Path-aware optimistic optimization for a mobile robot

Tudor Sântejudean, Lucian Buşoniu

Abstract— We consider problems in which a mobile robot samples an unknown function defined over its operating space, so as to find a global optimum of this function. The path travelled by the robot matters, since it influences energy and time requirements. We consider a branch-and-bound algorithm called deterministic optimistic optimization, and extend it to the path-aware setting, obtaining *path-aware optimistic optimization* (OOPA). In this new algorithm, the robot decides how to move next via an optimal control problem that maximizes the long-term impact of the robot trajectory on lowering the upper bound, weighted by bound and function values to focus the search on the optima. An online version of value iteration is used to solve an approximate version of this optimal control problem. OOPA is evaluated in extensive experiments in two dimensions, where it does better than path-unaware and local-optimization baselines.

I. Introduction

We design and evaluate a method for a mobile robot to sample an unknown function defined over its operating area or volume, so as to find the global optimum of this function. The key difference from classical optimization is that the path taken by the robot is important, since it influences energy and time costs; and the function is unknown and must be learned from samples. We call this scenario "path-aware global optimization". It can be useful in many practical scenarios, where the optimum sought could be e.g. of some physical measurement such as pollutant concentration, temperature, humidity etc. [7], [13], the maximal density of surface or underwater ocean litter, the largest-bandwidth location for radio transmission [8], the largest sand height on the seabed for dredging, maximal or minimal forest density in inaccessible areas [18], and so on. As a first approximation, the length of the path serves as a proxy for the costs, although of course more accurate models are possible that take into account the dynamics of the robot, the terrain etc.

Local optimization methods like gradient descent, 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 [15]. We do evaluate such an approximate-gradient alternative; of course, the fundamental limitation is that these methods can only find a local optimum.

Global optimization techniques [10] on the other hand, like branch-and-bound, see [12], Chapter IV of [10] or Bayesian

optimization [9], [19] usually make arbitrarily large "jumps" in the space of solutions to sample a new point. Here, large steps are unsuitable because they would overcommit: the robot samples the unknown function as it travels, so during a long path, new information becomes available, and the old travel direction might become suboptimal. Figure 1 gives some intuition. 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 changes heading to the magenta trajectory.
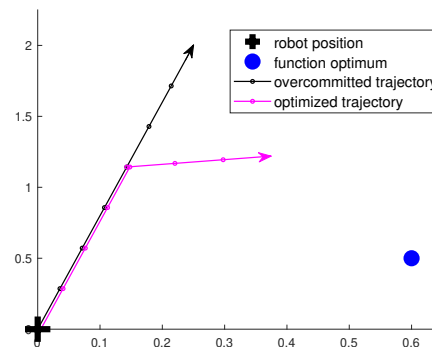

Fig. 1: Illustration of overcommitment.

In this paper, to solve the path-aware optimization problem, we extend an algorithm from the branch-and-bound class: deterministic optimistic optimization (DOO) [17], [16]. DOO organizes the search space as a tree containing at each depth a partition of the space, where the size of the subsets decreases with the depth. It makes a Lipschitz continuity assumption on the function and refines at each iteration a tree node with a maximal upper bound on the function value (i.e. a node that is likely to contain an optimum). We pick DOO because it guarantees a convergence rate towards the global optimum, which in essence says that only a "subspace" of a certain near-optimality dimension must be sampled to find the optimum [17].

The main contribution of the paper is to make DOO path-aware, by formulating the decision of which direction to go at each time step as an optimal *control* problem. Due to the danger of overcommitment, we do not immediately sample the largest-upper bound location, as DOO would, but we still exploit the underlying objective: to refine the upper bound around the optima. Specifically, the reward in the control problem is the volume by which each decision is expected to

refine the upper bound, i.e. the difference between the bound before, and after measuring the new sample. This refinement is weighted by the value of the bound and of the function itself at the current point, to focus the refinement around the optima. We run an online version of a value iteration algorithm [2] to solve the control problem of maximizing the cumulative rewards (weighted refinements) along the trajectory.

