A simple path-aware optimization method for mobile robots [⋆]

Tudor Sântejudean [*] **Lucian Buşoniu** [*] **Vineeth Varma** [**] **Constantin Morărescu** [**]

[*] *Technical University of Cluj-Napoca, Romania (e-mail: tudorsantejudean@gmail.com, lucian@busoniu.net)*
[**] *Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France. {vineeth.varma, constantin.morarescu}@univ-lorraine.fr.*

Abstract: We present an approach for a mobile robot to seek the global maximum of an initially unknown function defined over its operating space. The method exploits a Lipschitz assumption to define an upper bound on the function from previously seen samples, and optimistically moves towards the largest upper-bound point. This point is iteratively changed whenever new samples make it clear that it is suboptimal. In simulations, the method finds the global maxima with much less computation than an existing, much more involved technique, while keeping performance acceptable. Real-robot experiments confirm the effectiveness of the approach.

Keywords: Optimization, learning control, mobile robots.

1. INTRODUCTION

Consider a scenario in which a mobile robot must find the optimum of an initially unknown function defined over its physical operating area or volume. 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 with a robot 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), or the strongest-signal location for radio transmission in networked robots (Fink and Kumar, 2010; Busoniu et al., 2020), among others.

Local optimization methods are inappropriate since they do not find the global optimum. On the other hand, global optimization (Horst and Tuy, 1996) like branch-and-bound techniques (Lawler and Woods, 1966) or Bayesian optimization (Frazier, 2018) may choose the next sample at a location arbitrarily far away from the current robot position. That is because they do not take into account that the robot must physically move to that location, which is costly. In path-aware optimization, the robot must be able to revise its trajectory if from the newly accumulated samples it becomes clear that the optimum is located in another direction.

In (Santejudean and Busoniu, 2021), we introduced a path-aware, optimistic optimization method (OOPA) that is a variant of the branch-and-bound, deterministic optimistic optimization (DOO) algorithm (Munos, 2011). The bound exploits a Lipschitz assumption on the function. Like other global methods, DOO samples points irrespective of their position, in this case choosing at each iteration the point where an upper bound on the function is maximal. In OOPA, a "sawtooth" upper bound is built from position-function value pairs encountered along the trajectory. Instead of going straight to the maximal-bound point like DOO, OOPA solves an optimal control problem that aims both to visit locations with large function values or upper bounds, and also to refine (lower) these upper bounds. An intricate procedure is used to estimate these refinements and solve the optimal control problem approximately.

While OOPA works well in practice, it is computationally intensive and depends on tuning several parameters empirically. The algorithm involves several types of approximation, which make it difficult to analyze. Our objective in this paper is to provide a much simpler and easier to tune algorithm, without sacrificing too much performance.

To this end, we use the same upper bound as in OOPA, but instead of solving a complicated optimal control problem, we adopt a much simpler rule. The algorithm moves towards the maximal-upper bound point as long as the *bound* value there is larger than all the sampled *function* values. When this condition is no longer true, the algorithm turns towards the new maximal-bound point. We call this method "Turn When Function value is larger", abbreviated FTW.

In simulated experiments, we show that FTW is competitive with OOPA in terms of how quickly it reaches close to the optimum, while being significantly cheaper computationally. We investigate the dependence of the performance on the smoothness of the underyling function. We finally provide a real-robot application of our method where a TurtleBot3 mobile robot seeks the darkest location on a 2D surface.

[⋆] 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; and by the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, SeaClear support project number PN-III-P3-3.6-H2020-2020-0060.

Besides optimization, our technique is related to path planning in robotics (LaValle, 2006; Alexopoulos and Griffin, 1992; Aggarwal and Kumar, 2020). Classically, path planning aims to find a given goal position; for us, the goal position is unknown, and we exploit the structure of the problem to provide a custom path planning approach.

The paper has the following structure: first, in Section 2 the problem is defined, and DOO and OOPA are presented. Then, the new algorithm FTW is explained in Section 3. Section 4 presents our simulation results and Secton 5 the real experiment. In the end, in Section 6 some conclusions and future work are given.

