MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Path-aware search for single or multiple optima with a mobile robot

DIPLOMA THESIS

Author: **Tudor-Voicu Sântejudean**

Scientific supervisor: **prof.dr.ing. Lucian Buşoniu**

2021

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Approved,

DEAN
Prof.dr.ing. Liviu MICLEA

HEAD OF AUTOMATION DEPARMENT
Prof.dr.ing. Honoriu VĂLEAN

Author: **Tudor-Voicu Sântejudean**

Path-aware search for single or multiple optima with a mobile robot

1. **The problem:** Using a mobile robot, learn from samples an unknown function defined over its operating space in order to fulfill the following objectives: find the optima of this function in the least number of steps, and possibly gather data over a wireless network where the maxima represent wireless antennas.

2. **Content:** Presentation page, Declaration of authenticity, Summary of the diploma thesis, Contents, Introduction, Multiple Antenna Search, Path-Aware Global Optimistic, Conclusions and future work, Bibliography.

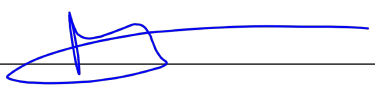
3. **Place of documentation:** Technical University of Cluj-Napoca.

4. **Advisors:** -

5. **Thesis emission date:** 08.07.2021

6. **Thesis delivery date:** 08.07.2021

Author signature 

Scientific supervisor signature 

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Declaration of authenticity

I, the undersigned, **Tudor-Voicu Sântejudean**, identified by the CI with series MM, number 875386, CNP 1990114245027, author of the thesis:

Path-aware search for single or multiple optima with a mobile robot

conceived for the final graduation examination

at the **Faculty of Automation and Computer Science**,

the Systems Engineering specialization in English,

of the Technical University of Cluj-Napoca,

2020-2021 July session,

declare on my own responsibility that this thesis is the result of my own effort and intellectual activity, based on my own research and on information obtained from the sources cited both in the text and in the bibliography.

I declare that this thesis does not contain plagiarized fragments, and that the bibliographic sources were used in conformity with the Romanian and international legislation regarding ownership rights.

I also declare that this thesis has not been previously submitted to any other university for a higher degree.

In the situation in which this declaration is proven to be false, I will endure all the consequences and administrative sanctions, consisting in the cancellation of the graduation exam.

Date

08.07.2021

Tudor-Voicu Sântejudean

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

SUMMARY

of the Diploma Thesis:

Path-aware search for single or multiple optima with a mobile robot

Author: **Tudor-Voicu Sântejudean**

Scientific supervisor: **prof.dr.ing. Lucian Buşoniu**

1. Requirements: Develop mobile robot algorithms that optimize the traveled distance when searching for optima points of an unknown function and possibly gather data buffers over a wireless network, where the maxima represent wireless antennas.

2. Solutions: A Gradient Ascent solution that maps the local optima of the function (informally referred to as ”antennas”) and gathers data buffers from these points is first developed. The method uses Local Linear Regression to build a local approximation plane and follows the direction of the plane's gradient (pointing to the greatest function increase) with maximum velocity. Since it uses derivatives, the method is particularly sensitive to noisy objective functions. Next, we extend the deterministic optimistic optimization (DOO), that naturally targets the global optima, and create the Path-Aware Optimistic Optimization algorithm (OOPA). OOPA makes use of a Lipschitz continuity assumption and the samples taken across the trajectory seen so far to build the function upper bound. This upper bound, refined with each new acquired sample, optimistically points to the maxima of the function. We formulate each decision of direction taking as an optimal control problem (OCP) and solve it using Dynamic Programming. The OCP objective is to maximize the cumulative rewards seen as weighted upper bound refinements across the considered trajectory and implicitly minimize the path travelled by the robot till optima. Note that OOPA does not include any data transmission objectives.

3. Obtained results: The Gradient Ascent was tested against two baselines derived from the Travelling Salesman Problem (TSP) and Lawnmower method. In the deterministic case, when no noise is present on the communication channel, the Gradient Ascent provides encouraging results in terms of the distance travelled to fulfill the objectives, its performance being closer to the TSP compared to the Lawnmower. However, in the presence of noise, these results deteriorate as the Gradient Ascent can converge suboptimally or even break, especially when the

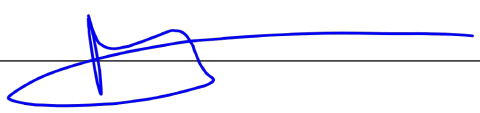
signal strength is drowned by noise. OOPA converges to the optima given a large enough number of function evaluations. It maximizes the long-term refinements of function upper bound and implicitly minimizes the travelled distance until optima is reached. The path-awareness property comes from the usage of all the previous trajectory samples when solving the OCP to take the next movement decision, unlike the Classical DOO (CDOO) approach that always commits to sampling the largest upper bound points. CDOO and Gradient Ascent have been used as baselines for the OOPA algorithm. As results show, OOPA found the optima in less distance compared to its baselines.

4. Tests and verification: The Gradient Ascent was tested on antennas generated uniformly random across the searching space from a fixed starting position, both in the case of deterministic and stochastic wireless transmission channels. OOPA was tested on fixed optima points in a deterministic setting, but with different starting positions and different types of functions (radial basis and pyramidal functions). Tests show that the developed methods perform better when compared to their baselines.

5. Personal contribution: The second chapter, presenting methods for multiple antenna search, is an adapted formulation of well-known algorithms available in the literature (gradient-based, TSP, lawnmower methods). The third chapter introduces a novel optimal control algorithm, Path-Aware Optimistic Optimization, that represents a main contribution of this work.

6. Bibliographical sources: Most cited resources are conference papers, lecture notes or books. Additionally, several sites and online articles presenting practical use cases relevant to our problem statement are being referenced throughout the work. All sources are attached in the last section called Bibliography.

Author signature 

Scientific supervisor signature 

Contents

1 Introduction

1.1 Context and motivation

Whether they are mobile or fixed, robots have become increasingly popular across various industries. They are employed in applications such as manufacturing [1], precision agriculture [2], education [3], surveilling and mapping of remote areas [4], etc. In this work we study the latter category, in which a mobile robot needs to plan its trajectory path to fulfill a series of navigation and transmission objectives. We define the following scenarios: *a*) Multiple Antenna Search, in which antennas spread throughout an arbitrary field need to be found and their data buffers gathered wirelessly by the robot in the shortest time; *b*) Path-Aware Global Optimization, in which the objective is to find only the true optimum of the sampled function as soon as possible, without including any transmission tasks. Due to energy and time constraints we design optimization techniques to solve the navigation and transmission tasks as soon as possible, while sampling an unknown function defined over a physical domain. Function optima can be seen as the maximum transmission point where an antenna might be situated [5], or optima points of other physical measurement such as forest density [6], pollutant concentration [7], etc.

Numerical optimization methods are a main topic of research due to the large number of problems they can address [8]. A popular class of such algorithms are the derivative based methods (Newton, Conjugate Gradient, Quasi-Newton, etc.) that use the function's gradient (and sometimes Hessian) to iteratively update on their initial estimate until reaching a local optima [9]. Other types of optimization methods are specifically designed to find the global optima of the objective function. e.g. optimistic optimization is a branch-and-bound technique that creates a search tree by sampling always the leaf with the highest upper bound value, associated with a subset that optimistically contains the function optima [10]. Multi-armed bandits are a set of sample-based optimization algorithms designed for the stochastic case, that can use a so-called upper-confidence-bound to minimize the long-term cumulative regret [11], [12]. Optimizing the collection of a quantity (represented by reward) across the trajectory performed by an agent represents an optimal control problem that can be solved through Reinforcement Learning [13]. Note that the performance of such learning-based algorithms is being constantly improved in terms of time complexity and tighter regret bounds, for instance [14], [15]. With such a diverse field, there is a good chance of finding a method that addresses the engineering problem at hand.

In this work we resort to a series of well-known optimization and optimal control approaches and combine them (where necessary) to solve the problems defined above. For *a*), a Gradient Ascent method has been developed and tested while *b*) was solved through an algorithm combining Dynamic Programming (DP) and Deterministic Optimistic Optimization (DOO). All these algorithms are described in-depth in the following sections and evaluated in extensive simulations to show that they solve the tasks mentioned above.

1.2 Problem statement

Given a mobile robot, it is required to find in the shortest time the optima of a function defined over the field on which the robot moves, while possibly transmitting/receiving data over a wireless network. This formulation is different from classical optimization problems in the sense that, due to dynamics constraints (limited velocity), the robot can only sample neighboring states to its current position. Thus, it is not possible to evaluate the function at arbitrary points of the space immediately one after the other, as that would require an arbitrarily large robot velocity. Moreover, sampling far-away states even at finite/small velocity without revisiting the trajectory and using the samples taken by the robot while reaching these new waypoints, would overcommit, as new information would become available and the old trajectory might become suboptimal. This makes the path of the robot important, especially in practical scenarios where energy and time requirements are present.

We identify some possible use cases of our problem: unmanned aerial vehicles (UAVs) searching for the largest bandwidth of surrounding antennas to transfer data faster, mapping of the ocean litter density to e.g. employ cleaning underwater robots, finding areas with high air pollutants in a city to restrict the traffic, etc.

Our work tackles two slightly different problems. In the first one, the robot aims to find all local maxima of the objective function. This can be seen as a search for multiple antennas (e.g. beacons), where the received signal strength indicator is the function to be optimized. In this case, it is interesting to add the objective of gathering the data buffers stored by these antennas using a wireless communication protocol. We call this scenario Multiple Antenna Search. Note that this scenario could also represent searching all litter hotspots or other similar use cases.

The second scenario is called Path-Aware Global Optimization, in which the robot targets only a global optimum. This can be seen as the maximum transmission point of an antenna, the maximal litter concentration in an area, highest traffic congestion area etc.

Even though the first scenario can cover the second one (indeed, by searching for all the local optima, the global one can be chosen as the maxima/minima of all these points), the second method finds the global maxima sooner as it does not enumerate all the local ones. Another difference lies in the implementation and robustness of the methods. As stated above, Gradient Ascent is used to solve the first scenario making use of derivatives to reach the points of interest. Intuitively, stochastic objective functions (e.g. noisy communication channels, scanned images, etc.) can lead to poor performance of gradient-based methods since such gradients are greatly affected by noise. Thus, a more robust algorithm that does not use derivatives is employed in the second scenario. It combines dynamic programming and deterministic optimistic optimization to quickly find the global optima. We call this algorithm Path-Aware Optimistic Optimization (OOPA). Note that OOPA does not include any transmission objectives yet and is mainly studied in a deterministic setting.

1.3 Contribution

The methods described in the second chapter represent reformulations and combinations of well-known algorithms (gradient-based, travelling salesman problem, lawnmower trajectory) adapted to our problem statement. The third chapter introduces a novel optimal control algorithm, Path-Aware Optimistic Optimization, that represents a main contribution of this work.

1.4 Thesis Structure

Next, Chapter 2 presents the Multiple Antenna Search. Chapter 3 introduces the Path-Aware Optimistic Optimization method, and Chapter 4 concludes the current work.

2 Multiple Antenna Search

2.1 Introduction and motivation

Given a mobile robot and a set of antennas spread throughout a field of known size, it is required to find their locations and download all antenna data buffers into the robot internal memory, in the least number of steps (or equivalently for constant velocity, in the shortest time) possible. Antennas (also referred as "transmitters" throughout the work) have unknown location, each storing different data buffers. The mobile robot must scan the field to find antennas and download data buffers while navigating. The transmission channel is wireless, with transmission rates being possibly affected by noise. As it is typical for radio protocols (e.g. Wi-Fi), the robot can transfer data from one antenna at a time while still being able to scan for surrounding antenna signals.

Similar learning problems in which the transmission and navigation objectives are imposed for unknown transmission rates have been previously studied in the literature. One example is the optimal control method of [16] that uses the Pontryagin maximum principle to optimize the time required by an agent to reach a waypoint while transferring data over a wireless network. Reference [17] tackles two different control problems: the transmission problem (PT), in which communication rates are arbitrarily shaped but deterministic; and transmission-and-navigation problem (PN), allowing stochastic rates that radially decrease around the antenna. Note that PN has as additional objective - navigation to an end position while emptying the data buffer - compared to PT. PT uses dynamic programming and supervised learning to solve the control problem and empty the buffer in the shortest time, given a set of surrounding antennas; while PN uses active learning to approximate the parameters of the rate function model and the optimal control of [16] to find the shortest path transmitting the data and reaching the required waypoint, given a single surrounding antenna. The learning procedure is done online based on the samples acquired across a single trajectory run (similar to the problems proposed in this work). Similar to our problem, the agent needs to transmit (equivalent to gathering) a data buffer over a wireless network in a field of unknown positioned antennas. Different to our approach, PT does not aim to find all the surrounding antennas, but only relay the data buffer of the robot once, in the minimum time possible. Different to PN, we aim to develop an algorithm that finds the optima without using a model of the transmission rate function (whose parameters are in turn learned through samples).

Algorithms employed in this chapter search for the antenna centers while transferring data into the robot's buffer at the same time. A series of algorithms such as: Gradient-Ascent, Travelling Salesman Problem (TSP) and Lawnmower method, have been adapted and integrated to suit our problem requirements. In short, Gradient-Ascent follows the antenna with the highest signal strength in the direction of the greatest rate function increase (given by its gradient) to transfer and find antenna locations. Travelling Salesman Problem has knowledge of the antenna positions, based on which it computes an optimal length tour to traverse and gather the data buffers. The Lawnmower method follows a lawnmower trajectory to approximate antenna positions using model-based equations while transferring the data. Extensive tests showcasing the parameter tuning and corre-

sponding results, both for the deterministic and stochastic rate functions, are presented later in this chapter.

An application of the multiple antenna problem is a field of wireless soil beacons monitoring the growing of the sweet corn. To maximize the crop harvesting, soil quality parameters (temperature, moisture, nutrients, etc.) are critical to be known, especially in the early germination stages. Beacons can gather such data and send it to surveilling UAVs acting as gateways [18], [19]. UAVs can collect the soil parameters provided by the beacons, whose positions are approximated based on the RSSI (Received Signal Strength Indicator). Note that the position of the beacons is unknown and underground, possibly planted with the planter along with the corn seeds. Knowing where the transmitters are located helps during maintenance cycles, as many wireless devices run on off-the-shelf batteries that need to be replaced after several months or years. This solution could help farmers to safely grow their corn crops by providing real-time, on-demand soil quality data and can bring cost reductions due to a better use of the irrigation systems or by facilitating the maintenance procedures (battery replacement, OTA firmware updates, etc.).