Solving the optimal control problem exactly at each decision step is impossible, both due to computational reasons and because finding the exact reward would require to know the future function samples. Therefore, we must resort to certain approximations, which we detail in the remainder of the paper. We validate the algorithm in extensive simulations, in which the key performance criterion is the path length taken to reach (close to) the optimum. We study the impact of the tuning parameters of the algorithm, its robustness to errors in the Lipschitz constant used to compute the upper bounds, and compare it to baselines adapted from classical DOO and gradient ascent.

Related work can also be found in other fields than optimization. For example, in artificial intelligence, bandit algorithms are a class of sample-based optimization of stochastic functions [11]. 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 [6] 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 [3], [4], [14]. Other variants, like coverage [5], 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.

Next, Section II gives necessary background on DOO. Section III states the problem and formulates it as optimal control, while Section IV provides the new algorithm to solve it. Section V presents the numerical results, and Section VI concludes the paper.

II. BACKGROUND

DOO is an algorithm belonging to the branch-and-bound class that aims to estimate the optimum of a function $f : X \to \mathbb{R}$ from a finite number of function evaluations. It sequentially splits the search space X into smaller partitions and samples to expand further only those partitions associated with the highest upper bound values. After the numerical budget has been exhausted, the algorithm approximates the maximum as the state with the highest f value evaluated so far. An assumption made by DOO is that there exists a (semi) metric over X, denoted by l, and f is Lipschitz continuous w.r.t. this metric at least around its optima, in the sense:

$$f(x^*) - f(x) \leq l(x^*, x), \forall x \in X \qquad (1)$$

where $x^* \in \operatorname{argmax}_{x \in X} f(x)$. Note that for convenience, we will require here the inequality property to hold for any pair $(x, y) \in X^2$:

$$|f(x) - f(y)| \leq l(x, y), \forall x, y \in X \qquad (2)$$

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

$$l(x, y) = M||x - y||, \forall x, y \in X \qquad (3)$$

where M represents the Lipschitz constant. The method can be extended to any metric l obeying the assumptions present in [16].

Here we will use an alternative approach to the partition splitting in DOO: the construction of a so-called "saw-tooth" upper bound [16], defined as $B : X \to \mathbb{R}$ so that:

$$f(x) \leq B(x) = \min_{(x_s, f(x_s)) \in S} [f(x_s) + l(x, x_s)], \ \forall x \in X \quad (4)$$

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

$$x_+ \in \operatorname{argmax}_{x \in X} B(x). \qquad (5)$$

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).

III. PROBLEM STATEMENT

Given an unknown function $f : X \to \mathbb{R}$, a global optimum must be found in the least number of steps. Consider the maxima locations:

$$x^* \in \operatorname{argmax}_{x \in X} f(x) \qquad (6)$$

where x represents a physical location in the space $X \subset \mathbb{R}^p$. No previous knowledge of the function is available to the robot and thus the function must be learnt from the samples taken across a single trajectory. The path travelled is important due to energy and time considerations often encountered in practical scenarios. Another constraint is that, due to limited velocity, the robot cannot sample at arbitrarily distant positions across the state space, being limited to neighboring ones.

The motion dynamics are described by the p-dimensional positions $x \in X$ and system inputs $u \in U \subset \mathbb{R}^p$. Most of the times $p \in \{2, 3\}$. For simplicity, we consider simple integrator dynamics with the possibility of extension to more complex dynamics. Thus, the discrete-time dynamics are given by $g : X \times U \to X$:

$$g(x_k, u_k) = x_k + u_k = x_{k+1} \qquad (7)$$

where k indexes the step of the considered trajectory.

The solution we propose is inspired by DOO as it builds and refines with each sample gathered the saw-tooth upper bound of the function. Unlike DOO, it cannot sample arbitrarily far in the search space (recall the dynamics constraints) and thus the classical approach of always sampling

the point with the highest B-value is inappropriate. By following it, the robot would not be using the samples gathered until the target is reached, possibly overcommiting to a trajectory that meanwhile became suboptimal. To address this issue, we make the algorithm aware of its path by defining an optimal control problem (OCP) with a different goal.