2. PROBLEM STATEMENT AND BACKGROUND

Consider a problem in which a mobile robot needs to approximate the global optimum of a function $f : X \to \mathbb{R}$ in minimum time using sequential evaluations of the objective f. The robot moves with the dynamics described by the control inputs $u \in U$ over the search space X, defined in discrete time by $g : X \times U \to X$:

$$g(x_k, u_k) := x_{k+1} \tag{1}$$

with k indexing the step of the trajectory. The agent has no previous knowledge of the function and thus f must be learnt while searching for the optimum $x^* := \text{argmax}_{x \in X} f(x)$. Due to dynamics constraints (e.g. limited velocity), it is not possible for the robot to sample at next steps positions situated arbitrarily far away from each other, being limited to the neighboring ones. This problem formulation is formally known as *path-aware global optimization* (Santejudean and Busoniu, 2021).

DOO is a global optimization algorithm belonging to the branch-and-bound class that aims to estimate the optimum of f from successive function evaluations. It sequentially splits the search space X into finer partitions and samples to expand further only those 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 x 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, w.r.t. which f is Lipschitz continuous at least around its optima, in the sense:

$$f(x^*) - f(x) \le l(x^*, x), \forall x \in X \tag{2}$$

where $x^* \in \text{argmax}_{x \in X} f(x)$. For convenience, we will require here the property to hold for any pair $(x_1, x_2) \in X^2$:

$$|f(x_1) - f(x_2)| \le l(x_1, x_2), \tag{3}$$

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

$$l(x_1, x_2) = M||x_1 - x_2||_\infty, \tag{4}$$

Our method can be extended to any metric l. Note that a regularity assumption such as Lipschitz continuity is likely required to obtain a global optimization algorithm that does not require sampling the entire space.

Exploiting the Lipschitz property described in equations (2)-(4), we use here an alternative 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 $B : X \to \mathbb{R}$ so that:

$$f(x) \le B(x) := \min_{(x_s, f(x_s)) \in S} [f(x_s) + l(x, x_s)], \ \forall x \in X \tag{5}$$

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. See Figure 2 for an example. At each iteration, the next state to sample is given by the formula:

$$x_+ := \text{argmax}_{x \in X} B(x). \tag{6}$$

The algorithm iteratively samples points selected with equation (6). Note that B is lowered (refined) with each new sample gathered by the robot, implicitly via (5).

The approach in which the robot picks the point according to (6), commits to this point, and only changes the trajectory once it has been reached, will be called Classical DOO (CDOO).

However, if one accounts for the dynamics constraints previously discussed, this classical strategy becomes inappropriate. By committing to paths leading to the maxima of B, the robot would travel trajectories that in the meantime became suboptimal, as new information (in the form of samples) becomes available. Figure 1 provides some intuition on this phenomenon, which we call overcommitment.

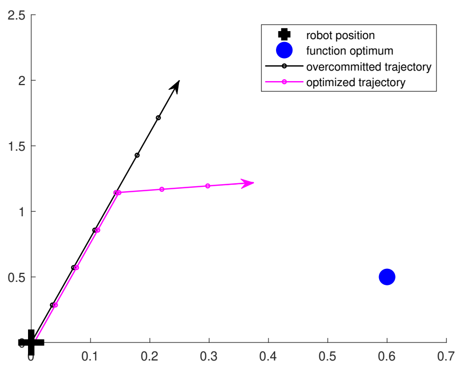

Fig. 1. Based on the information available to the robot when it is at the black cross, a classical algorithm decides to check the point at the black arrow. However, samples (dots) accumulated along the black trajectory give more information about the optimum (blue disk), so it becomes apparent that continuing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

To address these issues, Path-Aware Optimistic Optimization (OOPA) (Santejudean and Busoniu, 2021) also uses the upper bound of (5) while searching for the optimum, but it solves at each step an optimal control problem (OCP) to obtain the next action to be taken by the robot. The OCP objective is to maximize the cumulative sum of rewards (negative costs), where each reward is defined as a refinement of the upper bound weighted by the mean between the bound and function values:

