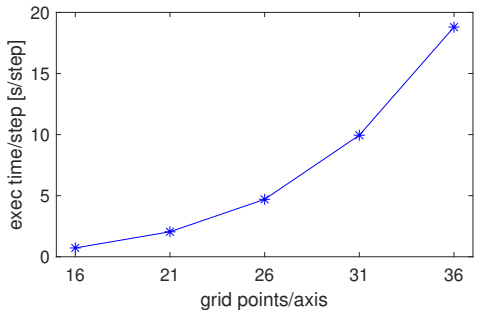

Figure 8: Execution time for varying grid size.

5.1.2. Prediction accuracy of the volume refinements

Since the reward values (7) are based on predictions of upper bound refinements, we will evaluate the accuracy of these predictions. Figure 9 compares at each step of the robot the predicted refinements with the actual ones (computed after the step has been executed and the new function value has been observed), for $m = 3$, $M = 730$ and 300 trajectory steps. The two values follow largely the same trend, with somewhat poorer accuracy at the beginning of the simulation, improving as the upper bound decreases and

the estimate $\widehat{f}(p_+)$ gets closer to the true $f(p_+)$. Thus, using the predictions is justified.

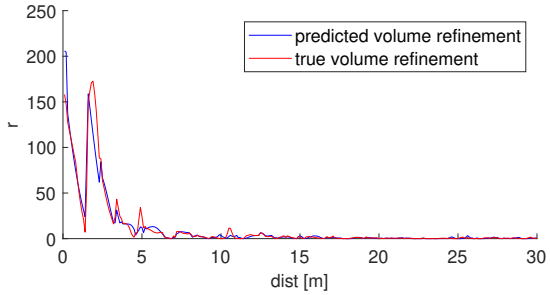

Figure 9: Prediction accuracy of the upper bound-based refinements.

5.1.3. Comparison to baselines

OOPA is next compared to the Classical DOO (CDOO) and Gradient Ascent baselines previously described at the start of Section 5. To have an informative comparison, we consider 5 equidistant starting positions placed along each side of the triangle linking the 3 centers of the RBFs. The robot will move with a fixed step of 0.1 m, equal to the step size of the 21^2 grid that is used in this experiment. Note that all methods sample the function with the same frequency, so that e.g. CDOO takes multiple samples along the (possibly overcommited) trajectory to the next chosen point. The number of nearest neighbors for the Gradient Ascent is set to $N = 4$ (at least three points are required to create the approximation plane).

Figure 10 reports the baseline comparison of OOPA. Near each starting position, drawn with 'x', the distance until the optimum for each method is given (when p^* is not found, the distance is replaced by a dash). The entries of each label correspond to OOPA, CDOO and Gradient Ascent, in this order, separated by slashes. For each algorithm, the number before the comma gives the distance traveled by the robot until reaching as close as 1 grid step size to p^*. After the comma, letters 'y' (for yes) or 'n' (no) show whether the maximum was found with $\delta = 0.1$ m accuracy. Note that for Gradient Ascent, we give the distance until convergence even when the algorithm reaches a local optimum (in which case 'n' is displayed).

According to Figure 10, the DOO-based methods always find the global optimum with δ accuracy. OOPA scores, on average, 43.9% less travel distance to p^* compared to CDOO. On the other hand, the Gradient Ascent converges to the optimum only in a fifth of the runs, however, it does so at faster rates compared to the DOO-based algorithms. More specifically, the gradient-based method converges to a local optimum unless it starts in a position close to p^*.

5.1.4. Influence of noisy function evaluations

We study the behavior of OOPA in the case of stochastic function evaluations. Each sample of f is multiplied before being used in the algorithm by a moderate noise coefficient z: $\widetilde{f}(p) = zf(p)$; where $f(p)$ is the expected value of $\widetilde{f}(p)$. In this experiment z is