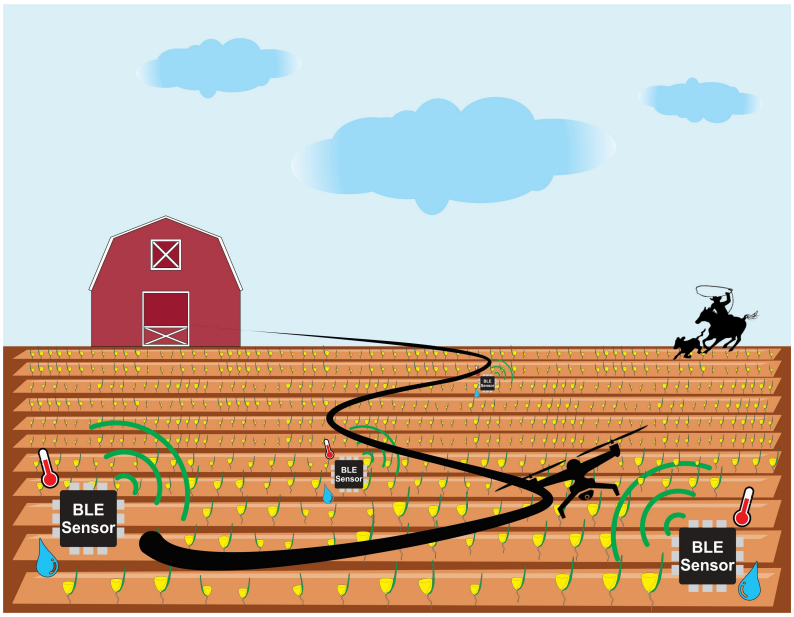


Figure 2.1. Possible use case of the multiple antenna problem: BLE (Bluetooth Low Energy) beacons monitor the sweet corn germination process by acquiring soil quality parameters (temperature, moisture). Measurement data gathered over a period of time is transferred to a surveilling UAV acting as a gateway. The UAV approximates the beacon positions based on the RSSI to help their localization during maintenance cycles (battery replacement, over-the-air firmware updates) and, essentially, for further information-gathering rounds.

2.2 Problem statement

We consider a 2D grid on which the agent is moving, thus let $p \in P \subseteq \mathbb{R}^2$ be the position. Robot movement on the grid is determined by the discretized actions $u_k \in U \subseteq \mathbb{R}^2$, taken at each given time step k. Simplified motion dynamics:

$$g : P \times U \to P, \; p_{k+1} = g(p_k, u_k) \tag{2.1}$$

can be defined using the nonlinear, unicycle updates:

$$p_{k+1} = p_k + T_s \cdot u_{k,1} \cdot [\cos(u_{k,2}), \sin(u_{k,2})]^T \tag{2.2}$$

where T_s is the sampling period and $u_{k,1}$ and $u_{k,2}$ respectively represent the velocity and robot heading, $u_k = [u_{k,1}, u_{k,2}]^T$. One can observe that such dynamics are of first order (there are no other motion related states, e.g. additional velocities).

The agent stores in its internal memory the data transferred from the field antennas, each such internal buffer evolving as follows:

$$b_{i,k+1} = \min\{b_{max},\ b_{i,k} + T_s \cdot r_{i,k}\} \in \mathbb{R} \tag{2.3}$$

where $b_{i,k+1}$ is the buffer corresponding to the i^{th} antenna (denoted by a_i, with i representing the specific antenna index) at the next step $k+1$; b_{max} is the preset max buffer size to be transferred and $r_{i,k}$ is the rate transfer function sampled from a_i, considered to be constant during the sampling period T_s.

Transfer rates vary based on the robot position and on the random fluctuations. Rates sampled from the surrounding antennas, $r_{i,k} \sim \mathcal{R}_i$ where $\mathcal{R}$ is a density function depending on robot position, $r_{i,k} \in \mathbb{R}_+^*$, can be seen as continuous, average values taken constant along a sampling period T_s. This leads to a discrete-time evolution of all buffers in the proposed algorithms.

A general model-based formula for transmission rates has already been defined in the literature [17]:

$$r_{i,k} = R_i \log_2 [1 + z_k \cdot S_i (p_k)] \tag{2.4}$$

where $S_i (p_k)$ represents the Signal-to-Noise Ratio sampled at position p_k and R_i is a constant specific to the antenna a_i (which can be computed empirically if no previous value is available). An SNR model is also available in [17]:

$$S_i (p_k) = \frac{K_i}{(||p_k - p_i|| + h_i)^\gamma} \tag{2.5}$$

where K_i is an antenna-specific constant (transmission power, normalization weight etc. of the antenna a_i), p_i is the antenna position, h_i describes the SNR shape and γ represents the path loss exponent.

Note: The SNR values decrease as robot moves further away from the transmitter. The decrease speed is influenced by the parameters h_i and γ. The greater the parameters, the steeper the decrease of the SNR and of the rate function.

Note: The parameters presented in formulas (2.4) and (2.5) (K_i, γ, h_i, R_i, $S_i(p_k)$, p_i, $r_{i,k}$) are, in general, antenna-dependent and can differ from one transmitter to the other. In this work, similar types of antennas are considered having a slight variation in parameters (to simulate real-life scenarios).

Note: z_k is a number drawn from a Rice density and acts as a weighting factor for the

SNR in the rate function formula. Its purpose here is to simulate the noise present on the transmission channel. The general formula of a Rice sample is given in (2.6).

$$z_k = \frac{1}{E_z} \left|\left[\begin{array}{c} z_1 + v \\ z_2 \end{array} \right]\right| \tag{2.6}$$

where z_1, z_2 are independent, normally distributed, random numbers, drawn from a zero-mean distribution with variance equal to 1; and $||*||$ represents the Euclidean Norm. To change the variance of z_k, the parameter v is tuned according to the problem requirements (as v takes smaller values, z_k varies more around its expected value). Coefficient E_z ensures that the expected value of z_k equals 1 and thus, the SNR expected value is $S(p)$.

The system's overall state is given by $x_k := [p_k, b_k]^T$, where b_k stores all buffers $b_{i,k}$ and r_k includes all sampled rate functions, $i = 1, \ldots, n_a$, with n_a denoting the total number of field antennas. Thus:

$$x_{k+1} = f(x_k, u_k, r_k) := \left(\begin{array}{c} g(p_k, u_k) \\ \min\left\{b_{max},\ b_{n_a,k} + T_s \cdot r_{n_a,k}\right\} \end{array} \right) \tag{2.7}$$

represents the system dynamics at any discrete time step k.

2.3 Methods

In this chapter we develop and study different approaches to solve the Multiple Antenna Search. As stated earlier, two goals are followed at the same time: transferring the data from the field transmitters (Transmission problem) and heading to the antenna centers to better approximate their positions (Navigation problem). All algorithms (Gradient-Ascent, TSP, Lawnmower) share the objective of transferring buffers, while the latter goal is mostly relevant in the Gradient Ascent approach (as it always heads to the strongest antenna signal). TSP and Lawnmower either know beforehand the approximate/exact transmitters positions (TSP) or solve model-based equations to find them (Lawnmower). These two methods will act as baselines for the Gradient-Ascent algorithm. Results will be discussed for deterministic and stochastic transmission rates. Before running any simulations, one can intuitively expect the algorithms to perform as follows:

$$TSP < Gradient\ Ascent < Lawnmower$$

The above inequality referrers to the total number of steps (or equivalently time, for constant $u_{k,1}$) required by the robot to carry out its tasks. In other words, it is expected to obtain lower times with the TSP approach compared to the Gradient Ascent and similarly for the Gradient Ascent to the Lawnmower method.

Note: As TSP algorithms return to their starting position after visiting all the target points, we will consider the robot performing a Gradient Ascent/Lawnmower sweep to be returning in its initial position after fulfilling the objectives.

2.3.1 Gradient Ascent

An intuitive approach would be to follow the signal strength of the most powerful antenna at each step. The gradient-based method uses this strategy to head to the strongest rate function sampled from the surrounding antennas, with the goal of reaching its center and transferring the buffer more quickly.

Therefore, at each step k the robot samples the antennas in range and stores pairs of the form $(p_k, r_{i,k})$. An approximate rate function $\hat{r}_{i,k}$ is then computed using Local Linear Regression (LLR), as follows. Linear regression is applied on the closest N neighbors to the current agent position, then an affine approximator $\alpha_i \cdot p + \beta_i$ is built ($\alpha_i \in \mathbb{R}^2; \beta_i \in \mathbb{R}$). Replacing in this formula $p = p_k$, $\hat{r}_{i,k}$ is found. Differentiating the form $\alpha_i \cdot p + \beta_i$ gives a gradient estimate at the local robot position, gradient that leads to the steepest ascent in the rate function. The gradient equals, in this case, the slope α_i and is followed by the robot with unit velocity. Note that as the maximum rate function with such a simple approximator form would be at infinity ($p_k \to [p_1, \infty]; [\infty, p_2]$ as $p_{k,1}; p_{k,2} \to \pm\infty$), the gradient ascent is mostly meaningful in the current position and the affine approximator needs to be recomputed at next iterations. This shows that adaptive step-size gradient ascent is impractical, as one should head straight (with maximum velocity) to the greatest rate function increase pointed by the gradient. Iterations of p_k heading to the closest antenna will stop once a convergence test succeeds: transmission rates are over a convergence bound (c_{bnd}) and the last, say, s_p rate samples have the mean lower than a preset convergence threshold (c_{thr}) compared to the current estimate $\hat{r}_{i,k}$ (both determined empirically based on algorithm results). Thus, a mean of the last s_p rate samples before reaching convergence is performed to get a better approximation of the antenna locations. Other tests using Backtracking Armijo or Optimal Gradient Armijo [9] were carried out, but the results obtained were poorer in terms of overall solution steps. Another test could be the exceeding of a preset k_{max} number of iterations per transmitter position search.

The tuning parameters of the Gradient Ascent method are:

- N – number of nearest neighbors for LLR;
- s_p – number of sample rates considered while testing for convergence bounds.

Intuitively, higher N and s_p work better for stochastic transmission rates, and lower values of these parameters can be chosen in the deterministic case, where faster convergence in a lower number of steps can be achieved, due to the absence of the noise on the communication channel.

Note: It is required that $N \geq 3$ to build the affine approximator (as it has 3 parameters: $\alpha \in \mathbb{R}^2,\ \beta \in \mathbb{R}$).

Note: While running the LLR, pairs of the form (p_k, r_k) corresponding to the closest N neighbors might not form linearly independent vectors (say, the robot is moving in straight line). The system's matrix is in this case singular and approximator's parameters cannot be found. Instead, we solve the system with the first 3 linear independent samples found or simply return the transmission rate of the nearest neighbor if any 3 rows have

linear dependencies. In the latter case, define the next action deterministically as follows:

$$u_k(1) = 1\text{m/s and } u_k(2) = k \cdot \frac{2 \cdot \pi}{N}. \tag{2.8}$$

The pseudocode of the Gradient Ascent is presented in Algorithm 1.

2.3.2 TSP

The Travelling Salesman Problem (TSP) is a well-known and widely studied method in the fields of combinatorial mathematics and computer science. It aims to determine the shortest route going through a series of so-called cities (nodes of a graph) while still returning to its initial city.

The Multiple Antenna Search represents an instance of the generic TSP due to several reasons. Antennas visited by the mobile robot are equivalent to the TSP's nodes through which the salesman must travel. Also, it is required in both cases that the "traveler" follows an optimal (as short as possible) path. Note that in the antenna problem their exact location is not necessarily available beforehand and must be learned from samples, while in the TSP case the node locations are fully known. The Multiple Antenna Search has an additional objective, i.e. the transmission of the data buffers, while the TSP focuses only on the navigation task.

In real-life applications, if one performs a Gradient-Ascent sweep, transmitter centers become available and the TSP approach can be used afterwards. This way, buffers are transferred faster and the robot returns sooner compared to the case in which another Gradient Ascent sweep would be run. This leads to savings in time and power consumption.

It is known that TSP is an NP-hard problem. Optimal solutions are computationally heavy to find and if one considers using brute force algorithms the number of possible routes to be compared for shortest length can reach up to $(n_a - 1)!/2$ (recall the total number of antennas denoted by n_a). Thus, optimization approaches have been employed to solve the TSP in reasonable amount of time. While these approaches might lead to a sub-optimal result (not necessary the shortest route), they are still close to the optimal path. We present next the linear search algorithm 2-Opt that uses search heuristics to optimize (decrease) the length of the routes followed by TSP.

Given an initial solution, possibly the route drawn by the Nearest Neighbors, 2-Opt looks for path length improvements by simply switching two arcs of the route. If the resulting length is lowered, the new change in arcs is kept and the target nodes are swapped. This approach continues until the route converges to a sub-optimal solution or a pre-defined number of improvements has been reached. Intuitively, 2-Opt method gets rid of the path crossings that can appear in the Nearest Neighbors trajectories, while still allowing the TSP algorithm to be executed quickly.

Next, a non-recursive 2-Opt implementation will be provided, similar to [20]. The pseudocode of the 2-Opt is presented in Algorithm 2.

Algorithm 1 Gradient Ascent

Input: g, field bounds, b_{max} max antenna buffer to transfer data from, c_{thr}, c_{bnd} convergence threshold and bound, s_p convergence samples, N nearest neighbors to apply LLR on, k_{max} max number of steps per antenna iteration, n_a total number of antennas.

measure initial state x_0, initialize total time steps $k = 0$
repeat
  sample $r_{i,k}$ from surrounding antennas, store pairs $(a_i, p_k, r_{i,k})$ in robot memory
  if any antenna a_i in range **then**
    $a_{cls} = \text{argmax}_{a_i} \{r_{i,k} |\ r_{i,k} \text{ in } (a_i, p_k.r_{i,k}) \text{ just sampled}\}$
  else
    $a_{cls} = \text{argmin}_a \{||p_k - p|| \mid p \text{ in all } (a, p, r) \text{ memorized pairs}\}$
    repeat
      compute p_{k+1} heading to a_{cls} and find u_k leading to p_{k+1}
      apply action u_k to move robot to p_{k+1}
      increment total time steps $k = k + 1$
      sample $r_{i,k}$ from surroundings, store pairs $(a_i, p_k, r_{i,k})$ in robot memory
      if new antennas a_i in range **then**
        $a_{cls} = \text{argmin}_{a_i} \{||p_k - p|| \mid p \text{ in } (a_i, p_k, r_{i,k}) \text{ just sampled}\}$
      end if
    until undiscovered antennas in range
  end if
  initialize antenna a_{cls} iteration steps $k_{cls} = 0$
  repeat
    compute the mean μ of the last s_p samples of r_{cls}
    if $(||\mu - r_{cls,k}|| > c_{thr}$ or $\mu < c_{bnd})$ and $k_{cls} < k_{\max}$ **then**
      apply **LLR** on neighbor pairs (a_{cls}, p, r_{cls}) to find $\alpha^T \cdot p + \beta$ approximator
      if $\alpha^T \cdot p + \beta$ approximator found **then**
        perform normalized **Gradient ascent**: $p_{k+1} = p_k + \alpha$
        find u_k leading to p_{k+1}
      else
        choose deterministic actions u_k with (2.8)
        apply action u_k to move robot to p_{k+1}
        sample $r_{i,k}$ from surrounding antennas, store pairs $(a_i, p_k, r_{i,k})$
      end if
    else if antenna location a_{cls} not marked **then**
      antenna a_{cls} location found, mark it on robot map
    end if
    if $b_{cls,\ k+1} < b_{max}$ **then**
      transfer data $b_{cls,\ k+1} = \min\{b_{max},\ b_{cls,k} + T_s \cdot r_{cls,k}\}$
    end if
    increment antenna iteration steps $k_{cls} = k_{cls} + 1$
  until $b_{cls,\ k} = b_{max}$ and antenna a_{cls} center found
  update total time steps $k = k + k_{cls}$
until all n_a antenna locations found

Algorithm 2 2-Opt

Input: n_a antennas and their positions.