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

Here, $\rho(x, u)$ is the reward function, $\widehat{f}$ an estimate obtained using a sample-based approximator (e.g. nearest-neighbor), B is the upper bound, and $r(x, u)$ is the volume predicted to be refined by applying action u in state x. The idea is that we prefer actions that lead us to regions where $\widehat{f}$ or B are large (since either may contain optima), and at the same time obtain large refinements of the upper bound, i.e. a large amount of information about this bound. For more details regarding the computation of ρ and how the OCP is solved, please refer to Santejudean and Busoniu (2021).

3. ALGORITHM

While OOPA behaves well in experiments, it is computationally intensive and nontrivial to tune. Next, we provide a much simpler algorithm and investigate how much performance must be sacrificed to achieve this simplicity.

Like CDOO and OOPA, our new approach extends the DOO principle by building and refining with each new f-sample gathered the upper bound in (5), with the goal of focusing the search towards x^*. To tackle the danger of overcommitment, the robot continuously monitors the upper bound and function values with each new sample gathered. If the upper bound of the currently targeted state (the last-chosen maximal-B location) becomes lower than an f-sample previously seen by the robot, the current path is clearly suboptimal. In such a case, the robot updates its target to the current maximum of B. This idea is expected to save energy and time, and will be called FTW, as the robot Turns When a previous Function value becomes larger than the upper bound of the currently targeted position.

Figure 2 provides some intuition on the difference between the CDOO and FTW methods. The global maximum of the function is marked with a red star and denoted with f^*. The robot is heading towards the left endpoint of X, where initially the bound was maximal (marked with a blue star at the end of the blue dotted line). At step $k+3$, the refined bound at this initial target point becomes lower than the previous function sample $f(x_k)$, as shown by the horizontal dotted line, and thus FTW changes direction towards the right. In contrast, CDOO continues the suboptimal trajectory until reaching the position initially targeted, thus wasting energy and time.

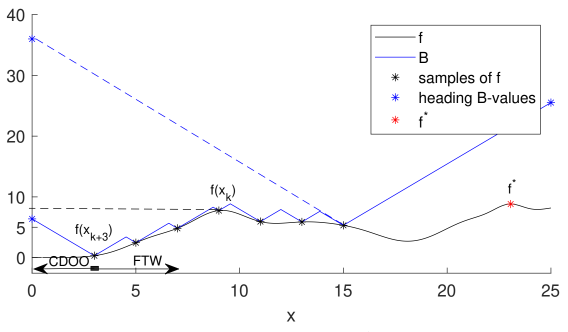

Fig. 2. The latest upper bound B (the saw-tooth drawn with continuous blue lines) envelopes f (black line) and has been refined with each new sampled gathered (black stars). When the upper bound of the current target point (blue star to the left) becomes lower than a previously seen sample (black dashed line), FTW changes direction, thus reaching sooner the global optimum compared to CDOO. The dashed blue line is the initial upper bound.

Algorithm 1 presents the steps of the FTW method. The method starts in a position x_0 and applies the dynamics in (1) to move towards the maximal B state, denoted by x_t (target state). At each iteration step k, the robot takes a new sample $f(x_k)$ and updates the upper bound B and f_k^*, the maximal f-value seen so far. If f_k^* is greater (or equal) to the bound value $B(x_t)$, a new target associated with the latest maximum value of B is chosen next. Then, the control action leading closest to x_t among all available actions:

$$u_k = \arg\min_u ||x_t - g(x_k, u)|| \tag{8}$$

is determined and applied so that the new state of the agent becomes x_{k+1}.

The algorithm stops when either the total number of iterations was exhausted or convergence was reached. Convergence is obtained when f_k^* becomes larger than all (or equal to some) B values. In that case, the position corresponding to f_k^* is returned as an approximation of the optimum.