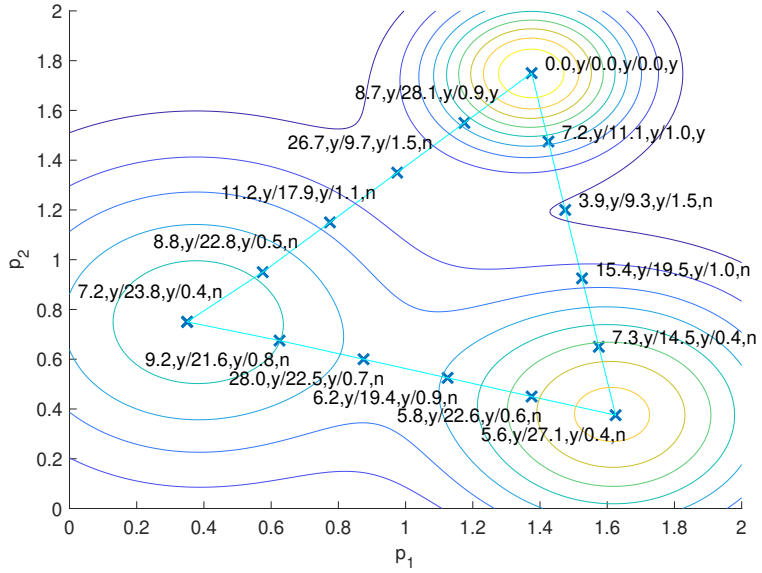

Figure 10: Baseline comparison of OOPA to the CDOO and Gradient Ascent methods.

drawn from a Rice distribution (Busoniu et al., 2020) with moderate variance, as most of the probability mass belongs to the interval $z \in [0.8, 1.2]$. This type of noise is realistic e.g. in radio transmission, where f could represent the signal-to-noise ratio.

Figure 11 reports the best function sample against the travel distance for a deterministic run and 25 runs with stochastic function evaluations of OOPA. The deterministic results are drawn with a blue line, while the red line shows, on average, the best function sample for each trajectory step in the stochastic case. The mean performance of the stochastic experiments is following to a good extent the shape of the deterministic scenario. Together with the 95% confidence interval (colored in grey), this highlights that the algorithm handles reasonably well a moderate noise level, even though it was not specifically designed for it.

5.1.5. Behavior with a non-differentiable function

In the simulations to follow, we keep the setup from above with the exception of the RBFs which are replaced by rectangular pyramidal functions (PBFs). The goal here is to check the behavior of the method when the objective function is non-differentiable. The parameters of the PBFs are: side lengths $b_i = [1.3; 1.3], [0.6; 0.6], [1; 1]$, heights $h_i = [148.75, 255.0, 212.5]$, rotation angles $a_i = [\pi/4, 2\pi/5, \pi/6]$ and centers $c_i = [0.375; 0.75]$, $[1.375; 1.75], [1.625; 0.375]$ (see Figure 12 for a representation of a generic pyramidal function and Figure 14 for a contour plot of the new f). The value of the global optimum and its position remain unchanged. The Lipschitz constant is computed starting from the tangent of the angles between the pyramid lateral faces and the horizontal plane and then tuned empirically to $M = 850$. This value of M creates close-to-true upper bounds for the new objective f.

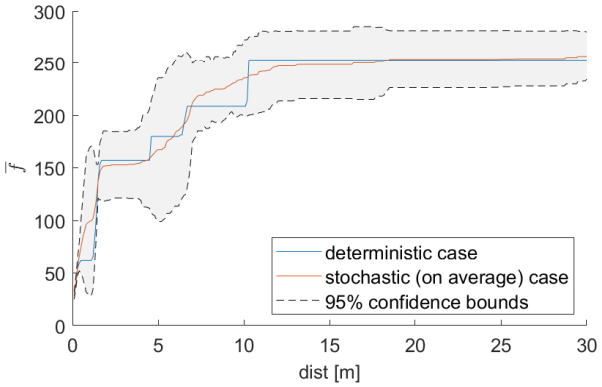

Figure 11: Illustration of the stochastic experiments with 95% confidence interval on the average performance of OOPA.