1: create the antenna distance matrix D, its entries $d_{i,j}$ representing the distance between antenna i and j respectively
2: define σ_{max} maximum updates to tour (trajectory) T
3: create Nearest Neighbors tour T (always looking for the closest unvisited node in D)
4: **repeat**
5: initialize tour updates $\sigma = 0$
6: **for** $x = 1 : n_a - 3$ **do**
7: **for** $y = x + 2 : n_a - 1$ **do**
8: **if** $D(T(x), T(x+1)) + D(T(y), T(y+1)) > D(T(x), T(y)) + D(T(x+1), T(y+1))$ **then**
9: swap $T(x+1)$ and $T(y)$
10: $\sigma = \sigma + 1$
11: **end if**
12: **end for**
13: **end for**
14: **until** $\sigma = 0$ or $\sigma \geq \sigma_{max}$

The possibility of ending up with a suboptimal path length after applying 2-Opt exists. However, the 2-Opt optimization step could be applied on a more diverse set of tours T, not only on the Nearest Neighbors one, transforming an initial longer path into a shorter one. We will create a more diverse route using the Randomized Nearest Neighbors (RNN): instead of always picking the closest distance neighbor, the algorithm will consider multiple possible candidates (closest 3 neighbors for instance) and choose one of them randomly. The fewer possible candidates considered, the greedier the pick will be. 2-Opt will then be applied on the resulting route and the final path will be compared in length to other paths obtained after repeating the above procedure a pre-defined number of times (generating multiple so-called sub-optimal paths). In other words, a Repeated Randomized Nearest Neighbors (RRNN) method is employed. This has a greater chance of finding the optimal path or a very close solution to it.

We call Optimal Path TSP (OPTSP) the TSP algorithm having the route built using RRNN and 2-Opt. OPTSP assumes the positions of the transmitters to be known, so it is a model base approach. A simple and intuitive approach will be to follow the trajectory built using OPTSP and transfer the data from each heading antenna. The robot will remain in the antenna center if the buffer was not yet fully transferred.

Note: We will often use the standard TSP notation from now on, even though we are referring to the OPTSP algorithm. The RRNN and 2-Opt algorithms are implicitly used whenever the TSP is run.

The tuning parameters of the OPTSP method are:

- N_{cand} – number of nearest neighbors' candidates for RNN;
- k_{paths} – number of generated paths by RRNN to apply 2-Opt on.

The pseudocode of the OPTSP is presented in Algorithm 3.

Algorithm 3 Optimal Path TSP (OPTSP)

Input: g, n_a total number of field antennas and their positions, b_{max} max antenna buffer, k_{paths} number of RNN sweeps and N_{cand} number of nearest neighbor candidates.

measure initial state x_0
create distance matrix D based on x_0 and antenna positions a_i
create a near-optimal travelling tour T by running RRNN with 2-Opt k_{paths} times, considering N_{cand} candidates for RNN
initialize total time steps $k = 0$
repeat
    choose next heading antenna a_i based on T
    repeat
        sample $r_{i,k}$ from current heading antenna a_i
        if $b_{i,\ k} < b_{max}$ **then**
            transfer data $b_{i,k+1} = \min\{b_{max}, b_{i,k} + T_s \cdot r_{i,k}\}$
        end if
        if a_i reached **then**
            $u_k = 0$
        else
            find u_k leading to a_i location
            apply u_k to move to a_i
        end if
        update total time steps $k = k + 1$
    until $b_{i,k+1} = b_{max}$ and antenna a_i reached
until all n_a antennas visited

2.3.3 Lawnmower

The Lawnmower approach leads the robot on rectangular-shaped trajectories while the data buffers are transferred from the strongest antenna in range, at each time step k. These trajectories are described by the lawnmower's resolution and have a great impact on the navigation and transmission objectives, requiring empirical tuning. To find the transmitter positions, it is required that knowledge of the transmission function models is available beforehand. Knowledge of the SNR and rate function formulas (including the parameters K_i, h_i, γ, R_i of each antenna, all except the actual antenna locations p_i) is needed to be able to approximate the transmitter positions.

Starting from (2.5), we get:

$$||p_k - p_i|| = \left(\frac{K_i}{S_i\left(p_k\right)}\right)^{\frac{1}{\gamma}} - h_i =: \tau_k \tag{2.9}$$

As the robot directions are of rectangular shape, next positions p_{k+l} can written as follows:

$$p_{k+l} = \begin{cases} p_k + l \cdot [T_s \cdot u_k(1),\ 0], & \text{if robot moves on the x axis} \\ p_k + l \cdot [0,\ T_s \cdot u_k(1)], & \text{if robot moves on the y axis} \end{cases} \tag{2.10}$$

where $T_s \cdot u_k(1)$ represents the distance travelled by the robot in one sampling interval T_s; the l^{th} step from position p_k leads to p_{k+l}.

From (2.9) and (2.10) one can write the following equations:

$$\left(p_k\left(1\right) - p_i\left(1\right)\right)^2 + \left(p_k\left(2\right) - p_i\left(2\right)\right)^2 = \tau_k^2 \tag{2.11}$$

$$\left(p_{k+l}\left(1\right) - p_i\left(1\right)\right)^2 + \left(p_{k+l}\left(2\right) - p_i\left(2\right)\right)^2 = \tau_{k+l}^2 \tag{2.12}$$

The last equation is equivalent to:

$$\left(p_k\left(1\right) + l \cdot T_s \cdot u_k(1) - p_i\left(1\right)\right)^2 + \left(p_k\left(2\right) - p_i\left(2\right)\right)^2 = \tau_{k+l}^2 \ \ (6) \tag{2.13}$$

or equivalently

$$\left(p_k\left(1\right) - p_i\left(1\right)\right)^2 + \left(p_k\left(2\right) + l \cdot T_s \cdot u_k(1) - p_i\left(2\right)\right)^2 = \tau_{k+l}^2 \tag{2.14}$$

Subtracting (2.11) and (2.14) and performing a series of computations leads to:

$$p_i\left(j\right) = \frac{\tau_k^2 - \tau_{k+l}^2}{2 \cdot l \cdot T_s \cdot u_k(1)} + \frac{l \cdot T_s \cdot u_k(1)}{2} + p_k\left(j\right) \tag{2.15}$$

where $j \in \{1; 2\}$ and $p_k = [p_k(1),\ p_k(2)]^T$.

This semi-model-based approach will become useful in the case of Rice fluctuations, as the rates are highly influenced by the noise of the transmission channel. Pairs of the form (p_k, τ_k) and (p_{k+l}, τ_{k+l}) are plugged first in (2.11) and (2.12); then using (2.15) an approximation of the antenna position $\widehat{p_i}$ is computed. Performing a mean over multiple approximate values of $\widehat{p_i}$ will, ideally, give a better approximation of p_i.

Note: We informally define a 'sub-field' of the whole field to be a part of it described by the mower trajectory composed of one vertical sweep on the y axis and one horizontal sweep on the x axis. Figure 2.2 shows a 'sub-field' sample along with the mowing trajectory.

The tuning parameters of the Lawnmower method are:

- w_{mow} – (sub-field) mower trajectory width;
- h_{mow} – (sub-field) mower trajectory height;
- $u_k(1)$ – mower velocity while performing sweeps on the width and height of the field.

The step size and implicitly the number of steps the mower performs on the horizontal and vertical sweeps need to be carefully determined, otherwise antenna locations could be missed or buffers not fully transmitted (provided multiple antennas are close to each other in a given field region).

The pseudocode of the Lawnmower method is presented in Algorithm 4.

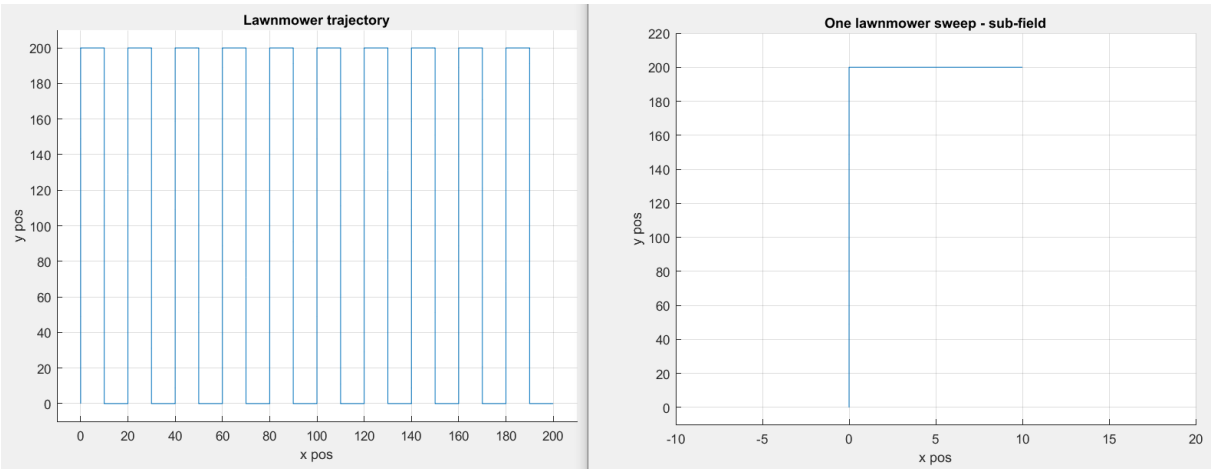


Figure 2.2. On the left, a possible robot lawnmower trajectory is given. A so-called 'sub-field' describing a vertical sweep followed by a horizontal one is plotted on the right.

Algorithm 4 Lawnmower

Input: g, w_{field}, h_{field} field width and height, SNR and transmission rate models (K_i, h_i, γ, R_i of each antenna), w_{mow} sub-field width, h_{mow} sub-field height, b_{max} max antenna buffer to transfer data from, $u_k(1)$ mower velocity.

measure initial state x_0
compute the number of sub-fields $s_f = \frac{w_{field}}{w_{mow}}$ or $s_f = \frac{h_{field}}{h_{mow}}$
for $l = 1,\ 2,\ \ldots, s_f$ **do**
    repeat at each time step $k = 0,\ 1,\ 2, \ldots$
        sample r from surroundings, store pairs (a, p, r) in robot memory
        if any antenna a_i in range **then**
            $a_{cls} = \text{argmax}_a \{r |\, r \text{ in } (a, p, r) \text{ just sampled with } b_{cls,k} < b_{max}\}$
            transfer data $b_{cls,k+1} = \min\{b_{max},\ b_{cls,k} + T_s \cdot r\}$
        end if
        find u_k leading to p_{k+1} on the sub-field trajectory (2.10)
        apply u_k
    until sub-field trajectory finished
end for
solve model-based equations of the form (2.15) using the stored samples to get an array of antenna position approximations
perform a mean on array positions to get the location of each antenna

2.4 Experiments and discussion for deterministic rates

In simulations, the motion dynamics (2.1) are used. Recall that $g : P \times U \to P$, $p_{k+1} = g(p_k, u_k)$. To run the algorithms, we take $P = [0; 200]^2$m as the field and U the set of actions of the form $u_k = [u_k(1), u_k(2)]$, where $u_k(1) \in \{0; 1\}$m/s is the robot speed and $u_k(2) \in [-\pi; \pi]$ is the robot heading or the direction pointed to by the local gradient estimate. The antenna buffers to be transferred will have a moderate size of 400Mbit. Consider initially the sampling time $T_s = 2$s.

To simulate the transmitters signals (SNR, rate function), all antennas will be modeled using (2.4) and (2.5). Thus, take the Rice coefficient $z_k = 1$ corresponding to the deter-

ministic case, $R_0 = 0.75$, $K = 10^4$, SNR shape $h = 1$ and the path loss exponent of the free space $\gamma = 2$. $n_a = 10$ antennas will be spread uniformly randomly throughout the field P.

In the next subsections, all algorithms will be examined for:

- trajectories followed by the robot,
- tuning of the parameters until (near-)optimal solutions are obtained in terms of total number of steps s_{total} (time efficiency for constant robot velocity) and average mean squared error $avgMSE$ (between antenna real positions and approximated ones),
- overall performance comparison for all methods discussed.

Note: The average mean squared error ($avgMSE$) is evaluated as follows:

$$\frac{1}{n_a} \cdot \sum_{i=1}^{n_a} ||p_i - \hat{p_i}||^2 \tag{2.16}$$

where p_i and $\hat{p_i}$ respectively represent the real and the approximated position of the i^{th} antenna.

2.4.1 Gradient Ascent

Trajectory Trajectories described by this algorithm are not necessary the shortest paths in terms of steps, because the agent is always heading to the strongest antenna signal in range, without considering the minimization of the overall trajectory length (unlike TSP).

For a given set of antenna positions, the ideal trajectory chosen by the Gradient Ascent algorithm is presented in Figure 2.3 (ideal by ignoring the approximations done by the LLR, for instance, that sometimes avoids straight line trajectories due to the linear dependencies of the samples $(p_k, r_{i,k})$, thus building longer routes).

Here, a pair of arcs (roads) crossing each other can be observed. This makes sense, as the robot will always choose the antenna with the strongest signal to head to (in the gradient approach), without considering lowering the overall length of the path. This drawback is solved in the TSP case, where the shortest path is obtained and followed by the robot.

Running the algorithm for the initial setting $N = 4$ and $s_p = 6$, gives good results: $s_{total} = 651$ and $avgMSE = 0.49$ (Figure 2.4).

Note: Robot position at given steps is marked with a disc, its color indicating the buffer size transferred so far from the current antenna (dark blue – empty buffer, dark red – full buffer). The segment linking the robot initial position and the uppermost transmitter is the returning direction of the robot (to its starting position). No data transfers are performed during this return.

Influence of tuning parameters The first parameter to be tuned is the number of nearest neighbors used by the LLR to find the affine approximator $\alpha \cdot p_k + \beta$. It is required that $N \geq 3$ to build the approximator (as it has 3 parameters: $\alpha \in \mathbb{R}^2$, $\beta \in \mathbb{R}$) and $N \leq 6$ to not make LLR too computationally heavy.