Algorithm 1 FTW

```
Input: search space $X$, motion dynamics $g$, Lipschitz
    constant $M$, total trajectory steps $n$
 1: initialize sample set $S \leftarrow \emptyset$
 2: measure initial state $x_0$
 3: initialize target point $x_t = x_0$
 4: for each step $k = 0, \dots, n-1$ do
 5:     sample $f(x_k)$, add pair $(x_k, f(x_k))$ to $S$
 6:     update max $f$-value sampled $f_k^* = \max f(x_s)$
            where $(x_s, f(x_s)) \in S, s = 1, \dots, k$
 7:     update the upper bound $B$ (5)
 8:     if $B(x_t) \le f_k^*$ then
 9:         find max $B$-value state:
              $x_{b,k}^* = \text{argmax}_{x_b \in X} B(x_b)$
10:         update target $x_t = x_{b,k}^*$
11:         if $B(x_t) \le f_k^*$ then
12:             convergence occurred, break loop
13:         end if
14:     end if
15:     find action $u_k = \arg\min_u ||x_t - g(x_k, u)||$
16:     apply $u_k$, measure next state $x_{k+1}$
17: end for
18: return $\widehat{x^*} = \arg\max_{x_s} f(x_s)$, where $(x_s, f(x_s)) \in S$.
```

In the following sections, we study the performance of the algorithm experimentally. Analysis will be the focus of future work.

4. SIMULATION RESULTS

For the following simulations consider a simulated robot with the unicycle-like, motion dynamics in (1):

$$x_{k+1} = x_k + T_s \cdot u_{k,1} \cdot [\cos(u_{k,2}), \sin(u_{k,2})]^T \tag{9}$$

with $T_s = 1\,\text{s}$ denoting the sampling period and action $u_k = [u_{k,1}, u_{k,2}]^T$ giving the velocity $(u_{k,1})$ and heading of the robot $(u_{k,2})$. We will take as robot velocity $u_{k,1} \in [0, 0.2]\,\text{m/s}$ with the heading $u_{k,2} \in [0, 2\pi)$. The robot will travel at full speed towards the maximum-B state until the distance to this state becomes smaller than $0.2\,\text{m}$, the interval travelled at full speed in one sampling period. In the latter case, the agent adapts (lowers) the step size to reach the maximum B state. We do this because a faster moving robot explores sooner the search space, whereas an adaptable step size can lead closer x^*. The robot will search for the maximum in the space $X = [0, 4]\,\text{m} \times [0, 4]\,\text{m}$. To find the maximum B-state without increasing too much the computational time, we will evaluate B over a grid of states. In the sequel, the discrete states over which B is computed will be given by an equidistant grid X_{grid} of 21×21 points (placed at a spacing of 0.2 m on both axes).

When searching for the optimum, the agent will sample a function represented by the sum of five radial-basis functions (RBFs), each having the following coefficients: widths $b_i = [1.25; 1.25], [0.7; 0.7], [1; 1], [0.75; 0.75], [0.5; 0.5]$ and heights $h_i = [150, 255, 215, 100, 125]$. The centers of the RBFs (c_i) will be discussed later. Using these parameters, an RBF in 2D is:

$$R_i(x) = h_i \cdot \exp\left[-\sum_{j=1}^{d} \frac{(x_j - c_{ij})^2}{b_{ij}^2}\right] \tag{10}$$

where R_i represents the i^{th} RBF function, $d = 2$ is the number of dimensions of the RBF and $x = [x_1, x_2]^T$ denotes the evaluation state. A possible contour plot of f could be seen in Figure 3 on page 4. The Lipschitz constant corresponding to a given f is computed analytically as the maximum absolute derivative of f, found using the first two derivatives, so that it produces true bounds B.

4.1 Comparison to baselines

We will compare next the FTW method against two baselines: CDOO and OOPA. For OOPA we will consider the headings $u_{k,2} \in \{0, \pi/2, \pi, 3\pi/2\}$ leading the robot in the states up/down/right/left at next iterations. We restrict the robot headings during an OOPA sweep due to the high computational times required to solve the OCP.