We start with a high-level intuition of this OCP, followed by the formal definitions. We do not sample directly the highest B-value points, but rather lower the upper bound around the optima to implicitly find x^*. 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. The classical learning dilemma between the exploration and exploitation tradeoff arises here too: encouraging a) will lead to excessive refinements in untraveled regions, less focused around high-value function points (too high exploration), while 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 high 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 p; for $p = 1$ it translates to area). With each new sample the function 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 and instead we must rely on approximations to predict it; refer to 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 state x, weighted by the average of the function value and its upper bound, both evaluated at x:

$$\rho(x, u) = \frac{\widehat{f}(x) + B(x)}{2} r(x, u) \quad (8)$$

where $\rho(x, u)$ is the reward function, B is the upper bound function defined in (4) and $r(x, u)$ represents the volume predicted to be refined by taking action u in state x. The refinement is computed in the following way. First, denote with $S = \{(x, f(x)) | x \in X\}$ the set of samples acquired so far. Compute next the upper bounds B_1 and B_2 using (4) and two slightly different sets of samples: B_1 with $S \cup \{(x, \widehat{f}(x))\}$ and B_2 with $S \cup \{(x, \widehat{f}(x)), (x_+, \widehat{f}(x_+))\}$, where $x_+ = g(x, u)$. The volume refined is determined through trapezoidal numerical integration over the difference $B_1 - B_2$ across the p dimensions of X.

The terms $B(x)$ and $r(x, u)$ direct the refinements to locations with high upper bounds where optimistically a maximum might be situated (via $B(x)$) and where the robot has the potential to significantly lower B (via $r(x, u)$).

Factor $\widehat{f}(x)$ in (8) tells the robot to visit states closer to those having high function values. Due to a limited number of samples acquired along the single-run trajectory, $\widehat{f}(x)$ is in most cases a prediction (especially at the beginning of the run). We compute this prediction by taking the function value of the closest point to x that was already sampled. We do this for two reasons: taking a lower quantity than $\widehat{f}(x)$ would contradict the optimistic approach of our algorithm by overly encouraging exploitation, and taking a higher value, say closer to $B(x)$, would translate into an overly optimistic approach that encourages too much the exploration. If state x was sampled before, its corresponding function value is directly taken.

Figure 2 gives an example of the reward calculation for a simple 1D case, where the evaluation point is denoted by x_k and the next point (corresponding to u_k in $\rho(x_k, u_k)$) is x_{k+1}. As x_k was already sampled, its function value $f(x_k)$ is used to approximate $\widehat{f}(x_k)$, $\widehat{f}(x_{k+1})$ (as x_k is the closest sampled point to x_{k+1}) and $B(x_k)$. The area predicted to be refined is colored in green and is of course an approximation, because one cannot guess in advance the next function samples, but only rely on approximations to predict them.
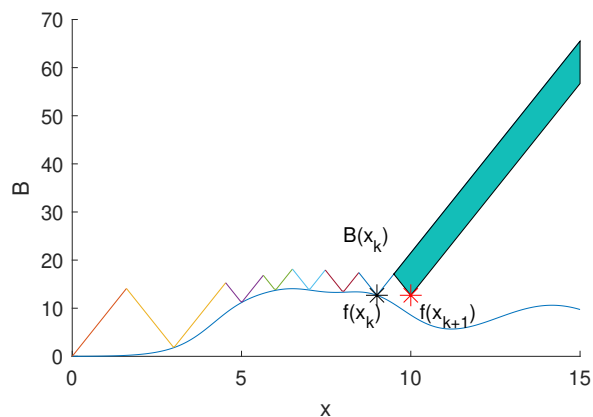
Fig. 2: The sampled function is drawn with a blue line and the saw-tooth envelope represents its upper bound. The function sample $f(x_k)$ (black star) is used to approximate $\widehat{f}(x_k)$, $\widehat{f}(x_{k+1})$ (red star) and determine $B(x_k)$. The area predicted to be refined is colored in green.

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

Due to the online learning character, the OCP is a time-varying problem in which $\rho(x, u)$ changes with each new sample gathered as more information is available. More specifically, the volume refinements r, function shape estimate $\hat{f}$ and the upper bound function B change as the agent describes its path. This leads to a reward definition in (8) that is time-varying and we cannot derive the convergence guarantees present in classical value iteration. However, we expect the changes 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.

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 maxima), we define V in the infinite horizon setting:

$$V^h(x) = \sum_{k=0}^{\infty} \rho(x, u) \quad (9)$$

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.[1] We are searching for an optimal policy, denoted by h^*, such that:

$$V^{h^*}(x) \geq V^h(x), \forall x, h. \quad (10)$$

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

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

Once this equation has been solved to find Q^*, the optimal policy is given by $h^*(x) = \arg\max_u Q^*(x, u)$.