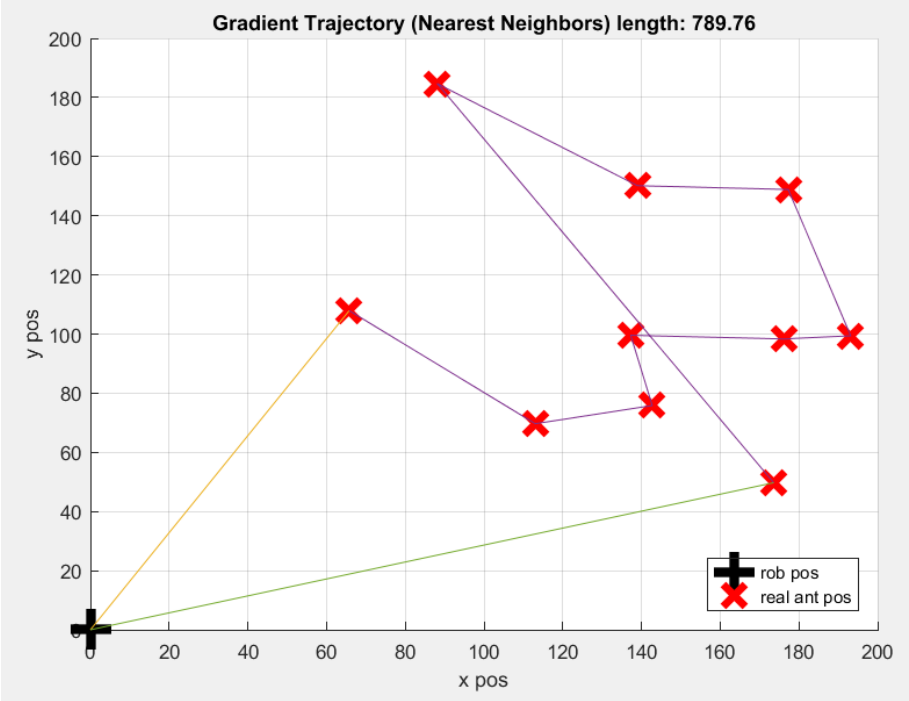


Figure 2.3. Ideal trajectory described by the robot when performing a Gradient-Ascent sweep on a given set of antenna positions.

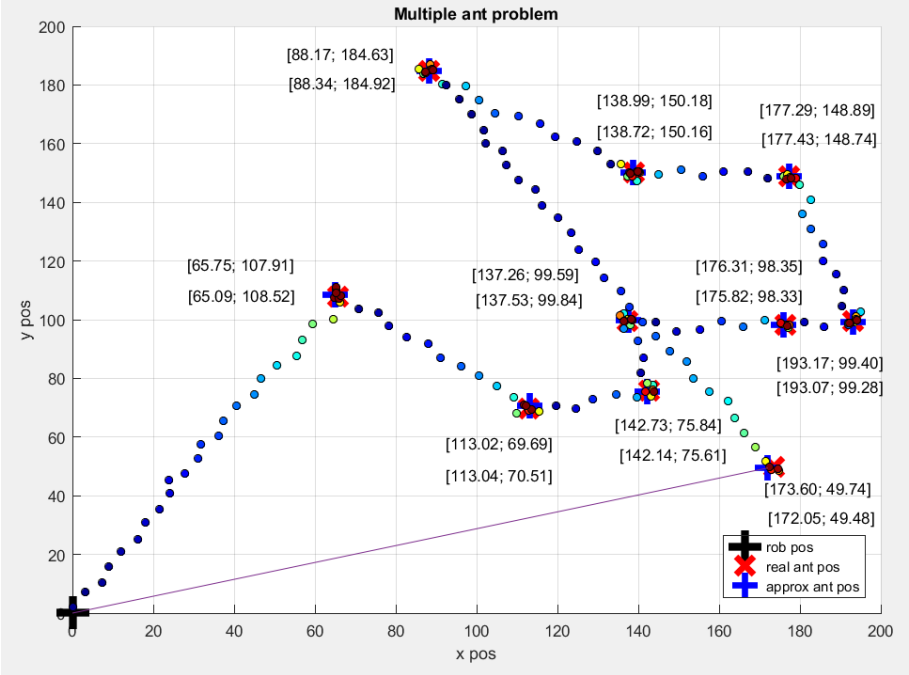


Figure 2.4. The trajectory and the evolution of the buffer sizes transferred from each antenna during a Gradient Ascent sweep. The colored disks show the position of the robot at each three sampling times and highlight the amount of buffer transferred from the heading antenna (dark blue represents an empty buffer and dark red a full one). The initial robot position is denoted with a black "+" mark, antenna real positions with a red "×" and their approximate positions with a blue "+" mark. The robot's return path is shown by the segment linking the last visited antenna and its initial position. Near each antenna, the real position written as "[x, y]" is situated above the Gradient Ascent's approximated position.

To get reliable insight regarding the best choice of N, 25 different algorithm runs will be performed, each on a different set of antenna positions generated uniformly randomly inside the field. Corresponding s_{total} and $avgMSE$ will be reported for each choice of N.

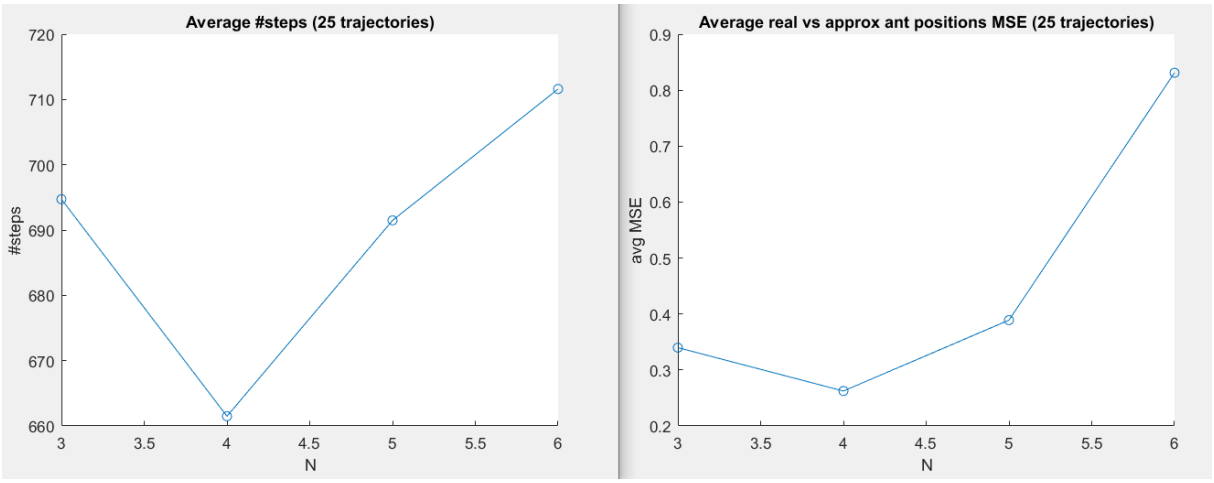


Figure 2.5. Average number of steps and mean squared error obtained running 25 Gradient Ascent sweeps, each on a different antenna positions setting. Results are obtained while tuning the number of nearest neighbors (N) used by the LLR.

As Figure 2.5 suggests, $N = 4$ is the best choice in terms of both s_{total} and $avgMSE$. On average, for $N = 4$, one Gradient Ascent run takes $s_{total} = 661$ and $avgMSE = 0.26$.

Next we study the parameter s_p used for testing the convergence (obtained when the robot is very close to the heading antenna's position).

Note that the antennas' maximum transmission rates, obtained when robot is on top of the antenna, are in this case around 10Mbit/s ($\pm 5\%$ variations). The convergence bound should be taken large enough to not allow the algorithm to converge too fast thinking that it has found the heading antenna, as at the beginning of the antenna searching rates are low ($\leq$ 1Mbit/s) and close in value to each other. Tests have shown that a value of $c_{bnd} = 2$Mbit/s can be used as convergence bound.

Plotting a single result of the transmission rates evolution till the convergence to an antenna position (Figure 2.6) shows that the chosen convergence bound $c_{bnd} = 2$Mbit/s makes sense, as around this value a more pronounced exponential increase in the rate function can be noticed (again, recall that the SNR form has an exponential decrease due to the factor γ at the denominator); an oscillating behavior of the rates occurs after robot gets in close proximity to the center and reaches the steady state output of the rate function, and, as it doesn't adjust its step size accordingly (only moves of length $u_k(1) \cdot T_S = 2$m), it will continue oscillating . Thus, a mean of an even number of s_p samples will be chosen.

Running the Gradient Ascent on 25 (different) antenna position settings, one can observe the impact of $s_p \in \{4; 6; \dots; 18\}$ on both s_{total} and $avgMSE$. s_{total} has a linear growth with the increase in s_p while $avgMSE$ has an exponential decrease as s_p gets higher. Cases of $s_p \in \{6; 8\}$ produce less than a half $avgMSE$ compared to the case of $s_p = 4$ and double compared to $s_p \in \{10; 12; \dots; 18\}$. As large s_p values $(10; 12 \dots)$ reach a plateau in

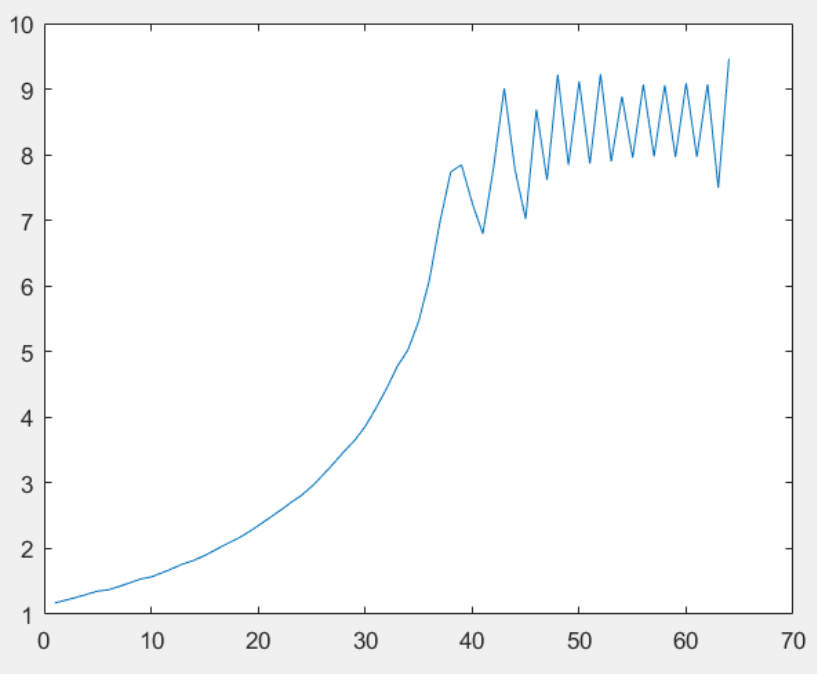


Figure 2.6. Evolution of the approximated transfer rates as robot approaches the antenna location. Rates increase exponentially until robot reaches a position close to the antenna's center. Afterwards, rates reach the steady state and begin to oscillate as the robot does not adjust its step size and continues to perform movements of constant length. A convergence test that detects the steady state response is defined to show that the antenna location was found.

terms of $avgMSE$ producing large path lengths, any values below $s_p < 10$ could work. To conclude, the value $s_p = 6$ for the deterministic case was chosen, as $s_{total} = 662.32$ and $avgMSE = 0.24$ obtained represent good results (lowering s_{total} might be more critical – energy consumption – compared to the antenna positions accuracy, in which for most applications an $avgMSE \approx 0.25$ could be accepted).

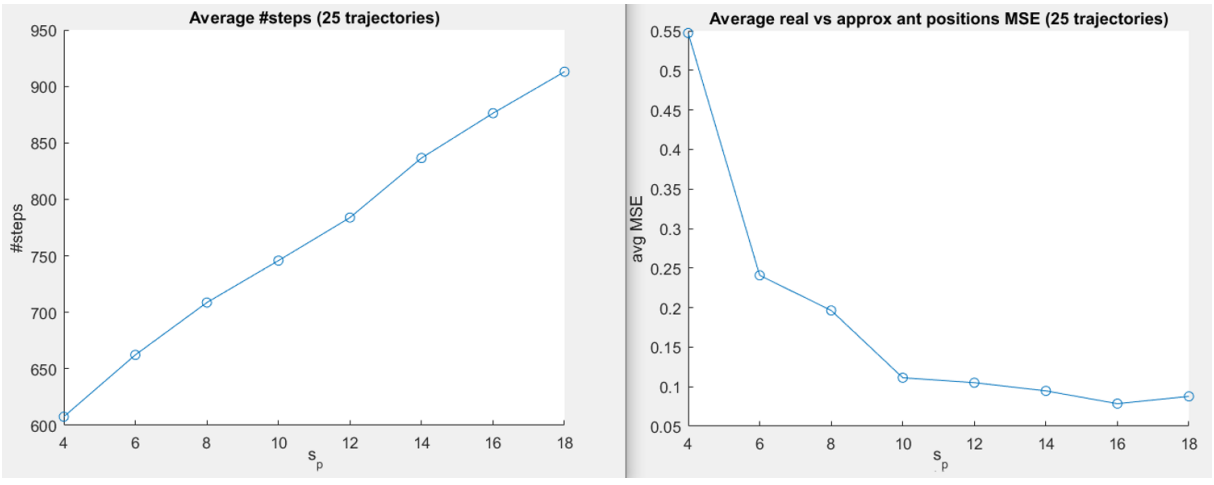


Figure 2.7. Average number of steps and mean squared error obtained after running 25 Gradient Ascent sweeps, each on a different antenna positions setting. Results obtained while tuning number of rates (s_p) considered for the convergence test.

According to Figure 2.7, a good parameters setting for the Gradient Ascent (deterministic case) is: $N = 4$ and $s_p = 6$.

2.4.2 TSP

Trajectory In the TSP case, the paths generated are near optimal when it comes to the distance to be travelled by the robot. A possible example of such a trajectory can be seen in Figure 2.8 (position of the transmitters is kept from the previous Gradient Ascent trajectory study; note that the path crossing was successfully removed).

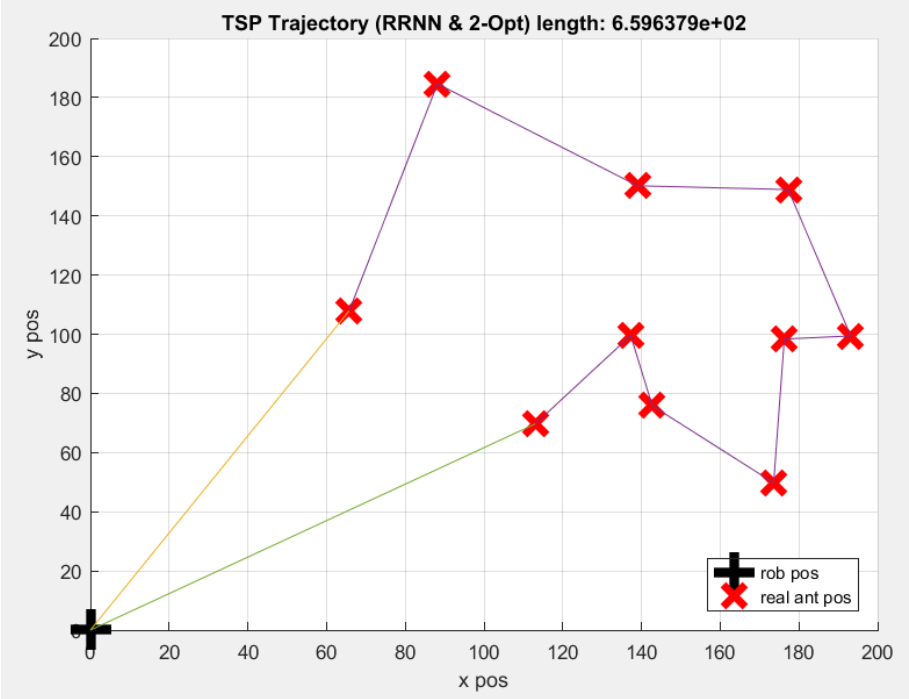


Figure 2.8. TSP trajectory drawn by the robot on a given set of antenna positions. Note that any trajectory crossings present in the Gradient-Ascent route were removed due to the use of 2-Opt and RRNN.