For the baseline comparison, the RBFs from above are chosen with the centers positioned as follows: $c_i = [0.75; 1.5]$, $[2.75; 3.5]$, $[3.25; 0.75]$, $[1.25; 2.5]$, $[2; 0.25]$. Since the OOPA method does not provide a convergence test, we will consider as metric the distance traveled by the robot to reach as close as $\delta = 0.2$ m to x^* (as δ equals the grid discretization step of X for OOPA). For a fair comparison, we will take 15 randomly chosen points in X as initial positions for our agent.

Figure 3 reports the results. The distances till x^* for OOPA, CDOO and FTW (in this order, separated by slashes) are reported below each robot starting position marked with a blue 'x'. On average, FTW scores 10% fewer steps compared to CDOO while searching for x^* with δ accuracy. Compared to OOPA, FTW takes 30% more steps till the optimum, but achieves this using significantly lower computational times. While OOPA takes on average 1.85 s per iteration step, the time required by FTW (or CDOO) is below 0.01 s per step.

It would be instructive to study the evolution of the best sample seen so far, $\overline{f} = \max_{x_s}(f(x_s))$, as an average of the 15 runs, for each one of the three methods. Figure 4 reports the results for 250 trajectory steps. The FTW performance in terms of $\overline{f}$ is between OOPA and CDOO, OOPA being the algorithm which (on average) gathers faster higher function values and thus finds x^* sooner.

4.2 Influence of the Lipschitz constant

In this experiment we will study the impact of the landscape smoothness, i.e. the impact of the Lipschitz constant, on the performance of the FTW and CDOO methods. A quantitative study will be performed with 50

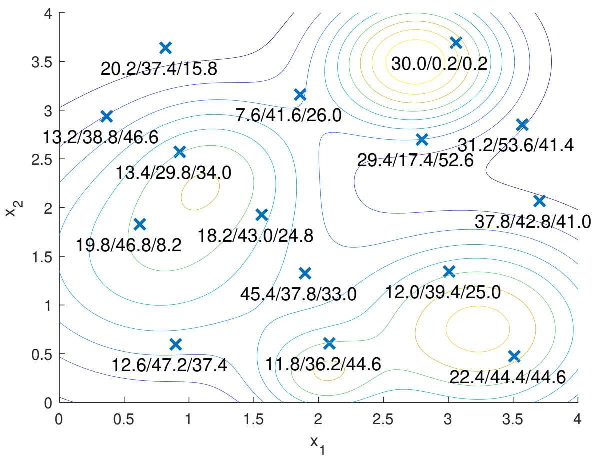

Fig. 3. Comparison between OOPA, CDOO and FTW methods in terms of steps until reaching as close as $\delta = 0.2$ m to x^*. FTW improves the CDOO average steps by roughly 10%, but scores 30% more steps than OOPA. However, FTW achieves this at much lower computational times compared to OOPA.

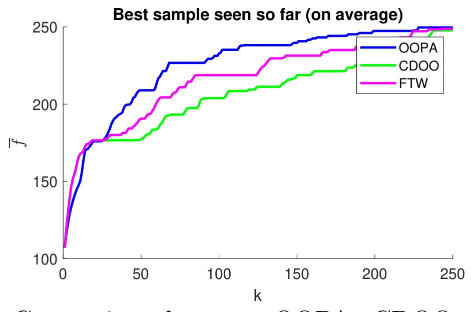

Fig. 4. Comparison between OOPA, CDOO and FTW methods in terms of best sample seen so far for the first 250 samples (on average) acquired during the baseline experiments.

different sets of center positions (placed in X) for each of the RBFs from above. Thus, 50 different landscapes result. For each such landscape, the width of the RBFs will range as follows: $b = \lambda \cdot b_i$, where the width factor $\lambda \in \{0.5; 1.0; 1.5; 2.0; 2.5; 3.0; 3.5; 4; 4.5; 5\}$. For each of the 50 landscapes, the robot starts in an initial position uniformly randomly chosen inside X. The evolution of the Lipschitz constant for each width factor is presented in Figure 5: the larger the λ, the smoother the landscape and thus the lower the value of M.