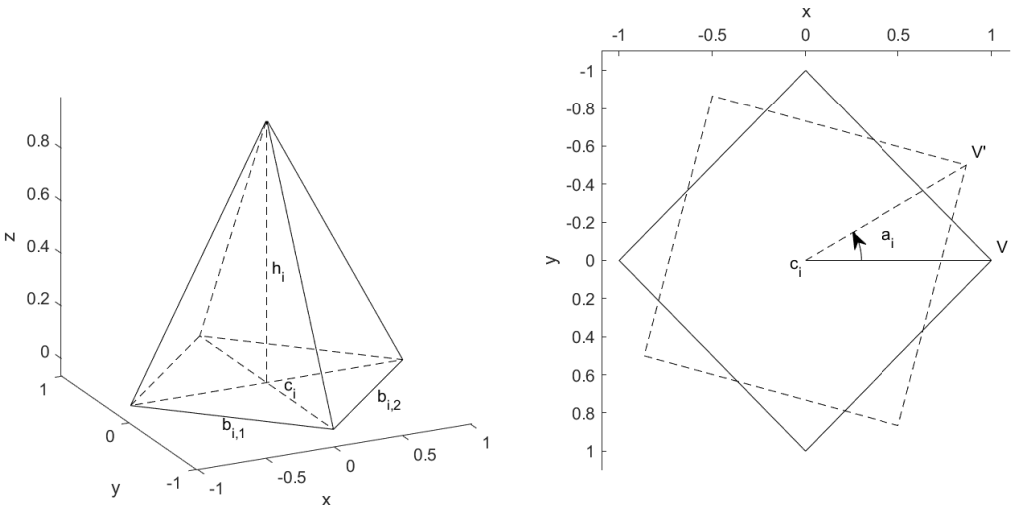

Figure 12: Representation of the parameters for a pyramidal function. Side lengths b_i, height h_i and center c_i are shown on the left plot. On the right figure, the angle a_i rotates clockwise the pyramid basis around its center starting from the parallel to the horizontal axis passing through c_i.

We start by verifying the behavior of the method in the event of underestimation/overestimation of the Lipschitz constant by setting $M' = \lambda \cdot M$, with $M = 850$ and $\lambda \in \{0.01; 0.05;\ 0.1; 0.5; 1; 2; 5; 10; 50; 100\}$. Figure 13 presents the outcome of the experiment. When M' took large values or was set to $M/2$, the optimum was found with the required δ accuracy. However, p^* is poorly approximated when M' is chosen one or two orders of magnitude less than M. Similar results to the case of the RBF-based f are obtained therefor.

Next, the baseline comparison is performed. Note that because f is now non-differentiable, applying the Gradient Ascent is mostly an heuristic attempt. Figure 14 shows that, except for a few outliers, the DOO-based algorithms find p^* with accuracy of

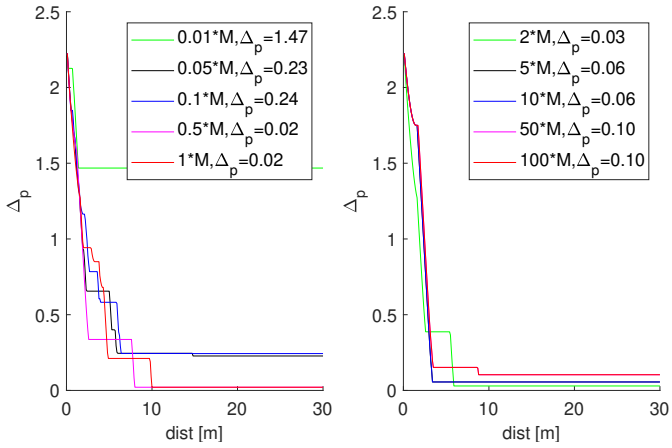

Figure 13: Robustness of the algorithm's performance to underestimation (left) and overestimation (right) of the Lipschitz constant, for the pyramid-based objective function.

$\delta = 0.1\,\text{m}$. When this happens, OOPA scores about half the travel distance on average to p^* (roughly 52.7%) compared to CDOO. Finally, the gradient-based method finds the global maximum in only 3 out of the 15 trials, but with faster convergence rates. Again, overall similar results are obtained to the RBF functions.

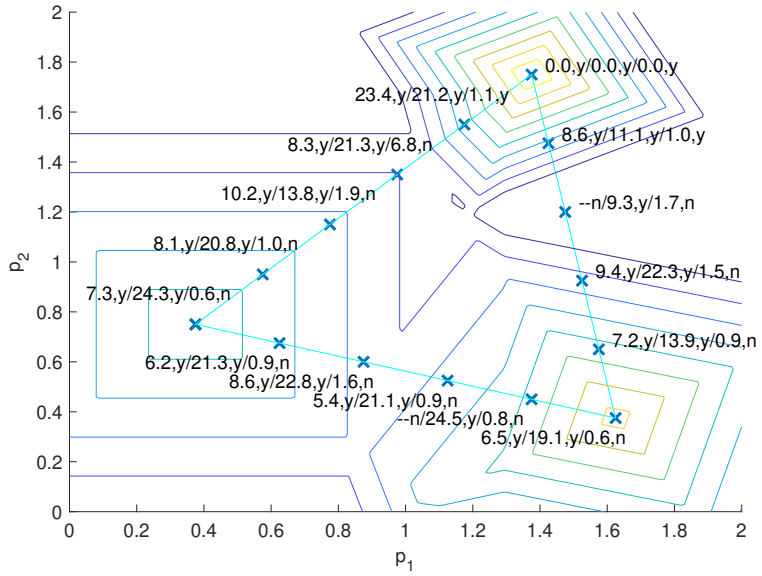

Figure 14: Baseline comparison of OOPA to the CDOO and Gradient Ascent methods in the case of pyramidal-based objective function.