TSP results show $s_{total} = 476$, compared to the $s_{total} = 651$ obtained by the Gradient Ascent sweep, that is a decrease of 26.9%. Also, the ideal Nearest Neighbors path followed by the robot for the gradient-based algorithm is 20% longer than the route built using RRNN and 2-Opt (789.76 vs 659.64).

A sample of trajectory and buffers size evolution when using the TSP method is presented in Figure 2.9.

Influence of tuning parameters N_{cand} and k_{paths} affect the final near optimal path as follows: the greater the N_{cand} the less greedy the RNN generated tour will be and thus more varied routes will be built. By increasing k_{paths}, the chance of converging to the optimal path after applying 2-Opt increases. Tuning of these parameters can be done straightforwardly: $N_{cand} = 3$ is picked from [21] along with $k_{paths} = 100$ setting, obtaining good results (10% shorter paths on average compared to the Nearest Neighbors case). As in [21] the number of cities was 85, any number lower that this can use these settings as well (above simulations were run on 10 transmitters, "cities").

Note: Finding the optimal path for TSP using the above approach can be done before the robot starts its trajectory. Recall that in this case the antenna positions are known beforehand. The RRNN and 2-Opt efficiency is therefore not critical for the online simu-

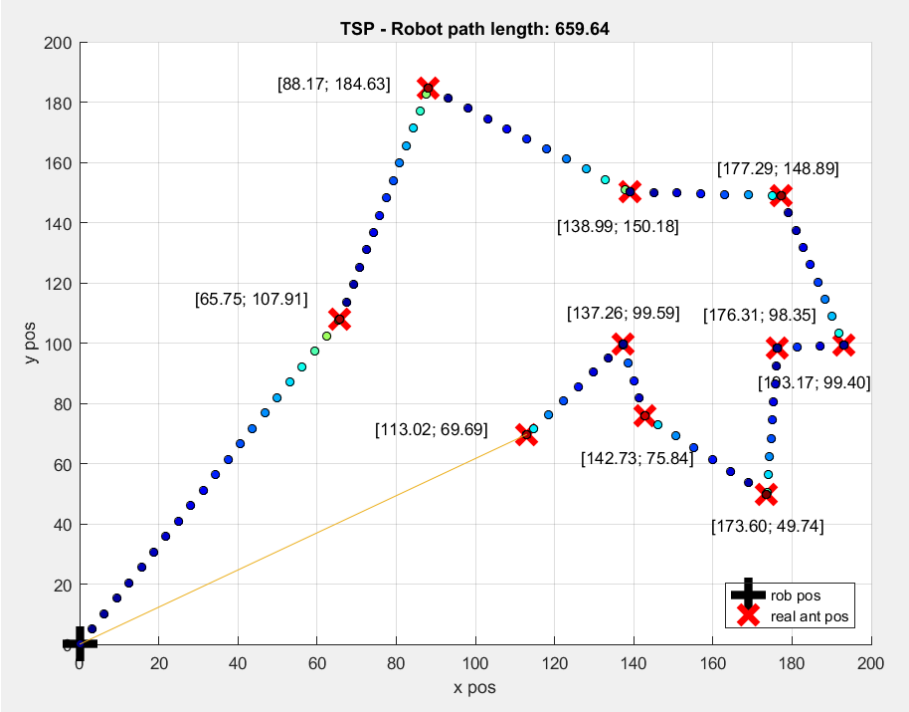


Figure 2.9. Evolution of the buffer sizes transferred from each antenna during a TSP sweep, given a set of antenna positions.

lation and tuning the parameters of this method can be made directly based on "expert knowledge" available in the literature and common sense.

Running TSP on 25 different antenna position settings (quantitative analysis on above parameters) show an average of $s_{total} = 449.5$. Compared to the Gradient Ascent average $s_{total} = 662.32$, TSP achieves 32.2% less trajectory steps.

2.4.3 Lawnmower

Trajectory The path described by the robot is the same as the one presented in Figure 2.2 with the only difference that the robot returns to its initial state (like TSP and Gradient Ascent algorithms). Thus, the resulting path has a total of $s_{total} = 1200$, that is about 2 and 3 times more steps (on average) compared to the Gradient Ascent and TSP methods respectively.

Note: In the deterministic case, the Lawnmower method finds the transmitter centers solving model-based equations, so all centers approximated are in fact the actual, real positions of the antennas and thus $avgMSE = 0$. It will make more sense to study the $avgMSE$ in the fluctuations case.

Influence of tuning parameters The parameters have been tuned empirically to ensure that the data buffers are transmitted before the robot finishes its trajectory. The parameter values obtained are: $w_{mow} = 20$, $h_{mow} = 200$ and robot velocity $u_k(1) = 1\text{m/s}$.

The evolution of the buffer sizes can be seen in Figure 2.10, along with each change in antenna id that the robot is transferring data from "$[< ant_{id} >]$:", at any given time. Also, note that "$(< ant_{id} >)$" represents the id of the antenna plotted near its label. Once

all buffers have been transferred, label '[0]:' is plotted, meaning that no more transfers are occurring. Disk colors and meaning remain unchanged from previous simulation plots.

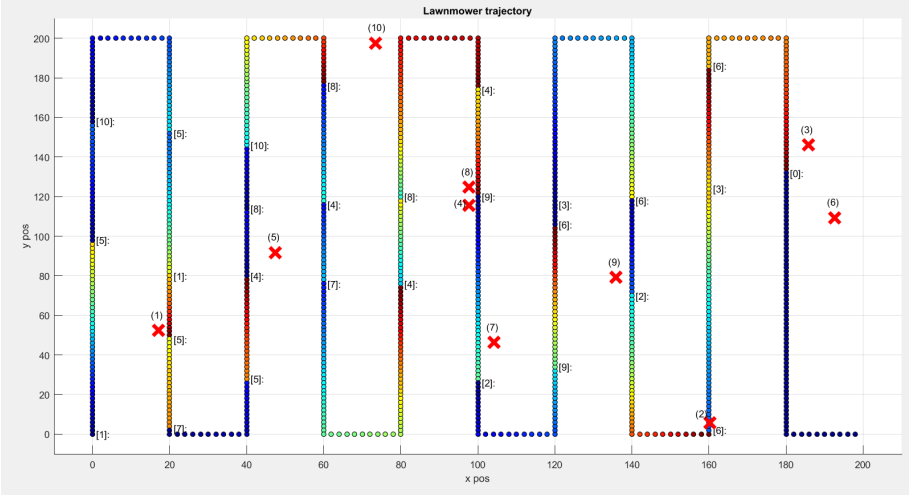


Figure 2.10. Trajectory and evolution of the buffer sizes gathered by the robot during a Lawnmower sweep.

Note: Since the method parameters were tuned empirically and the transmitters positions are generated randomly (uniformly distributed), the robot does not finish the data transfers exactly at the end of the lawnmower trajectory, but earlier. If the robot successfully carries out this task, it simply follows the trajectory without transferring data but still acquiring and storing in memory pairs of the form $(p_k, r_{i,k})$, later used for estimating antenna centers (useful especially in the case of stochastic rate functions).

Both the Lawnmower and TSP use to some extent "expert knowledge" and can therefore be considered model based methods. The Lawnmower method knows the function rate model and has access to the parameters K_i, h_i, γ, R_i of each antenna, while TSP requires that antenna positions to be available. Comparing the results, these algorithms give valuable insight and act as baselines for the unknown rate model and transmitter positions Gradient Ascent. Concluding:

$$TSP < Gradient\ Ascent < Lawnmower$$

in terms of s_{total}, all methods being able to fully transfer the data buffers from the antennas.

2.5 Experiments and discussion for stochastic rates

To simulate the presence of the noise on the transmission channel, Rice coefficients will be chosen in the rate function formula (2.4). A moderate noise level is obtained by taking $v = 15$, in which case most of the probability mass falls into the interval $z_k \in \{0.75; 1.25\}$. This is about 25% variation compared to the deterministic case (or the expected value).

2.5.1 Gradient Ascent

First, we start by running the algorithm keeping the parameters tuned in the deterministic case, i.e. $N = 4,\ s_p = 6$. Recall the sampling time value of $T_s = 2$s.

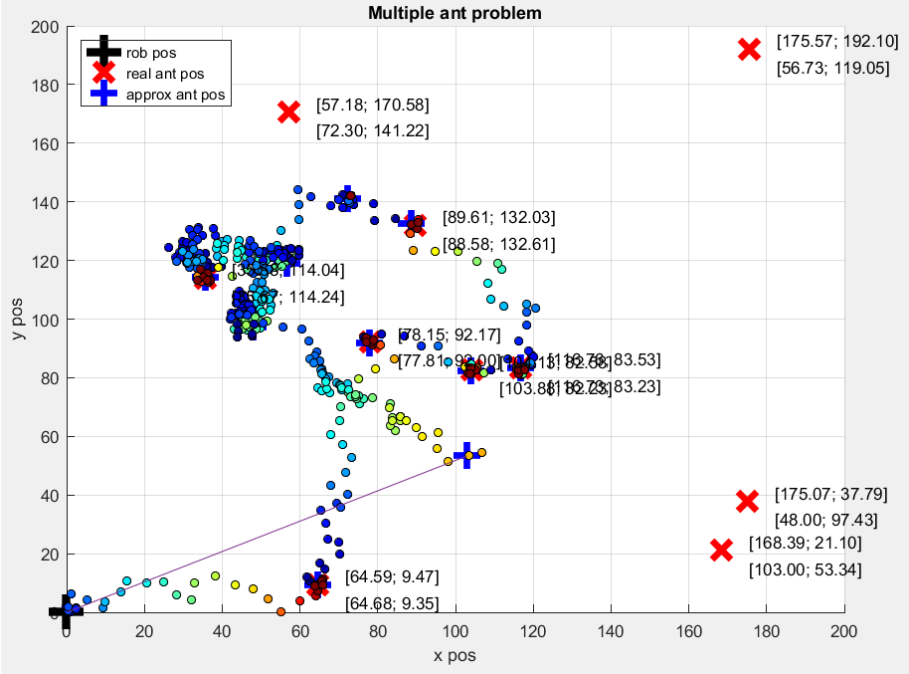


Figure 2.11. Trajectory obtained running the Gradient Ascent method in the fluctuations case, given a set of antenna locations. Keeping the parameters setting from the deterministic case gives poor results as there remain unfound antenna locations. Intuitively, the algorithm sampling time $T_s = 2s$ needs to be increased.

Results show $s_{total} = 1357$ and an $avgMSE = 4556.95$, which are poor results. Unfortunately, even if one increases N or s_p results do not become significantly better (most antennas remain undiscovered).

As an intuition, for lower values of the approximator's gradient, i.e. far away antennas that lead to low transmission rates, the gradient is drowned by noise. Also, if the step size of the mobile robot has a low value, the chance of reaching the antenna position in the presence of noise decreases drastically. This is because only samples that are taken farther away from each other provide informative enough data and can mitigate the noise. We express the step size as follows: $u_k(1) \cdot T_s$, where $u_k(1) = 1\text{m/s}$ is the fixed robot max speed. A simple and intuitive approach to solve the above shortcoming to noise would be to increase the sampling time, e.g. $T_s = 3$.

We run the simulation this time with the setting $T_s = 3\text{s}$ and obtain $s_{total} = 970$ and $avgMSE = 20.79$. This time results get closer to the deterministic case. Thus, it could pay off studying a larger T_s in the fluctuation case. Note that these $s_{total} = 970$ are for larger step size of $u_k(1) \cdot T_s = 3\text{m}$, and as $T_s = 3\text{s}$, they are translated into $time = s_{total} \cdot T_s = 970 \cdot 3 = 2910\text{sec}$. Figure 2.12 shows the results.

Algorithm sampling time will be quantitatively analyzed in the interval $T_s \in \{2; 2.2; \dots; 3.8\}$ to see what value would be the most appropriate. For this, 10 different (uniformly distributed random) antenna positions have been created and each such setting will have the Gradient Ascent method run for 20 times. Results are presented in Figure 2.13.

However, recall that larger steps lead to higher time spent by the robot on the field. A

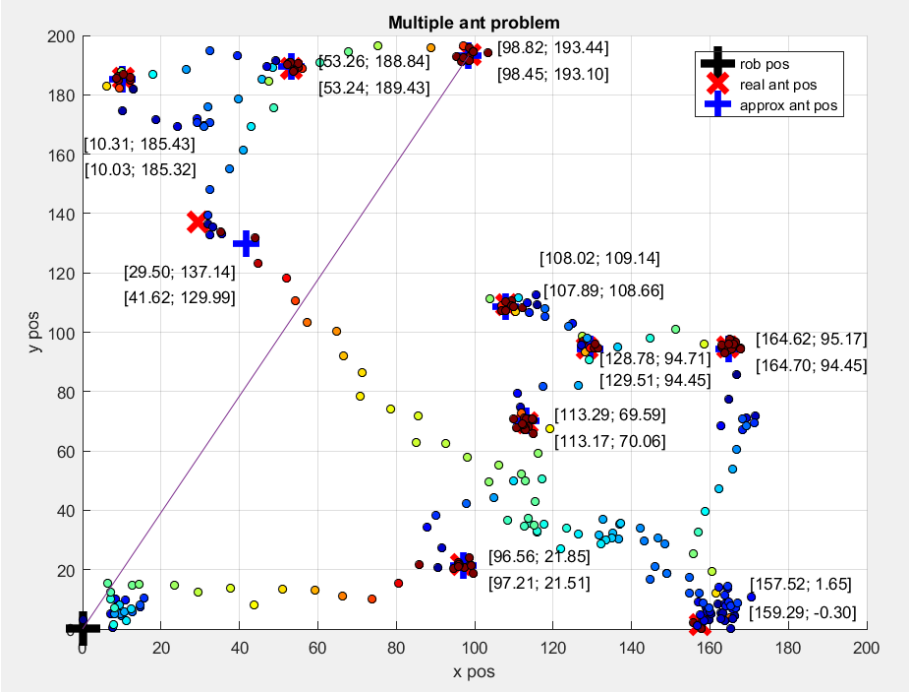


Figure 2.12. Trajectory obtained running the Gradient Ascent method after increasing the sampling time to $T_s = 3s$. All antenna locations were found, however, the position accuracy could be improved by retuning the parameters.