IV. ALGORITHM

Algorithm 1 applies value iteration (VI) in an online scheme to solve the OCP defined above. To build the upper bound and value function estimations, the robot needs to gather informative samples during its exploration procedure. Recall that samples can only be gathered in the current positions of the robot and are further used to approximate the function in unsampled points when computing the rewards in (8).

At each step k, the robot takes a sample $f(x_k)$ and adds it to the sample set S. Then, we update upon the current value function estimation by running m value iteration (VI) updates of the following form:

$$Q_+(x, u) = \rho(x, u) + \max_{u'} Q(g(x, u), u') \quad (12)$$

based on the Bellman equation (11). For simplicity, the states are discretized into a grid X_{grid}, having n_{grid} equidistant points along each of the X dimensions. We pick states on X_{grid} and actions so that the next state given the dynamics always falls on the grid. Actions leading the robot to the states up, down, left or right with one grid position with respect to the current position are considered. Thus, the VI updates (12) must be run only on states from X_{grid} and on the discretized actions. Many representation schemes, including some for continuous actions, can be used to get rid of this limitation, and they will be studied in future work.

To fully solve the OCP, the true rewards should exploit how the B-values change (lower) with the new samples, meaning that the refinement gains at later steps would be smaller. Not knowing in advance the evolution of B

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

means that ρ will overestimate these gains at future steps. This represents a key reason why m needs to be limited. Moreover, the OCP changes slightly as it learns online and fully converging to the solution of step-k would likely not be useful, since at step-$k+1$ the OCP will be updated again.

The robot chooses next the action that maximizes the Q-values:

$$u_k = \arg\max_u Q(x_k, u), \quad (13)$$

applies it and measures the new state. The procedure continues until the number of steps in the trajectory, denoted by n, is exhausted.

Algorithm 1 describes the steps presented above.

Algorithm 1 Path-Aware Optimistic Optimization (OOPA).

Input: g, n_{grid} number of steps per X and B grid axes, discretized actions U, m number of VI sweeps, n trajectory steps, M Lipschitz constant
1: generate X and B grid, X_{grid}, using n_{grid}
2: initialize samples set $S \leftarrow \emptyset$
3: initialize $Q_0(x, u) \leftarrow 0, \forall x \in X, u \in U$
4: measure initial state x_0
5: **for** each step $k = 0, \ldots, n - 1$ **do**
6:     sample $f(x_k)$, add pair $(x_k, f(x_k))$ to S
7:     **for** each VI sweep $m' = 0, \ldots, m - 1$ **do**
8:         **for** all states $x \in X_{grid}$, actions $u \in U$ **do**
9:             $x_+ \leftarrow g(x, u)$
10:             find $\hat{f}(x)$, $\hat{f}(x_+)$ and $B(x)$ using S and (4)
11:             $S_1 \leftarrow [S, (x, \hat{f}(x))]$, $S_2 \leftarrow [S_1, (x_+, \hat{f}(x_+))]$
12:             compute $B_1(x), \forall x \in X_{grid}$ using S_1 and (4)
13:             compute $B_2(x), \forall x \in X_{grid}$ using S_2 and (4)
14:             compute $r(x, u)$ using trapezoidal integration
15:                 over $(B_1 - B_2)$ across X_{grid}
16:             compute $\rho(x, u)$ using (8)
17:             $Q_{m'+1}(x, u) = \rho(x, u) + \max_{u'} Q_{m'}(x_+, u')$
18:         **end for**
19:     **end for**
20:     $Q_0 = Q_m$
21:     $u_k = \arg\max_{u \in U} Q_m(x_k, u)$
22:     apply action u_k, measure next state x_{k+1}
23: **end for**

We call the algorithm Path-Aware Optimistic Optimization (OOPA). It has the following parameters that need to be tuned: m representing the number of VI updates, the Lipschitz constant M (used to compute B in lines $12-13$ via (4) and (3)), and the discretization factor n_{grid} that dictates the number of points taken across each dimension of X. The first parameter, m, impacts the propagation of the rewards across the state grid considered. Taking a too high m will not only lead to high computation times that are not viable in practice, but also overly extrapolate the rewards that are mostly overestimations of their true quantities. Therefore, we recommend taking $m \leq 5$. The Lipschitz constant is generally unknown and needs to be tuned empirically. Taking M much lower than its true value will create an upper bound that no longer satisfies the inequality $f(x) < B(x)$,

and thus the algorithm could break. A good approach is to take a rather high M and lower it sequentially based on the feedback provided by the experiments. Finally, the grid should a priori be taken as large as feasible given the computational resources; we investigate the effect of the grid size in the next section.