5.2. Real-life experiment with a TurtleBot3

Finally, we provide an indoor experiment with the OOPA method in which a ROBOTIS TurtleBot3 searches for the grayscale optimum of a 2D printed map.

The robot moves according to the nonlinear, unicycle-like dynamics (15) with steps of 0.1 m taken along one of the headings $\{0, \pi/4, \ldots, 7\pi/4\}$. During a sampling period of $T_s = 12\,\mathrm{s}$, the agent samples f, runs a series of interpolative Q-iteration updates, applies the resulting control command up to the next evaluation point of OOPA, and finally communicates the completion of the sequence to a remote controller. The rather high sampling period is due to the low linear and angular velocities of the robot, chosen to reduce the accumulation of position errors. Note that each high-level control action generates a new reference pose that will be tracked by a low-level controller as explained below.

The printed map over which P is taken recreates the RBF-based objective function from Section 5 (see Figure 15 for a view of the experimental setup). Thus, $P = [0, 2] \times [0, 2]\,\mathrm{m}^2$ and the optimum is situated in $p^* = [1.375; 1.75]$. We are using for image acquisition a Raspberry Pi Camera Module V2 placed at $o = 20\,\mathrm{cm}$ in front of the robot and pointing down perpendicular to the ground at a height of 20 cm. Note that the offset of the camera is relative to the center of the robot, i.e. the middle point of the segment linking the two back wheels. A sample of f is determined as follows. Compute first the RGB average value of all the pixels situated in the region of interest (ROI) of a sampled image. The ROI is the center rectangle of the image after splitting it into 9 smaller rectangles with equal width and height. To search for the darkest point on the map, f will be the subtraction between 255 (as the pixels are represented with byte values) and the RGB average value computed before. We will informally call such a sample of f a grayscale value.

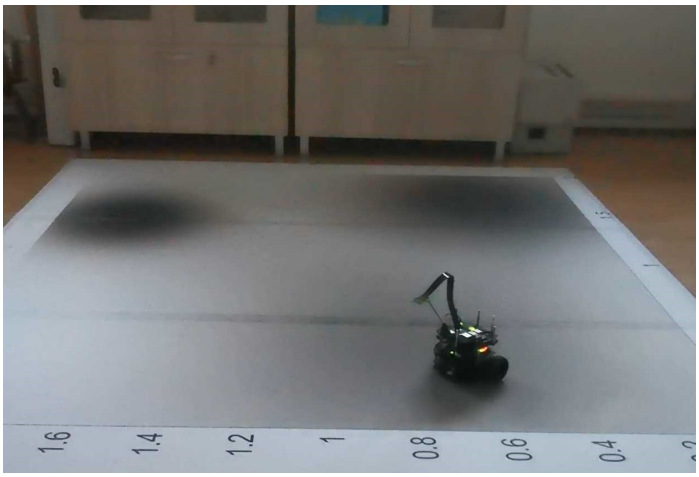

Figure 15: Overview of the TurtleBot3 experimental setup. Note the view is from the "left" of the surface in Figure 10.

Next, we present a simple method for dealing with the offset o, as positions of the

camera will be used in the algorithm and not of the actual robot. To avoid confusion we keep the previous notation p_k as the current camera location, while the pose of the robot will be denoted by $x_{r,k} = [p_{r,k}^T, \beta_{r,k}]^T$, where $p_{r,k}$ and $\beta_{r,k}$ are the position and orientation of the robot at step k. Given $x_{r,k}$, an action is sought such that the robot is led to a pose that brings the camera into p_{k+1} required by the algorithm at the next iteration. A possible solution is given by the equations:

$$\beta_{r,k+1} = \text{atan2}(p_{k+1,2} - p_{r,k,2},\, p_{k+1,1} - p_{r,k,1}) \tag{16}$$

$$d = ||p_{k+1} - p_{r,k}|| - o \tag{17}$$

$$p_{r,k+1} = p_{r,k} + d \cdot [\cos(\beta_{r,k+1}), \sin(\beta_{r,k+1})]^T \tag{18}$$

where atan2 is the two-argument arctangent. Then, $x_{r,k+1} = [p_{r,k+1}, \beta_{r,k+1}]^T$ represents the next, reference pose of the robot, which will be executed by the low-level controller, while p_{k+1} is used in OOPA as the evaluation point of f at step $k+1$.