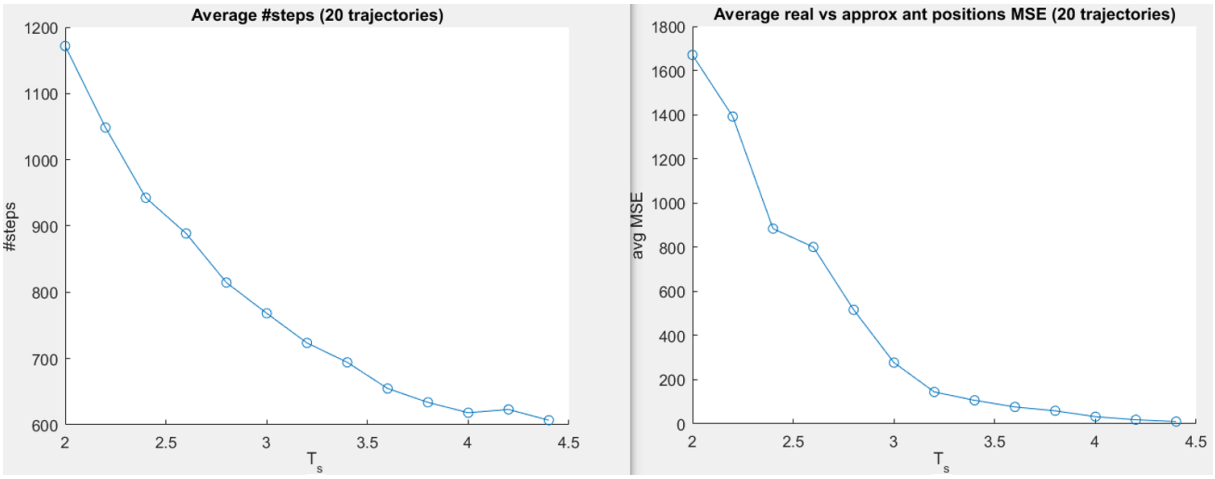


Figure 2.13. Average number of steps and mean squared error obtained running 20 Gradient-Ascent sweeps, each on 10 different antenna position settings. Results obtained (on a noisy communication channel) while tuning the algorithm sampling time (T_s).

quick conversion from s_{total} to time(seconds) leads to the following graphs in Figure 2.14.

Thus, values of $T_s \in \{3; 3.2\}$ can work well both in terms of s_{total} and $avgMSE$. Value $T_s = 3$ will be chosen because it results in less overall trajectory time, as one can decrease the $avgMSE$ by increasing N. The impact of N will be studied next, taking $N \in \{4; 5; 6; 7; 8\}$ and running the algorithm on the same 10 antenna position settings as above. Intuitively, the larger the N, the less the $avgMSE$ becomes (as it better rejects the noise).

Choosing $N \in \{7; 8\}$ leads to good results. $N = 7$ will be taken so that $avgMSE = 9.46$

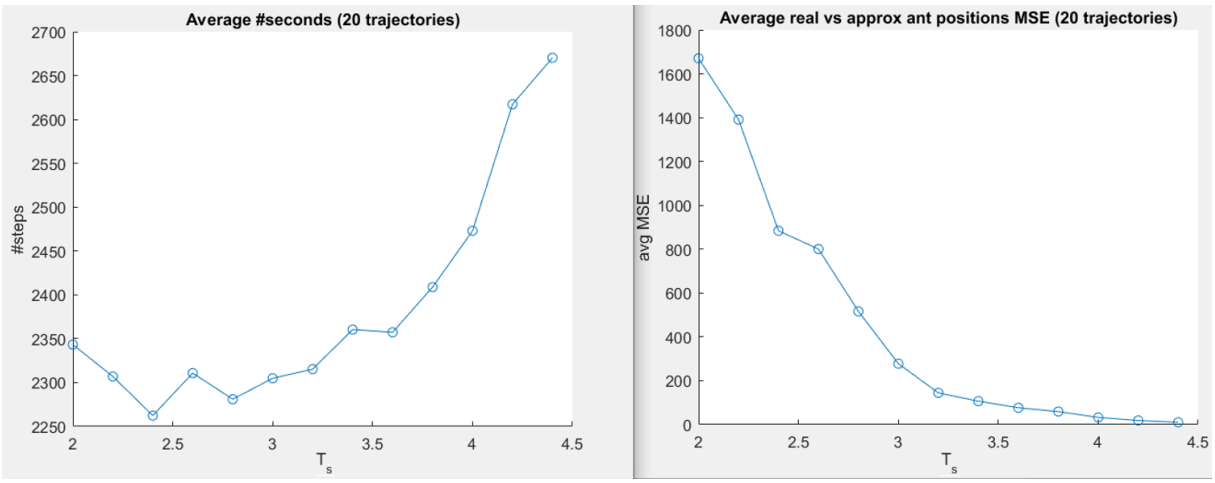


Figure 2.14. Average number of seconds and mean squared error obtained running 20 Gradient Ascent sweeps, each on 10 different antenna position settings. Results obtained (on a noisy communication channel) while tuning the algorithm sampling time (T_s).

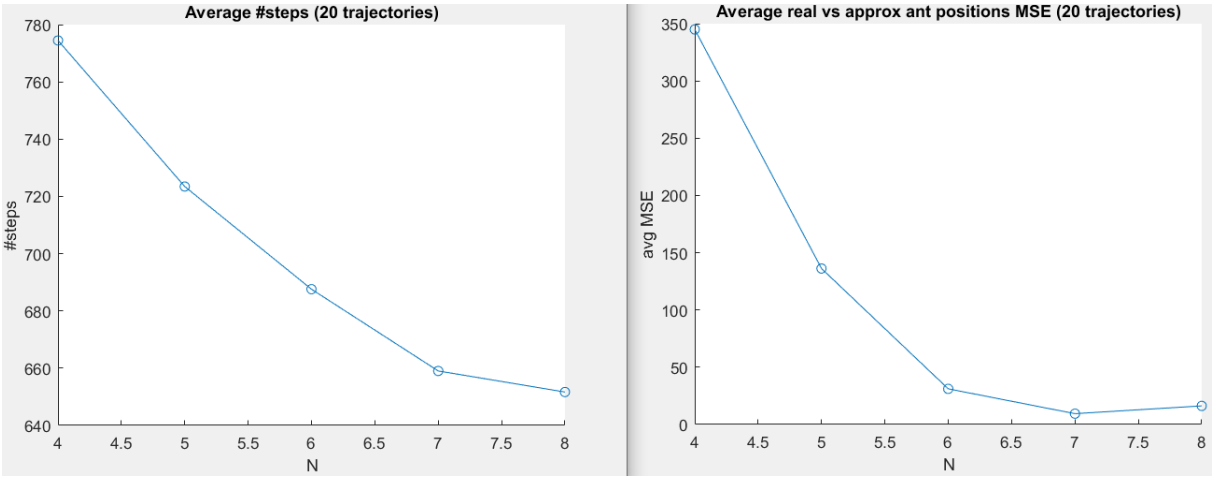


Figure 2.15. Average number of steps and mean squared error obtained running 20 Gradient Ascent sweeps, each on 10 different antenna position settings. Results obtained (on a noisy communication channel) while tuning the number of nearest neighbors (N) used by the LLR.

is 20 times smaller compared to the previous case when $N = 4$ (≈ 210 $avgMSE$). Note that due to the stochastic behavior of the SNR (Rice noise), s_{total} and $avgMSE$ can differ a bit from one simulation batch to the other.

The number of samples considered while testing the convergence, s_p, will be tuned next. Consider the simulation running for the same settings as above (20 different runs for 10 different antenna position settings) and $s_p \in \{4; 6; 8; 10; 12\}$.

A good tradeoff between s_{total} and $avgMSE$ would be achieved for $s_p = 6$, for which the results are similar to the ones obtained at the previous step (as there $s_p = 6$ was also a good setting in the deterministic case).

To conclude, parameters obtained after the tuning process are $T_s = 3$s, $N = 7$, $s_p = 6$. $avgMSE \approx 10$ and $s_{total} = 667.33$. In terms of trajectory seconds, $time = s_{total} \cdot T_s = 2002$s compared to $time = s_{total} \cdot T_s = 662.32 \cdot 2 = 1324.6$s obtained in the deterministic case. That is about 33% more time required by the robot to finish transferring the buffers

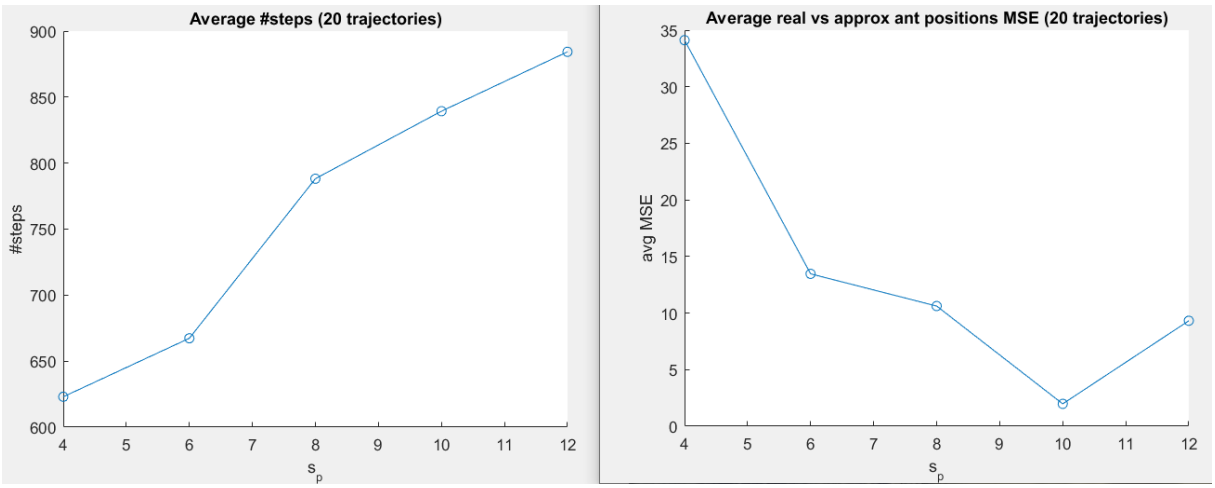


Figure 2.16. Average number of steps and mean squared error obtained running 25 Gradient Ascent sweeps, each on 10 different antenna position settings. Results obtained (on a noisy communication channel) while tuning the number of rates (s_p) considered for the convergence test.

and approximate the centers. $avgMSE$ on the other hand, increased from 0.24 to about 10 which represents roughly 3m in error from the real antenna positions. The Gradient Ascent performs poorly in the fluctuations case when estimating the transmitter centers, since it uses derivatives of the rate function that (in the stochastic case) are noisy. If one requires better position estimates, $avgMSE = 1.98$ can be obtained for larger $s_p = 10$, but at the cost of 25% more trajectory steps.

Lastly, recall the antenna locations setting chosen for experiments in Figure 2.12. It would be interesting to compare the improvements after tuning the parameters. For this, 20 different trajectory sweeps were performed on the same antenna positions. One such trajectory is presented in Figure 2.17.

Compared to the initial simulations, the trajectory in Figure 2.17 shows noticeably straighter paths drawn by the robot while heading to a given antenna. Results show an $avgMSE = 6.91$ and $s_{total} = 742.10$, more accurate antennas' localization but at the cost of more s_{total}. Two conclusions can be drawn: results are highly dependent on the antenna locations setting and the Gradient Ascent trajectory is highly influenced by the fluctuations, especially when inter-antenna distances are large.

2.5.2 TSP

50 trajectories will be run to estimate s_{total} and then results will be compared with the Gradient Ascent and the deterministic TSP. $T_s = 3$s is chosen for the simulations to make the comparison relevant for the stochastic case.

Results show $s_{total} = 303.66$ average steps per 50 different antenna position settings. This is almost half (55% less) compared to the Gradient Ascent $s_{total} = 667.33$. Translated in time $time = s_{total} \cdot T_s = 303.66 \cdot 3 = 910.98$ sec, while in the deterministic TSP case $time = s_{total} \cdot T_s = 449.5 \cdot 2 = 899$ sec, virtually similar results. This makes sense as in the TSP case antenna centers are known beforehand and there is no need for the robot to learn them from samples.

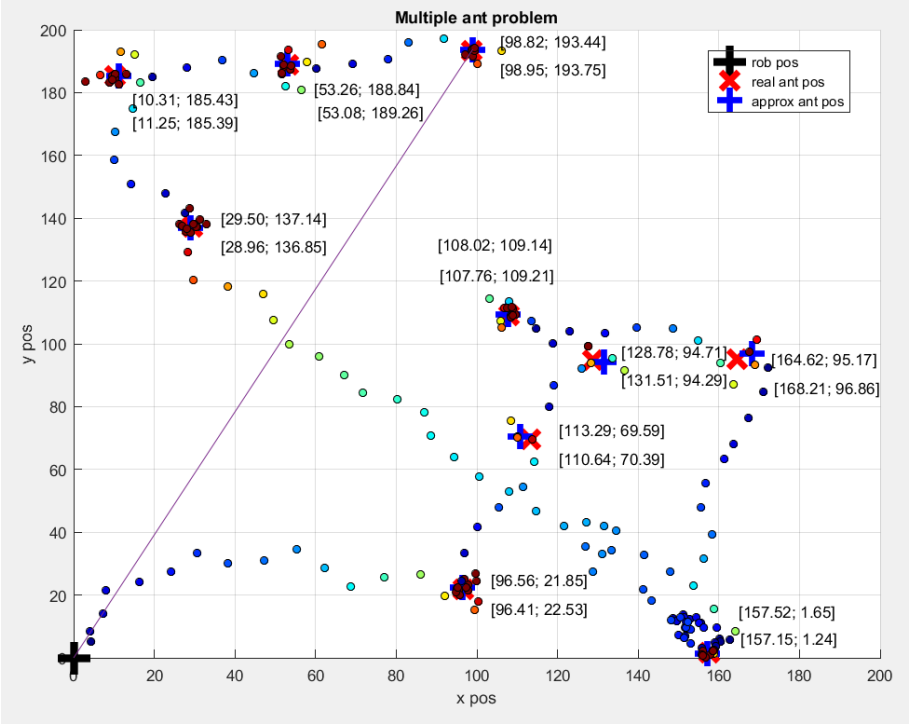


Figure 2.17. Trajectory obtained by running the Gradient Ascent method after parameters tuning. Antenna locations setting was kept from the initial simulations done in the fluctuations case. Compared to them, the trajectory shows noticeably straighter paths drawn by the robot while heading to a given antenna.

2.5.3 Lawnmower

As the lawnmower trajectory is predefined before the actual robot sweep of the field, the change in the sampling time is not relevant. Thus, the algorithm will run with the old setting of $T_s = 2$s and the time in seconds will be directly compared to the other methods run in the fluctuations case. We obtain 1200 steps that translate into $time = s_{total} \cdot T_s = 1200 \cdot 2 = 2400$s. This means 20% more that the 2002s of the *Gradient Ascent* and roughly 2.6 times the 910.98s of the TSP method.

Again, the inequality from above still holds true for stochastic transmission rates:

$$TSP < Gradient\ Ascent < Lawnmower$$

but this time the Gradient Ascent results are closer to the Lawnmower than to the TSP.

In terms of $avgMSE = 8.67$, that is about 2.94m average error from the real center position for each antenna. This is about the same as in the Gradient Ascent case, but at the cost of knowing the transmission rate model and its corresponding parameters.

2.6 Conclusion

Three types of algorithms have been developed and studied for the multiple antenna problem. Overall, in the deterministic case we obtained good results for the Gradient Ascent compared to its baselines (TSP, Lawnmower). In the fluctuations case, however, both the s_{total} and $avgMSE$ increased. This is partly due to the noise affecting the measurements when the rate function is stochastic. Note that results of the Gradient

Ascent are highly dependent on the inter-antenna distances, since if the next heading transmitter is far away from the current robot position its received signal will be drowned by noise, possibly breaking the algorithm.