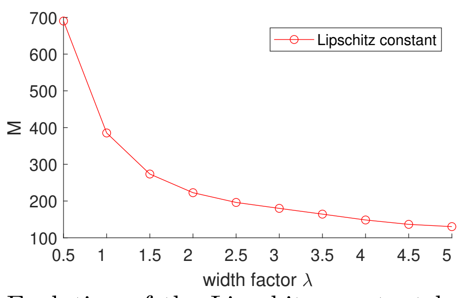

Fig. 5. Evolution of the Lipschitz constant based on the width factor of the RBFs.

Figure 6 shows the average number of steps until reaching convergence (left plot) and the standard deviation (right plot) for each of the width factor λ. FTW reaches convergence sooner on average compared to CDOO when the Lipschitz constant has moderate values. For large values of M, the robot will have to travel possibly the whole trajectory to the current target in order to lower the bound below a previously seen function sample. On the other hand, if the landscape is too smooth (close to a flat surface) both methods will need a lot of steps to reach convergence, as for a flat surface the whole state space (or all the points across X_{grid}) might need to be sampled to drop the bound below a sampled value of f. Finally, the FTW method brings slight improvements in terms of the variance of the performance, i.e. it is slightly more reliable.

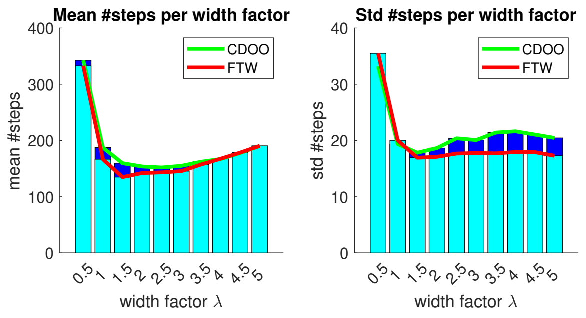

Fig. 6. Influence of the Lipschitz constant on the FTW and CDOO performance. The cyan and blue bars represent the mean (left) and standard deviation (right) of the total number of steps required by FTW and CDOO to reach convergence. The red/green lines highlight the evolution of the mean and std of FTW/CDOO for each one of the width factors (λ).

5. REAL-ROBOT EXPERIMENT WITH A ROBOTIS TURTLEBOT3

We verify the method empirically by providing a real-life experiment with a physical robot. In this experiment, a ROBOTIS TurtleBot3 searches for the lowest shade of gray position on a 2D printed surface.

The search space X will now be represented by a down-scaled version of Figure 3, specifically $X = [0, 2]^2\,\text{m}^2$. The width and center parameters of the RBFs have been reduced to half of their values used in the previous simulations, leading to a global maximum located at $p^* = [1.375; 1.75]$. The agent follows the nonlinear dynamics of (10) when searching for p^*. The meaning of the notations in (10) remains unchanged, with the only exception being the robot step size and sampling time, taken as $u_{k,1} = 0.1$ and $T_s = 3.5\,\text{s}$ to accommodate the above changes. Figure 7 provides a video still of the experimental setup corresponding to a view from the "left" of Figure 3.

We will describe next the acquisition of the function samples. For this task, a Raspberry Pi Camera Module V2 is deployed to acquire images from the physical map. The camera is pointing down at roughly 25 cm above ground and situated at $o = 20\,\text{cm}$ in the front of the robot center, i.e. of the middle point of the segment passing through the back wheels. The objective function f is evaluated as follows. Each acquired image is split into 9 smaller rectangles of equal width and height respectively. The

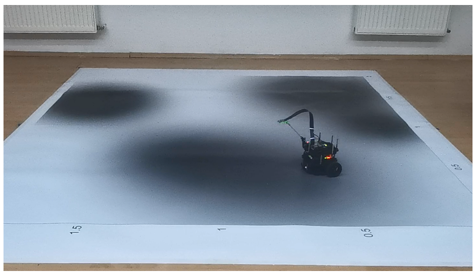

Fig. 7. Overview of the TurtleBot3 experimental setup.