V. EXPERIMENTS AND DISCUSSION

At first, we define a standard setup in which we aim to study the new method and compare the OOPA algorithm against two baselines: Classical DOO (CDOO) that uses the saw-tooth approach to build the upper bound and commits to sample always the highest B-value point; and Gradient Ascent that creates an approximation plane using Local Linear Regression [1] on the neighboring samples and follows the plane's gradient to quickly converge to a local maximum.

A 21x21 interpolation grid is taken across a state space of length $4x4$ m. The function to optimize is composed of a sum of three radial-basis (RBF) functions with different coefficients: width $b_i \in \{[1.3; 1.3], [0.6; 0.6], [1; 1]]\}$, height $h_i \in \{[148.75, 255.0, 212.5]\}$ and centers $c_i \in \{[0.75; 1.5], [2.75; 3.5], [3.25; 0.75]\}$ (see Figure 9 for a contour plot). So the global optimum is $f^* = 255$ situated in $[2.75; 3.5]$. The corresponding Lipschitz constant is approximated starting from the Mean Value Theorem and then tuned experimentally to ensure that it produces close to true upper bounds. Thus, the Lipschitz constant was set to $M = 364.54$.

A. Influence of tuning parameters. Refinement prediction accuracy

The first parameter to be tuned is m, the number of VI updates at each step taken by the robot. On the setup defined above, we run the algorithm for $m \in \{1, 2, 3, 4, 5\}$ and study the maximum f-value sampled: $\overline{f} = \max_{x_s}(f(x_s))$, and the minimum distance until x^*: $\Delta_x = \min_{x_s}(||x^* - x_s||)$, where x_s represents a sample point along the trajectory performed so far. Another metric of interest is the minimum difference between the optimum and the values of f sampled so far: $\Delta_f = \min_{x_s}(f^* - f(x_s))$, with x_s having the same meaning as above. The number of trajectory steps for each experiment is set to $n = 125$, equivalent to 25 m travel distance. The robot starting position is set in the grid center, $x_0 = [2; 2]$.

Figure 3 shows that odd values of m perform better in finding the maximum position with higher accuracy, while even values of m are suboptimal, converging to the second highest RBF. The best choice seems to be $m = 3$, which gets close to the maximum faster and scores 20% and 30% less distance to x^* compared to the cases of $m = 1$ and $m = 5$ (8.4 m compared to 10.6 m and 12.4 m), respectively. The intuition is that rewards based on the volume refinements need to be propagated across the state space, however not too much, since they are mostly predictions (approximations) and in most cases their values are overestimated, especially at the beginning of the experiment. Choosing $m = 3$ gives balanced results in terms of both exploration and exploitation.

Figure 4 (left) shows the refined upper bound of the sampled function built using (4) and the $n = 125$ samples
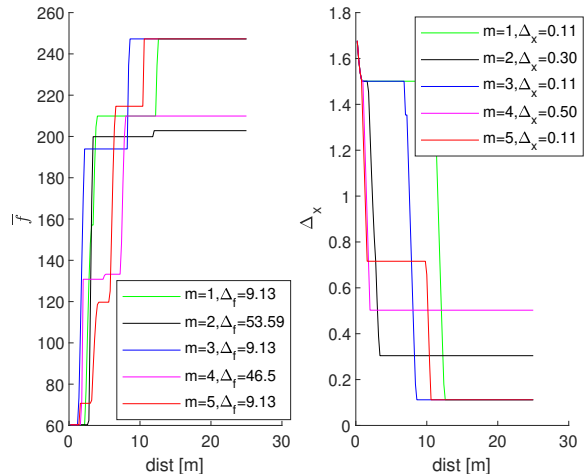