Therefore, advanced path planning algorithms should be developed in the future to better deal with the noise present on the communication channel.

3 Path-Aware Global Optimization

3.1 Introduction and motivation

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. 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], [22], of the density of surface or underwater ocean litter, the largest-bandwidth location for radio transmission [5], the largest sand height on the seabed for dredging, maximal or minimal forest density in inaccessible areas [6], and so on.

Optimization techniques are, of course, closest to our method. However, as stated above, unlike in classical optimization here the path travelled by the robot matters, since it influences energy and time costs; and the function is unknown and must be learned from samples. As a first approximation, we use the length of the path 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, would suit the setup well after modifications to handle the unknown function by approximating derivatives from samples. We do evaluate such an alternative; of course, the fundamental limitation is that these methods can only find a local optimum.

Global optimization techniques [23] on the other hand, like branch-and-bound, see [24], Chapter IV of [23] or Bayesian optimization [25], [26] 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. Continuing along it would waste energy and time. Population-based techniques like genetic algorithms [27] or particle swarm optimization [28] would require large teams of robots, which are often unfeasible in practice.

To solve the Path-Aware Global Optimization problem, we extend an algorithm from the branch-and-bound class, deterministic optimistic optimization (DOO) [10], [12]. 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 of its soundness: 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 [10]. These are important features for path-aware optimization.

To make DOO path-aware, we formulate the decision of which direction to go at each time step as an optimal *control* problem. Due to overcommitment, we do not immedi-

ately 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 refines (lowers) the upper bound, weighted by the bound and function values to focus the refinement around the optima. Then, we run a dynamic programming algorithm [29] to solve the control problem of maximizing the cumulative weighted refinements along the trajectory. The path length to find a good upper bound, and hence a good estimate of the optimum, is implicitly reduced by this objective.

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 we study the impact of the tuning parameters of the algorithm, its robustness to errors in the Lipschitz constant, and compare it to baselines adapted from classical DOO and gradient ascent.

Related work can be found in other fields than optimization. For example, in robotics, mapping requires building a map of the environment, and leads to the famous SLAM problem [30] 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 [31]–[33]. Other variants, like coverage [34], aim to find a shortest path that examines the entire space using the robot sensors. Different from all these paradigms, the aim here is not to build or sense the entire function, but rather just to find the optimum as quickly as possible.

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.

3.2 Preliminaries

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:

$$f(x^*) - f(x) \leq l(x^*, x), \forall x \in X \tag{3.1}$$

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

$$|f(x) - f(y)| \leq l(x, y), \forall x, y \in X \tag{3.2}$$

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

$$l(x, y) = M||x - y||, \forall x, y \in X \tag{3.3}$$

where M represents the Lipschitz constant. The method can be extended to any metric l obeying the assumptions present in [12].

Here we will also use an alternative approach to the partition splitting in DOO: the construction of a so-called "saw-tooth" upper bound [12], defined as $B : X \to \mathbb{R}$ so that:

$$f(x) \leq B(x) = \min_{(x_s, f(x_s)) \in S} [f(x_s) + l(x, x_s)], \ \forall x \in X \tag{3.4}$$

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:

$$x_+ \in \text{argmax}_{x \in X} B(x). \tag{3.5}$$

Note that B is lowered (refined) with each sample gathered by the robot.

3.3 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:

$$x^* \in \text{argmax}_{x \in X} f(x) \tag{3.6}$$

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 $g : X \times U \to X$:

$$g(x_k, u_k) = x_k + u_k = x_{k+1} \tag{3.7}$$

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 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. Starting with an intuition, 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 (3.4). 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 3.1 for more intuition. Finally, 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:

$$\rho(x, u) = \frac{\hat{f}(x) + B(x)}{2} r(x, u) \tag{3.8}$$

where $\rho(x, u)$ is the reward function, B is the upper bound function defined in (3.4) 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 (3.4) and two slightly different sets of samples: B_1 with $S \cup \{(x, \hat{f}(x)\}$ and B_2 with $S \cup \{(x, \hat{f}(x), (x_+, \hat{f}(x_+)\}$, 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 $\hat{f}(x)$ in (3.8) 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, $\hat{f}(x)$ 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 $\hat{f}(x)$ 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 3.1 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 $\hat{f}(x_k)$, $\hat{f}(x_{k+1})$ (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.

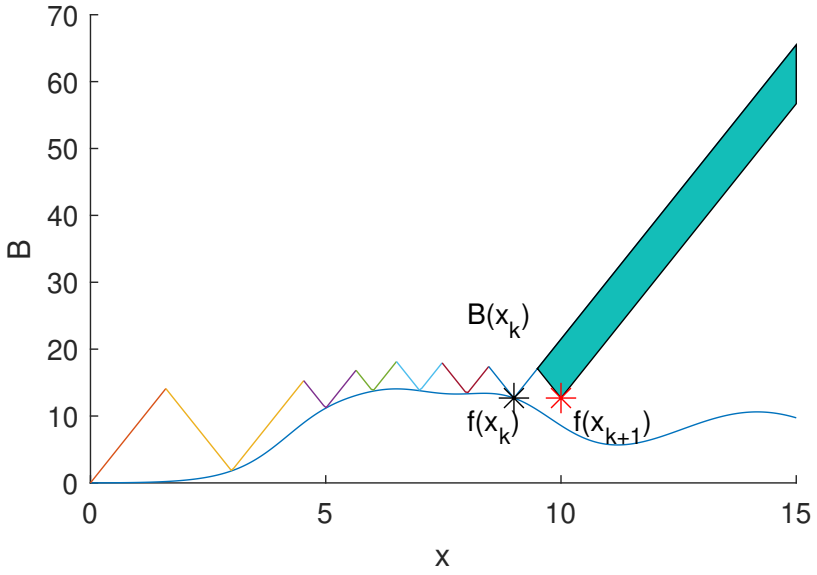


Figure 3.1. The sampled function is drawn with a blue line and the saw-tooth envelope represents its upper bound. The function sample $f(x_k)$ (black star) is used to approximate $\hat{f}(x_k)$, $\hat{f}(x_{k+1})$ (red star) and determine $B(x_k)$. The area predicted to be refined is colored in green.

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. We expect these 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 this value function in the infinite horizon setting:

$$V^h(x) = \sum_{k=0}^{\infty} \rho(x, u) \tag{3.9}$$

where $h : X \to U$ represents the control law and $u = h(x)$. This objective aims to explicitly maximize the upper bound based rewards, while the minimization of the travelled distance until the optimum occurs as a consequence. For this we are searching for an optimal policy, denoted by h^*, such that:

$$V^{h^*}(x) \geq V^h(x), \forall x, h. \tag{3.10}$$

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

$$Q^*(x,u) = \rho(x,u) + \max_{u'} Q^*(g(x,u),u') \tag{3.11}$$

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

3.4 Path-aware optimistic optimization

Algorithm 5 applies dynamic programming (DP) 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 (3.8).

At each step k, the robot takes a sample $f(x_k)$ and adds it to the sample set S. Then, m DP updates of the following form are run:

$$Q_+(x,u) = \rho(x,u) + \max_{u'} Q(g(x,u),u') \tag{3.12}$$

which turn the Bellman equation (3.11) into an iterative update. 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 DP updates (3.12) 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 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:

$$u_k = \arg\max_u Q(x_k,u), \tag{3.13}$$

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

Algorithm 5 describes the steps presented above.

Algorithm 5 Path-Aware Optimistic Optimization (OOPA).

Input: g, n_{grid} number of steps per X and B grid axes, discretized actions U, m number of DP sweeps, n trajectory steps, M Lipschitz constant

generate X and B grid, X_{grid}, using n_{grid}
initialize samples set $S \leftarrow \emptyset$
initialize $Q_0(x, u) \leftarrow 0 \ \forall x \in X, u \in U$
measure initial state x_0
for each step $k = 0, \dots, n-1$ **do**
    sample $f(x_k)$, add pair $(x_k, f(x_k))$ to S
    for each DP sweep $m = 0, \dots, m-1$ **do**
        for all states $x \in X_{grid}$, actions $u \in U$ **do**
            $x_+ \leftarrow g(x, u)$
            find $\hat{f}(x)$, $\hat{f}(x_+)$ and $B(x)$ using S and (3.4)
            $S_1 \leftarrow [S, (x, \hat{f}(x))]$, $S_2 \leftarrow [S_1, (x_+, \hat{f}(x_+))]$
            compute $B_1(x), \forall x \in X_{grid}$ using S_1 and (3.4)
            compute $B_2(x), \forall x \in X_{grid}$ using S_2 and (3.4)
            compute $r(x, u)$ using trapezoidal integration
                over $(B_1 - B_2)$ across X_{grid}
            compute $\rho(x, u)$ using (3.8)
            $Q_{m+1}(x, u) = \rho(x, u) + \max_{u'} Q_m(x_+, u')$
        end for
    end for
    $Q_0 = Q_m$
    $u_k = \text{argmax}_{u \in U} Q_m(x_k, u)$
    apply action u_k, measure next state x_{k+1}
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 DP updates, the Lipschitz constant M, 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.

3.5 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 sampling always the highest B-value point; and Gradient Ascent that creates an approximation plane using Local Linear Regression [35] on the neighboring samples and follows the gradient of this plane to quickly converge to a local maximum.

3.5.1 Influence of tuning parameters

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 3.7 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 $M = 364.54$.

The first parameter to be tuned is m, the number of DP 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 middle of the grid, $x_0 = [2; 2]$.

Figure 3.2 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 3.3 (left) shows the refined upper bound of the sampled function built using (3.4) and the $n = 125$ samples acquired. The robot trajectory in Figure 3.3 (right) shows more steps being spent around the highest RBF (centered in $[2.75; 3.5]$) and less around the ones with lower values (centered in $[0.75; 1.5], [3.25; 0.75]$). 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

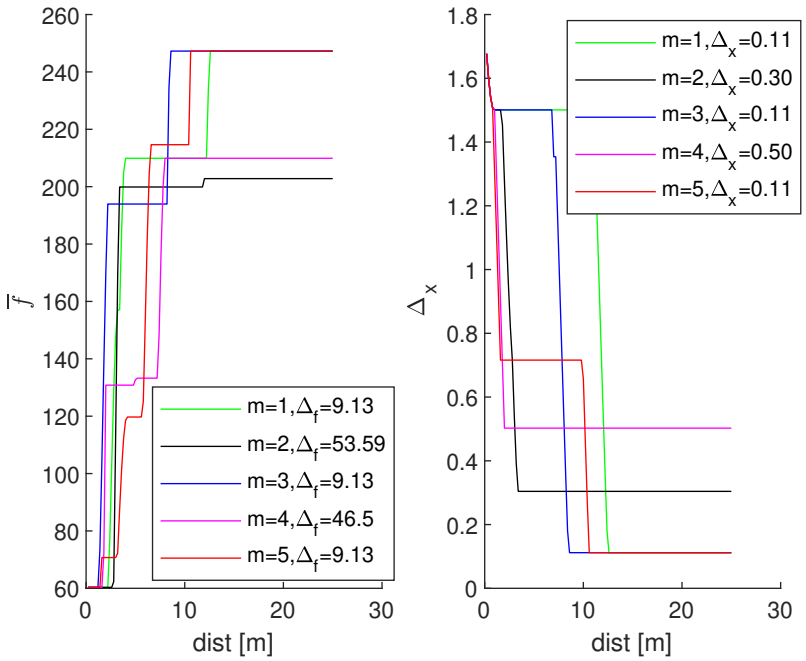


Figure 3.2. Illustration of the maximum value sampled so far (left) and the minimum distance to the maximum center x^* (right), denoted with Δ_x; for $m \in \{1, 2, 3, 4, 5\}$. The legend on the right shows the minimum distance between the samples of f and its maximum value f^*, denoted by Δ_f.

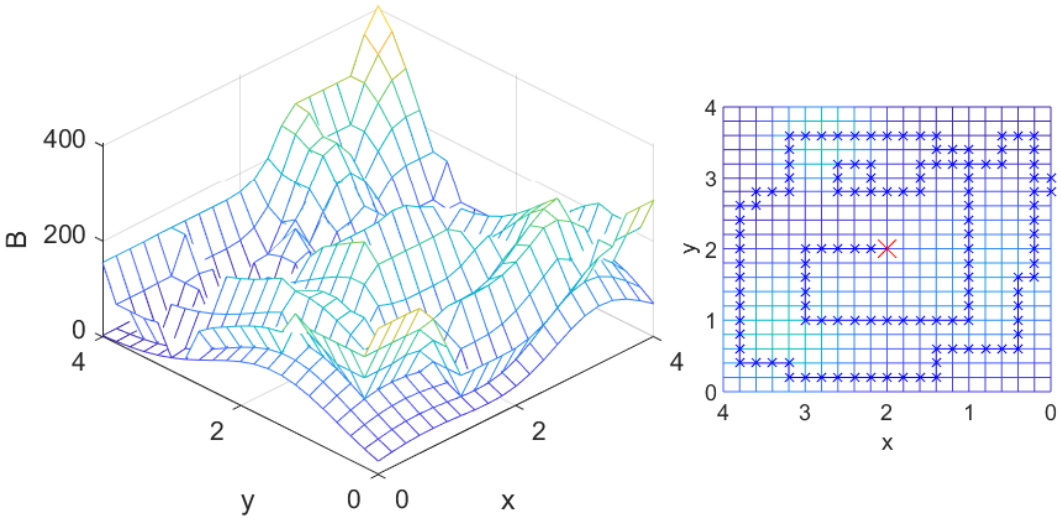


Figure 3.3. The sampled function is bounded from above by the refined B function (left), both evaluated across the same grid. The sampling trajectory of the robot, drawn with blue 'x' and starting from the red 'x', is displayed on the left. The refinements are geared towards higher function values, but not overly committed in those areas. This highlights a good exploration-exploitation tradeoff obtained for $m = 3$.

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.

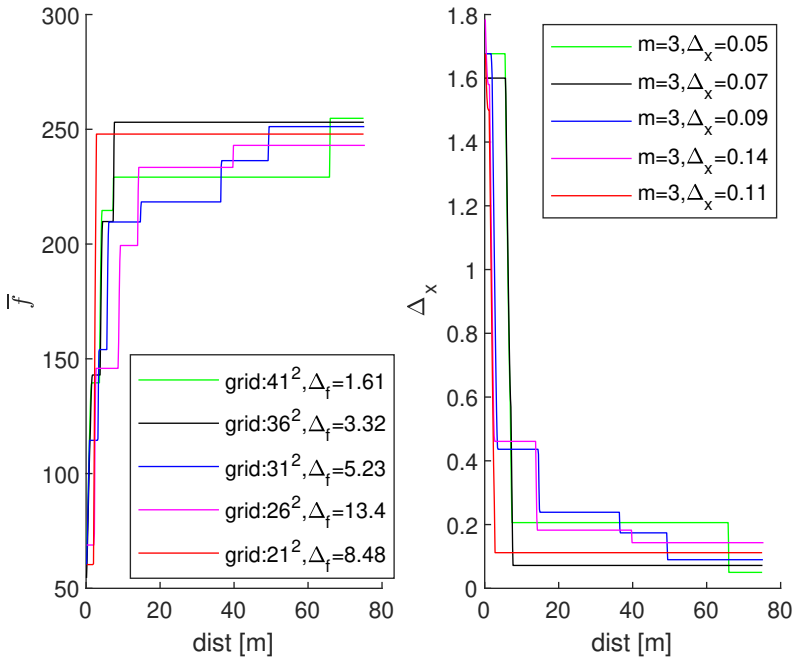


Figure 3.4. Smaller grid step sizes generally lead to better precision when searching for the optimum. Δ_f is displayed in the legend on the left, while the legend on the right displays Δ_x.