center rectangle represents our Region Of Interest (ROI) over which a mean of the RGB values corresponding to each of its pixels is performed. The resulting value is then subtracted from 255 (as the RGB values are described by byte values), obtaining a so-called *grayscale sample.*

Note that FTW works with positions of the camera instead of the robot's center. Thus, a way to deal with the camera offset o is sought. Denote with $x_{r,k}$ and $\alpha_{r,k}$ the position and orientation of the robot at step k. We are searching for an action which moves the camera to a position x_{k+1} required by FTW at the next iteration. Equations (11-13) provide a possible solution:

$$\alpha_{r,k+1} = \text{atan2}(x_{k+1,2} - x_{r,k,2},\, x_{k+1,1} - x_{r,k,1}) \tag{11}$$

$$d = ||x_{k+1} - x_{r,k}|| - o \tag{12}$$

$$x_{r,k+1} = x_{r,k} + d \cdot [\cos(\alpha_{r,k+1}), \sin(\alpha_{r,k+1})]^T \tag{13}$$

where $x_{r,k+1}$ and $\alpha_{r,k+1}$ describe the pose of the robot at the next iteration of the algorithm.

The parameters of FTW are set as follows: trajectory steps $n = 200$ (equivalent to 20 m travel distance), and the Lipschitz constant is tuned empirically to $M = 500$. M is lower compared to the one used in the simulations due to a smaller range of possible f-values. Due to natural limitations (non-ideal lighting conditions, the gradients of the printed map not replicating exactly the RBFs in Figure 3, camera errors, etc.) the values of f are limited to the interval $[100; 238]$. E.g. the camera perceives the white color as a light gray.

The algorithm implementation was done using the Robot Operating System (ROS) and Matlab ROS Toolbox. The system architecture and the way its components (topics, messages, nodes, etc.) participate to the control procedure is presented as a simplified `rqt graph` in Figure 8.

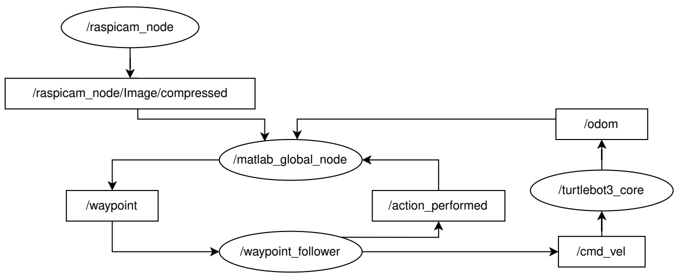

Fig. 8. Overview of the TurtleBot3 application workflow given by a simplified rqt graph.

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 FTW algorithm. At each trajectory step k, a new image is received from the `raspicam_node`. The image is processed and a grayscale value is plugged as the $f(x_k)$ input of the method. FTW returns the new reference position for the camera and uses the odometry information (`odom`) and equations (11)-(13) to compute the next control action. Each action is translated into a pose reference, which is further published by `matlab_global_node` on the `waypoint` topic.

The node `waypoint_follower` runs on the TurtleBot3 side. It subscribes to `waypoint` and implements the low-level control of the robot. Each time a new message is published to `waypoint` a callback function is run to immediately track the reference pose. Then, the translational and rotational speed of the robot are set and published to `cmd_vel` 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 to `waypoint_performed` to resume the FTW iterations. The cycle continues until the number of trajectory steps set at the beginning of the sweep is exhausted. A current limitation of the control procedure is that the robot cannot move backwards. Instead, it turns 180 degrees, moves the required distance and finally turns back. This issue will be dealt with in future work.