The parameters of OOPA are taken similar to the simulations performed above. Interpolation is done across a state grid of 21×21 equidistant points. We do not include additional state signals y in x. The number of Q-iteration updates is set to $m = 3$ and the trajectory steps reduced to $n = 150$. A series of natural limitations are present in this experiment: camera (hardware) errors, the printed map not exactly having the gradients present in the simulations, somewhat poor indoor lighting conditions, etc. The latter issue leads to a tighter range of possible grayscale samples, as white is perceived by the camera as a light gray. The lower plots in Figure 17 show that during two of the experiments, the values of f roughly ranged in the interval $[100; 240]$ instead of the usual $[0; 255]$.

We provide next an overview of the implementations. The algorithm was developed using the Robot Operating System (ROS) framework in Python and the Matlab ROS Toolbox. Figure 16 presents the overall system architecture through a simplified version of an `rqt` graph. This graph shows the workflow of the application and the way in which its components (nodes, topics, etc.) interact with each other.

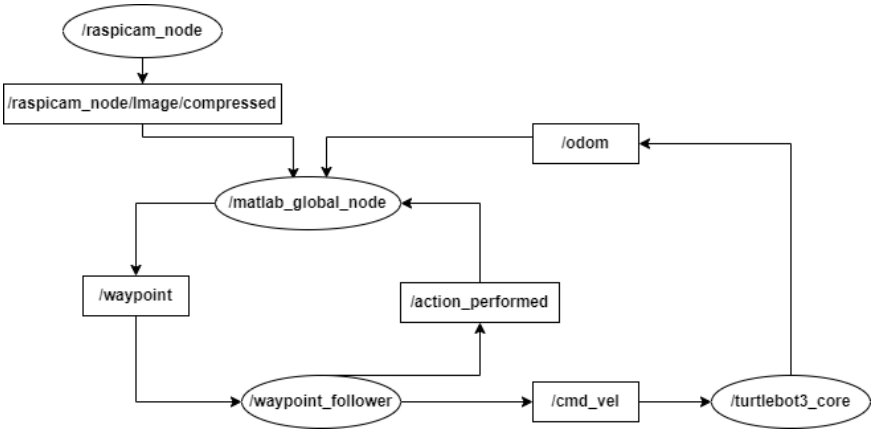

Figure 16: Simplified view of an rqt graph illustrating the control structure of the TurtleBot3 experiment.

The TurtleBot3 robot and a remote computer are simultaneously connected to the same ROS network. The remote computer runs the `matlab_global_node` which implements the high-level control through the OOPA algorithm. At each trajectory step k, a new image is received from the `raspicam_node` which publishes to the topic `Image/compressed`. Then, the image is processed and a grayscale value is plugged as the $f(p_k)$ input of the method. OOPA returns the new reference position for the camera and uses the odometry information given by the topic `odom` to compute the next control action. Each action is translated into a pose reference that is published by `matlab_global_node` on the `waypoint` topic as a `geometry_msgs/Pose` message. Note that we are publishing the next robot pose computed using (16)-(18), so that the camera will reach the next waypoint required by the algorithm.

The node `waypoint_follower` runs on the single-board computer of the TurtleBot3 robot. It is a subscriber of the `waypoint` topic and is responsible for the implementation of the low-level control of the robot. This control requires that the robot positions evolve as constant-speed, ramp signals. Each time a new message is published to `waypoint` a callback function is run to immediately track the reference pose. The translational and rotational speed of the robot are set at this point and further published to the `cmd_vel` topic to be executed by the core node provided by the manufacturer. Once the corresponding control sequence was successfully executed, an action completion flag is published on the `waypoint_performed` topic as a `std_msgs/String` message. Then, a callback function is run on the `matlab_global_node` side to resume the OOPA iterations. The cycle continues until the number of trajectory steps set at the beginning of the sweep is exhausted.