Fig. 3: Illustration of the maximum value sampled so far (left) and the minimum distance to the maximum center x^* (right), denoted with Δ_x; for $m \in \{1, 2, 3, 4, 5\}$. The legend on the left shows the minimum distance between the samples of f and its maximum value f^*, denoted by Δ_f.
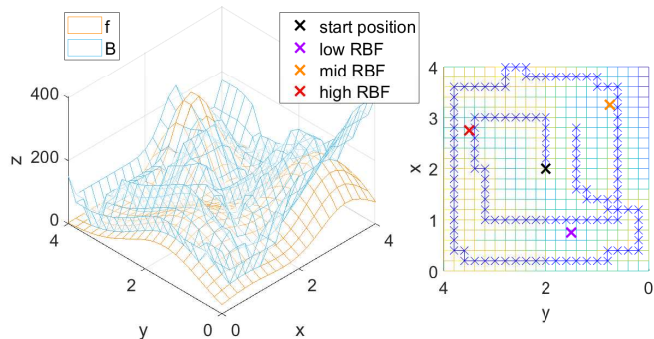

Fig. 4: Left: The sampled function (in orange) is bounded from above by the refined B function (in blue), both evaluated on the same grid. Right: The sampling trajectory of the robot, drawn with blue 'x' and starting from the black 'x'. The refinements are geared – without being overly committed – towards higher function values (RBF peaks). This highlights a good exploration-exploitation tradeoff obtained for $m = 3$.

acquired. The robot trajectory in Figure 4 (right) shows more steps being spent closer to the highest RBF (centered in $[2.75; 3.5]$) and less around the ones with lower values (centered in $[0.75; 1.5]$ and $[3.25; 0.75]$). This is expected, since the rewards take into account not only the upper bound, but also the function values.

Next, we will study the impact of the grid size on the behavior of the algorithm. For this, the state space will be split into grids of size $\{21^2, 26^2, 31^2, 36^2, 41^2\}$ and the behavior evaluated using the metrics from above. We keep the algorithm tuning and setup unchanged, with the exception of grid discretization and trajectory length set now to 75m. This length is equal to the product between the grid step size and the samples budget n for each experiment respectively.

Except a "lucky" case (21^2), finer grids find the optimum with higher accuracy. Note that on all grids x^* is found with
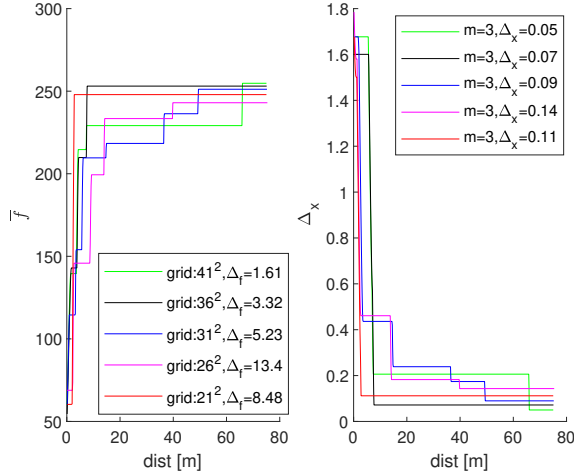
Fig. 5: Smaller grid step sizes generally lead to better precision when searching for the optimum. Δ_f is displayed in the legend on the left, while the legend on the right displays Δ_x.

a precision of 1 grid step size, as $m = 3$ was kept constant during this experiment.
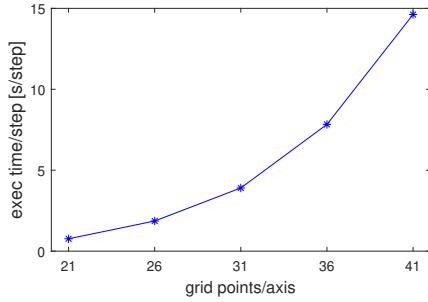

Fig. 6: Time complexity has a quadratic growth with respect to the grid size.

Time complexity has a quadratic growth with respect to the grid of points. Figure 6 shows the average execution time per step for each grid studied above.