The experimental results are reported in Figure 9. In the upper plot, the B (light blue surface) is refined with each new trajectory sample (drawn with thin blue lines and 'x's) acquired from the initial robot position $[0; 0]$ (bold blue 'x'). The global maximum is approximated as $\widehat{x^*} = [1.34; 1.85]$ (thin red 'x') located roughly 10 cm away from the true $x^* = [1.375; 1.75]$ (bold red 'x'). The lower plot shows the evolution of the maximum f-sample seen till step k, $\overline{f} = \max_{x_k}(f(x_k))$, where the maximum value sampled is $\widehat{f^*} = 219$. $\widehat{f^*}$ is as close as 19 grayscale units to the true experimental maximum of $f^* = 238$. Thus, we have demonstrated the method also empirically.

A video corresponding to the FTW run in Figure 9 and the video still in Figure 7 is available online at `http://rocon.utcluj.ro/files/ftw_turtlebot.mp4`. The video was accelerated by a factor of 12 so that it fits in 1 min.

6. CONCLUSIONS

We presented a method for optimization of a quantity defined for the operating region of a mobile robot. The method takes into account the path of the robot, and provides performance that is better than a classical optimization baseline. Compared to an existing path-aware technique, at the cost of some performance loss the method is computationally much faster. The method was successful in a real-robot experiment.

In future work, the first priority is the analysis of the FTW algorithm: a characterization of how fast the sub-optimality decreases as the length of the robot trajectory grows. Real-robot experiments will also be performed. Algorithmic improvements may be sought, e.g. allowing the algorithm to deviate slightly from the path to the largest-upper-bound point in order to explore possible maxima nearby. Obstacle avoidance and multirobot extensions are also interesting. All these improvements should be performed while keeping the algorithm simple enough to be analyzed.

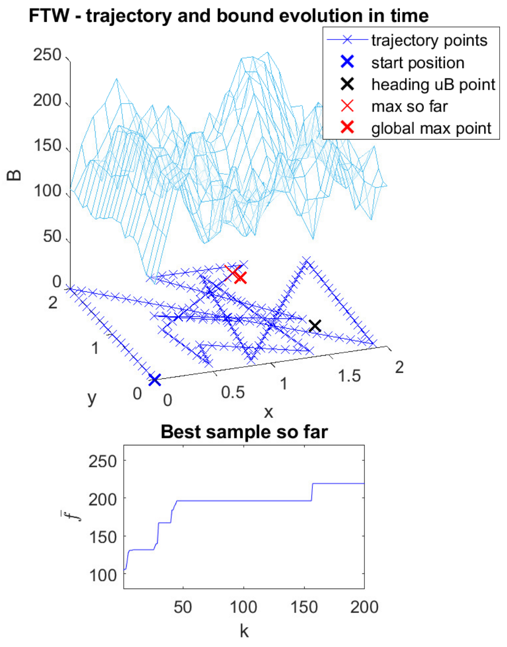

Fig. 9. Results of the TurtleBot3 experiment running the FTW method.

REFERENCES

Aggarwal, S. and Kumar, N. (2020). Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. *Computer Communications*, 149, 270–299.

Alexopoulos, C. and Griffin, P.M. (1992). Path planning for a mobile robot. *IEEE Transactions on systems, man, and cybernetics*, 22(2), 318–322.

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

Essa, K., Etman, S., and El-Otaify, M. (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. doi:10.4236/ojap.2020.92003.

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

Frazier, P.I. (2018). A tutorial on Bayesian optimization. Technical Report arXiv 1807.02811.

Horst, R. and Tuy, H. (1996). *Global Optimization—Deterministic Approach*. Springer Science & Business Media. doi:10.1007/978-3-662-03199-5.

LaValle, S.M. (2006). *Planning algorithms*. Cambridge university press.

Lawler, E. and Woods, D. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14. doi:10.1287/opre.14.4.699.

Lilienthal, A. and Duckett, T. (2004). Building gas concentration gridmaps with a mobile robot. *Robotics and Autonomous Systems*, 48, 3–16. doi:10.1016/j.robot.2004.05.002.

Munos, R. (2014). *From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning*. doi:10.1561/9781601987679.

Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Neural Information Processing Systems*, volume 24.

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