Due to the experimental limitations from above (smaller grayscale range), we expect the Lipschitz constant to be lower than $M = 730$ used in the simulations. However, this value represents a good starting point from which we will empirically tune M. The upper plots of Figure 17 report the trajectory results of two experiments in which M was taken 500 and 300. The plot in the top left is obtained for $M = 500$ and shows a trajectory that repeatedly encircles the middle point of the search space without focusing the refinements on the high f-valued regions. This is due to too much exploration. Thus, M needs to be decreased. $M = 300$ was chosen in the plot in the top right where a better exploration-exploitation tradeoff was obtained. Indeed, the robot spends more steps close to the higher RBF centers and approximates the optimum better as a result. It samples the highest f-value of $\overline{f} = 237.5$ as close as 2 cm to the optimum, approximating p^* in $\widehat{p^*} = [1.38, 1.73]$. In the previous sweep when M was set larger, p^* was found with an error of 14 cm in $\widehat{p^*} = [1.46, 1.64]$, where $\overline{f} = 221.7$. Note that the best function samples acquired so far for the two experiments are presented in the lower plots of Figure 17.

The paths followed by the robot on the physical map are not exactly the ideal ones presented in Figure 17. Due to the accumulation of errors in the control procedure, the robot drifts about $5-10$ cm from its expected position by the end of the OOPA run. As a result, the values of f that are sampled are not quite the expected ones. Such limitations are unavoidable in practice and could be partially solved by implementing a more precise positioning. For example a SLAM approach using a LiDAR could track more accurately the robot pose and thus lower the control errors.

The full video of the second real-life experiment highlighted in the right plots of Figure 17 is available online at `http://rocon.utcluj.ro/files/oopa_turtlebot.mp4`. Note that the video still in Figure 15 originates from the very same video, which is accelerated

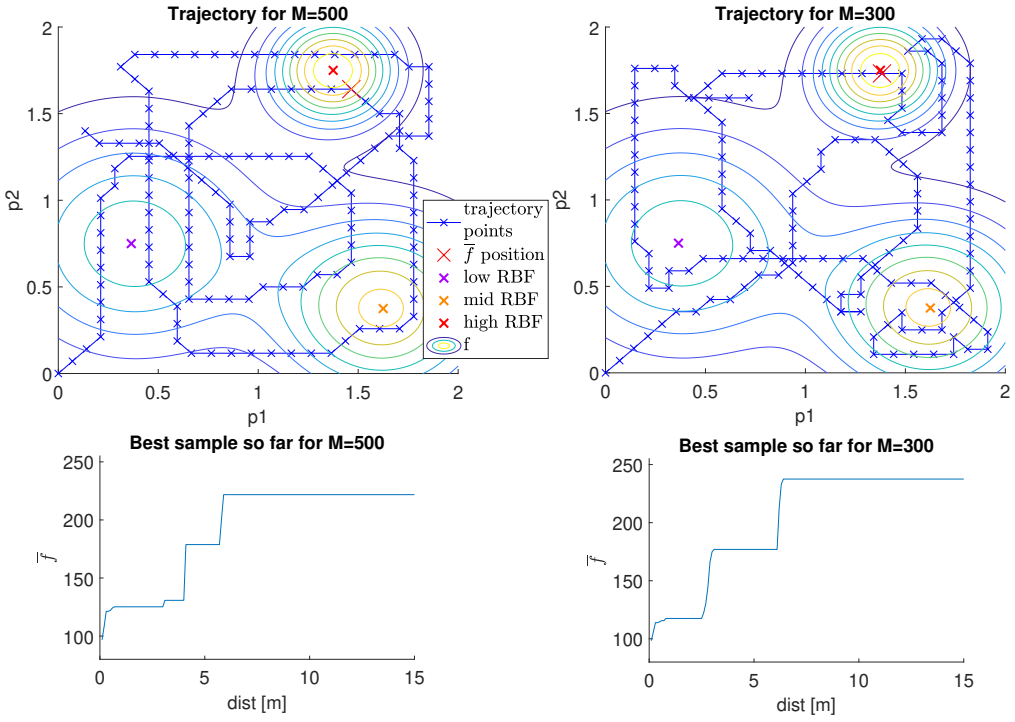

Figure 17: Left/Right plots: Trajectory and best sample acquired so far by the TurtleBot3 for $M = 500/M = 300$.

by a factor of 30 to fit in 1 min.