Since in practice the Lipschitz constant is generally unknown, it is instructive to check the robustness of the algorithm in the event of overestimation or underestimation of M. Another goal is to provide a way of choosing M, as in practical cases computing the Lipschitz constant analytically is often impossible, and M must instead be empirically tuned. We run the algorithm on the setup initially defined taking $M' = \lambda \cdot M$, with $\lambda \in \{0.2; 0.4; 0.6; 0.8; 1; 1.25; 1.5; 2; 2.5; 3\}$.

Figure 7 shows that taking M lower that a half of its initial approximation can lead to finding the optimum late or even break the algorithm. On the other hand, overestimating the Lipschitz constant is a safer choice when its value is not (precisely) known. In this experiment, higher values of M find x^* at worst later and do not break the algorithm. A possible reason is that even though the volume refinements are weighted by the mean of the sampled function and its upper bound value, the latter has a greater impact due to its generally higher value. Decreasing M too much will create
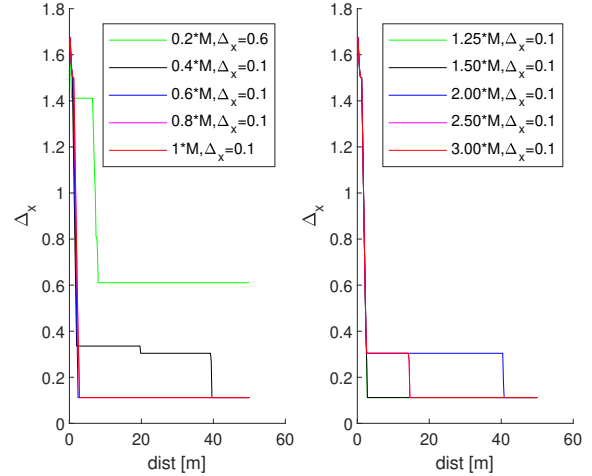

Fig. 7: Robustness to underestimation (left) and to overestimation (right) of the Lipschitz constant. Taking M less than half of its initial approximation leads to finding x^* late or even breaks the algorithm. Taking M more than twice of this approximation finds at worst later x^*.

an upper bound B with much lower values compared to the true ones. So, as a rule of thumb, one should generally choose M high and decrease its value based on experiments.
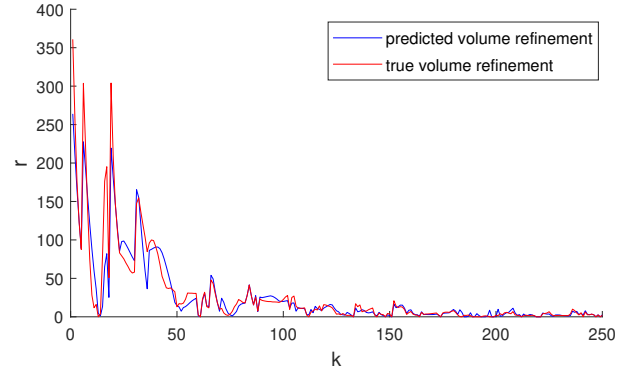

Fig. 8: Illustration of accuracy of the upper bound refinements.

Finally, since reward values (8) are based on predictions of upper bound refinements, we evaluate the accuracy of these predictions. Figure 8 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 $m = 3$ and 250 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 $\widehat{f}(x_+)$ gets closer to the true $f(x_+)$. Thus, using the predictions is justified.

B. Comparison to baselines

The learning-based algorithm is next compared to two different baselines. The first one is represented by a classical DOO (CDOO) algorithm that, similarly to OOPA, applies the saw-tooth approach (4) to refine the function envelope

with each sample taken. 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 aware of the information gathered through the samples performed along the trajectory to the next target point.

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's direction that results is followed by the robot with maximum velocity. This method has the natural limitation of converging only to local maximum points, however at faster rates.

To have a fair comparison, 15 equidistant starting positions placed along the lines drawn by the 3 RBF centers (recall the initial setup) are considered. To give enough time to the algorithms to find the maximum, we set the number of steps per trajectory to $n = 250$ (50 m travel distance). The robot will move with a fixed step of 0.2 m (equal to the step size of the 21^2 grid). 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. For the gradient-based method $N = 4$ ($N \geq 3$ is required to build the approximation plane).