6. Conclusions

We presented an approach for optimization that can be used by a robot that travels in the physical space on which the function is defined. Extensive simulations showed that the algorithm works well when compared to baselines from local and global optimization that do not take into account the path taken by the robot. The algorithm was also illustrated for a mobile ground robot.

Future work includes extending the approach to multiple agents, and attempting to find analytical near-optimality guarantees for the algorithm. Furthermore, it will be important to illustrate the method in practical problems with a "real" objective function like litter density.

Acknowledgments

This work was been financially supported from H2020 SeaClear, a project that received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871295; by the Romanian National Authority for

Scientific Research, CNCS-UEFISCDI, SeaClear support project number PN-III-P3-3.6-H2020-2020-0060; and by CNCS-UEFISCDI, project number PN-III-P2-2.1-PTE-2019-0367.

References

Bertsekas, D., 07 2019. Reinforcement Learning and Optimal Control.

Binney, J., Krause, A., Sukhatme, G. S., 2010. Informative path planning for an autonomous underwater vehicle. In: 2010 IEEE International Conference on Robotics and Automation. pp. 4791–4796.

Binney, J., Sukhatme, G. S., 2012. Branch and bound for informative path planning. In: 2012 IEEE International Conference on Robotics and Automation. pp. 2147–2154.

Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R., 2010. Approximate dynamic programming with a fuzzy parameterization. Automatica 46 (5), 804–814.

Busoniu, L., Varma, V. S., Loheac, J., Codrean, A., Stefan, O., Morarescu, I.-C., Lasaulce, S., 2020. Learning control for transmission and navigation with a mobile robot under unknown communication rates. Control Engineering Practice 100, 104460.

Choset, H., 10 2001. Coverage for robotics - a survey of recent results. Annals of Mathematics and Artificial Intelligence 31, 113–126.

Durrant-Whyte, H., Bailey, T., 2006. Simultaneous localization and mapping: Part I. IEEE Robotics Automation Magazine 13 (2), 99–110.

Essa, K., Etman, S., El-Otaify, M., 01 2020. Relation between the actual and estimated maximum ground level concentration of air pollutant and its downwind locations. Open Journal of Air Pollution 09, 27–35.

Fink, J., Kumar, V., 2010. Online methods for radio signal mapping with mobile robots. In: 2010 IEEE International Conference on Robotics and Automation. IEEE, pp. 1940–1945.

Frazier, P. I., 2018. A tutorial on Bayesian optimization. Tech. Rep. arXiv 1807.02811.

Horst, R., Tuy, H., 01 1996. Global Optimization—Deterministic Approach. Springer Science & Business Media.

Lalish, E., Morgansen, K. A., 2008. Decentralized reactive collision avoidance for multivehicle systems. In: 2008 47th IEEE Conference on Decision and Control. IEEE, pp. 1218–1224.

Lattimore, T., Szepesvári, C., 07 2020. Bandit Algorithms. Cambridge University Press.

Lawler, E., Woods, D., 08 1966. Branch-and-bound methods: A survey. Operations Research 14.

Lilienthal, A., Duckett, T., 08 2004. Building gas concentration gridmaps with a mobile robot. Robotics and Autonomous Systems 48, 3–16.

Lim, Z., Hsu, D., Lee, W., 09 2015. Adaptive informative path planning in metric spaces. The International Journal of Robotics Research 35.

Liu, S., Chen, P.-Y., Kailkhura, B., Zhang, G., Hero III, A. O., Varshney, P. K., 2020. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. IEEE Signal Processing Magazine 37 (5), 43–54.

Munos, R., 01 2011. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In: Neural Information Processing Systems. Vol. 24.

Munos, R., 01 2014. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning.

Nocedal, J., Wright, S., 01 2006. Numerical Optimization. Springer.

Plaku, E., Kavraki, L. E., Vardi, M. Y., 2010. Motion planning with dynamics by a synergistic combination of layers of planning. IEEE Transactions on Robotics 26 (3), 469–482.

Sankey, T., Donager, J., McVay, J., Sankey, J., 06 2017. UAV lidar and hyperspectral fusion for forest monitoring in the southwestern USA. Remote Sensing of Environment 195, 30–43.

Santejudean, T., Busoniu, L., 9–13 December 2021. Path-aware optimistic optimization for a mobile robot. In: Proceedings 60th IEEE Conference on Decision and Control (CDC-21). Austin, Texas.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., de Freitas, N., 2016. Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE 104 (1), 148–175.