We display in Figure 9 the result of the experiments. Near each starting position, drawn with 'x', the distance until the optima for each method is given. 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 convergence (when convergence does not occur, the distance is replaced by a '-'). After the comma, letters 'y' (for yes) or 'n' (no) show whether the maximum was found with accuracy of 1 grid step size, $\delta = 0.2$ m. 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).
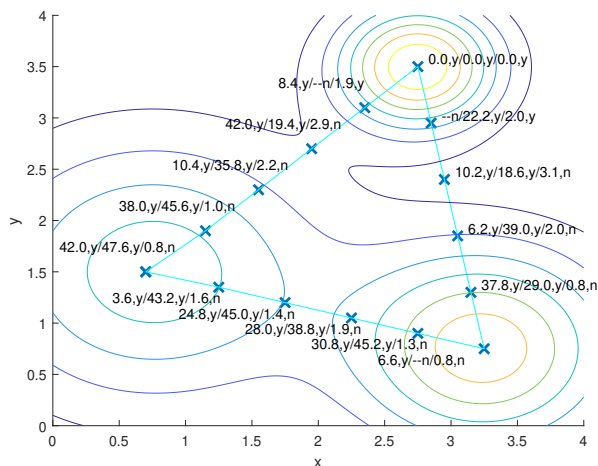

Fig. 9: Results of the OOPA, CDOO and Gradient Ascent methods.

Excepting a few outliers, the DOO-based methods find the maximum position with accuracy $\delta = 0.2$ m (1 grid step size). When x^* is found, OOPA scores 37.55% less distance on average compared to CDOO. Gradient Ascent finds the maximum in only a fifth of the runs, mainly when it starts close to the center of the highest RBF. When this happens, the gradient-based method tends to find x^* faster compared to OOPA or CDOO. In other initial states, Gradient Ascent converges to a local optimum. These behaviors are expected due to the local nature of the gradient and its straightforward choice of the heading direction.

VI. CONCLUSIONS

We considered the problem of finding a global optimum of a function defined over some physical space, by sampling it with a mobile robot. A method based on value iteration was defined to quickly reduce the upper bounds around optima, and thus implicitly find an optimum.

In future work we will consider more general robot dynamics than simple integrators, and demonstrate the method on real robots. Another objective is to provide near-optimality guarantees.

REFERENCES

[1] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1–5, pp. 11–73, 1997.
[2] D. Bertsekas, *Reinforcement Learning and Optimal Control*, 07 2019.
[3] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 4791–4796.
[4] J. Binney and G. S. Sukhatme, "Branch and bound for informative path planning," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2147–2154.
[5] H. Choset, "Coverage for robotics - a survey of recent results," *Ann. Math. Artif. Intell.*, vol. 31, pp. 113–126, 10 2001.
[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.
[7] K. Essa, S. Etman, and M. El-Otaify, "Relation between the actual and estimated maximum ground level concentration of air pollutant and its downwind locations," *Open Journal of Air Pollution*, vol. 09, pp. 27–35, 01 2020.
[8] J. Fink and V. Kumar, "Online methods for radio signal mapping with mobile robots," 05 2010, pp. 1940–1945.
[9] P. Frazier, "A tutorial on bayesian optimization," 07 2018.
[10] R. Horst and H. Tuy, *Global Optimization—Deterministic Approach*, 01 1996.
[11] T. Lattimore and C. Szepesvári, *Bandit Algorithms*, 07 2020.
[12] E. Lawler and D. Woods, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, 08 1966.
[13] A. Lilienthal and T. Duckett, "Building gas concentration gridmaps with a mobile robot," *Robotics and Autonomous Systems*, vol. 48, pp. 3–16, 08 2004.
[14] Z. Lim, D. Hsu, and W. Lee, "Adaptive informative path planning in metric spaces," *The International Journal of Robotics Research*, vol. 35, 09 2015.
[15] S. Liu, P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney, "A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 43–54, 2020.
[16] R. Munos, *From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning*, 01 2014.
[17] ——, "Optimistic optimization of a deterministic function without the knowledge of its smoothness," vol. 24, 01 2011.
[18] T. Sankey, J. Donager, J. McVay, and J. Sankey, "Uav lidar and hyperspectral fusion for forest monitoring in the southwestern usa," *Remote Sensing of Environment*, vol. 195, pp. 30–43, 06 2017.
[19] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.