Except a "lucky" case (21^2), finer grids find the optimum with higher accuracy. Note that on all grids x^* is found with a precision of 1 grid step size, as $m = 3$ was kept constant during this experiment.

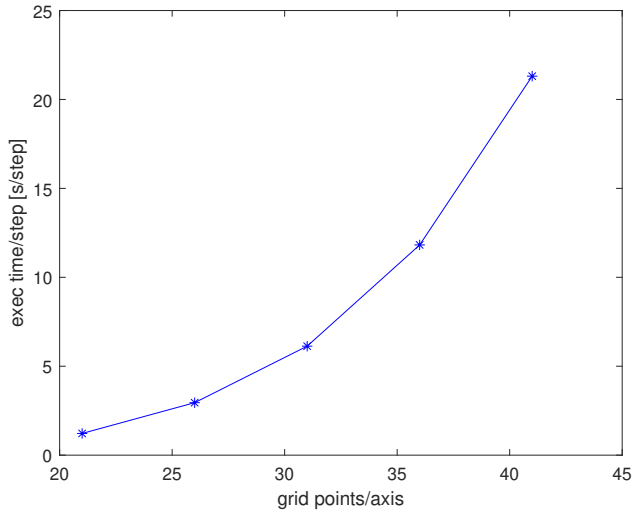


Figure 3.5. Time complexity has a quadratic growth with respect to the grid size.

Time complexity has a quadratic growth with respect to the grid of points. This can be observed in Figure 3.5 that shows the average execution time per step for each studied grid.

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 initial setup taking $M' = \lambda \cdot M$, with $\lambda \in \{0.2; 0.4; 0.6; 0.8; 1; 1.25; 1.5; 2; 2.5; 3\}$.

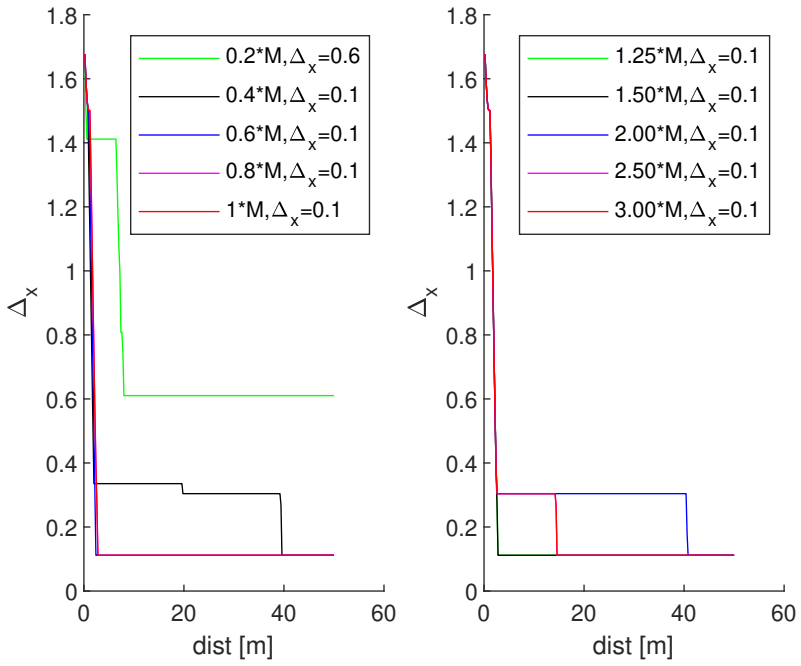


Figure 3.6. Robustness to underestimation (left) and to overestimation (right) of the Lipschitz constant. Taking M less than half of its initial approximation leads to finding x^* late or even breaks the algorithm. Taking M more than twice of this approximation finds at worst later x^*.

Figure 3.6 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 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.

3.5.2 Comparison to baselines

The learning-based algorithm is next compared to two different baselines using the same setup as above. The first baseline is represented by a classical DOO (CDOO) algorithm that, similarly to OOPA, applies the saw-tooth approach (3.4) to refine the function envelope with each sample taken. However, it fully commits to visit the maximum-B point (3.5) 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 the 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, a set of 15 equidistant starting positions placed along the lines drawn by the 3 RBF centers (recall the initial setup) is considered. To give enough time to the algorithms to find the maximum, we will set the number of steps per trajectory to $n = 250$ (50 m travel distance). In OOPA the robot will move with a fixed step of 0.2 m (equivalent to the step size of the 21^2 grid), while CDOO and Gradient Ascent can change their step in the interval $[0; 0.2]$ m. For the gradient-based method $N = 4$ (at least 3 is required to build up the approximation plane). We display in Figure 3.7 the result of the experiments.

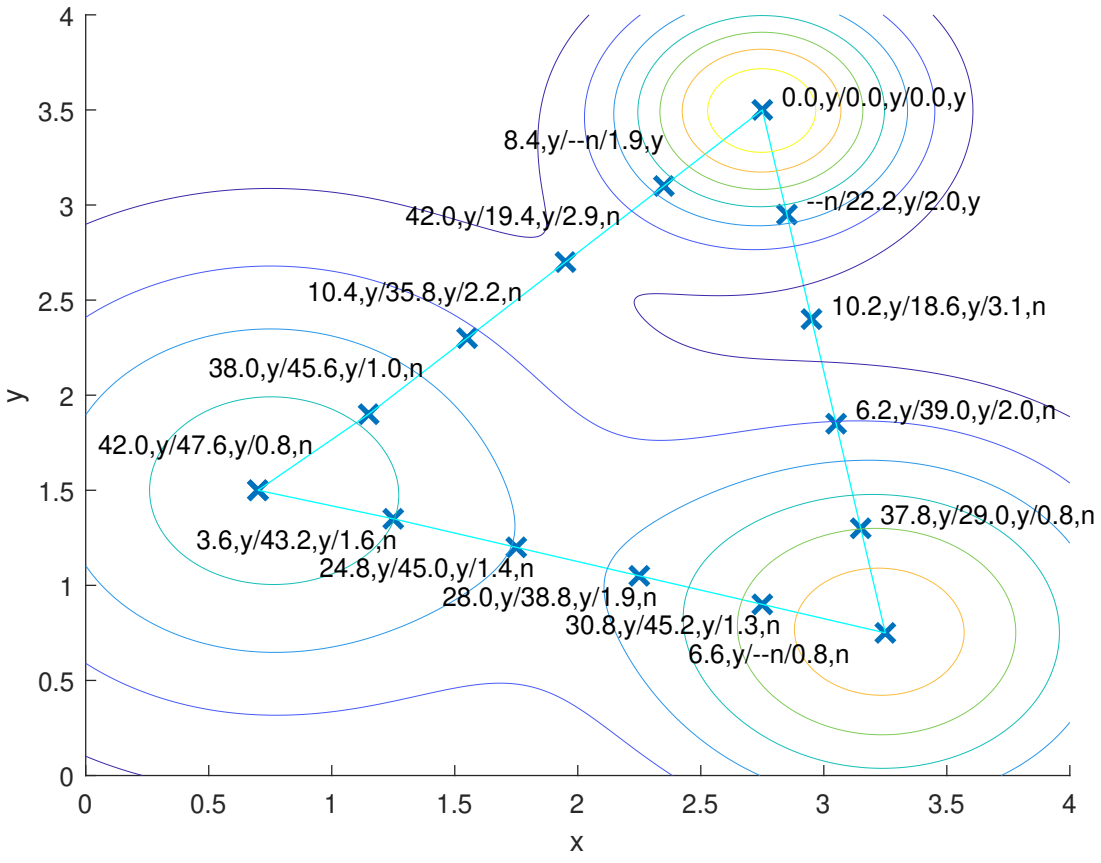


Figure 3.7. Results of the OOPA, CDOO and Gradient Ascent methods. The first entry of each text corresponds to OOPA, the second to CDOO and the third to Gradient Ascent. Characters 'y,n' show if the maximum was found with accuracy of 1 grid step size, $\delta = 0.2$ m. When this happens, the travelling distance to x^* is displayed alongside, '-' otherwise. The bold 'x' represent the starting positions.

Excepting a few outliers, the DOO-based methods find the maximum position at the distance of $\delta = 0.2$ m (1 grid step size accuracy). 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. These behaviors are expected due to the local nature of the gradient and its straightforward choice of the heading direction.

3.5.3 Behavior with a non-differentiable function

For the following experiments we keep the 21x21 interpolation grid across the 4x4 m search space from above. However, we replace the RBFs with rectangular pyramidal functions, with the goal of studying the algorithm behavior in the case of non-differentiable functions. The parameters of the pyramidal functions are: side lengths $b_i \in \{[2.6; 2.6], [1.2; 1.2], [2; 2]\}$, heights $h_i \in \{148.75, 255.0, 212.5\}$, rotation angles $a_i \in \{\pi/4, 2\pi/5, \pi/6\}$ and centers $c_i \in \{[0.75; 1.5], [2.75; 3.5], [3.25; 0.75]\}$ (see Figure 3.8 for a representation of a pyramidal function and Figure 3.10 for a contour plot). The global optimum remains $f^* = 255$, situated in $[2.75; 3.5]$. The corresponding Lipschitz constant is approximated experimentally to $M = 425$, a value that produces close-to-true upper bounds.

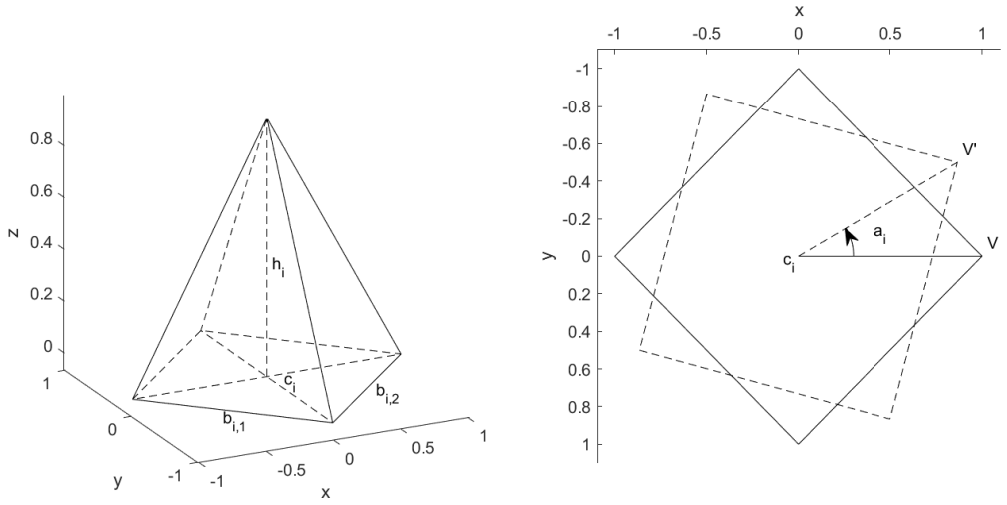


Figure 3.8. Representation of the parameters for a pyramidal function. Side lengths b_i, height h_i and center c_i are shown on the left plot. On the right figure, the angle a_i rotates clockwise the pyramid basis around its center starting from the parallel to the x axis passing through c_i.

We redo first the experiment of underestimation/overestimation of the Lipschitz constant using the pyramidal functions setting defined above. For this we set $M' = \lambda \cdot M$, with $\lambda \in \{0.2; 0.4; 0.6; 0.8;\ 1; 1.25; 1.5; 2; 2.5; 3\}$. Figure 3.9 shows that higher values of M typically find x^* with better accuracy compared to low values of the same constant. Again, taking a rather high M and decreasing it empirically represents a safer choice, as it gives better overall results without breaking the upper-bound properties used in the algorithm.

The comparison of OOPA to CDOO and Gradient Ascent baselines is performed next, keeping the same settings as in the RBFs case (excepting the sampled function). Note that the Gradient Ascent might fail due to the non-differentiable (pyramidal-based) function that is sampled, thus applying it is mostly an empirical attempt.

Figure 3.10 shows that the DOO-based algorithms nearly always find x^* with accuracy of $\delta = 0.2\,\text{m}$ (excepting a few outliers). When x^* is found, OOPA scores roughly 39.3% less travelling distance on average compared to CDOO. The Gradient Ascent converges to the

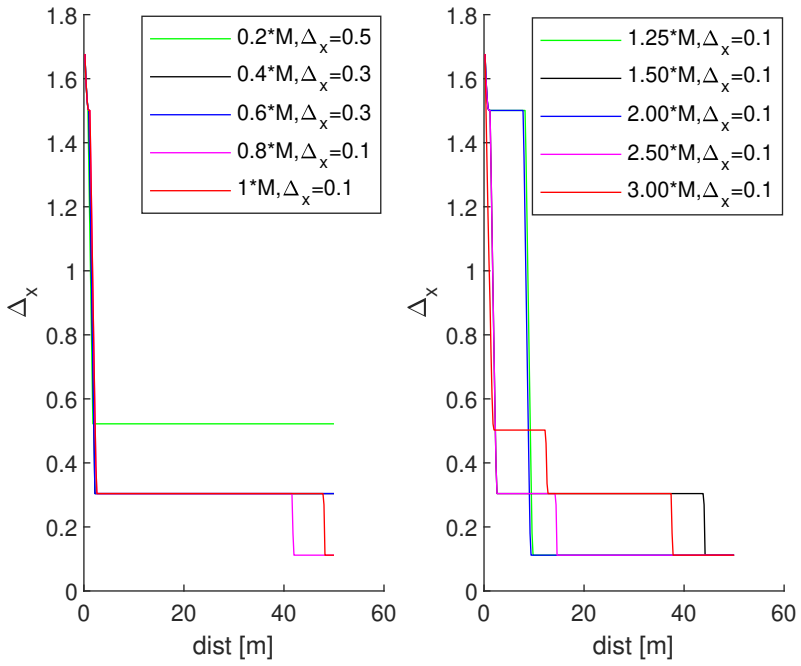


Figure 3.9. Robustness to underestimation (left) and to overestimation (right) of the Lipschitz constant when sampling pyramidal based functions. Low values of M perform poorer while searching for x^* (sometimes even breaking the algorithm) compered to high values of the same constant.

global maxima in only 3 out of the 15 runs, however, it does so at faster rates. Again, results are similar to the case of the RBF-based function.

3.6 Conclusion

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 dynamic programming and optimistic optimization was defined to quickly reduce the upper bounds around optima and implicitly find an approximate location of it.

In the future, the method will be extended to find multiple optima (e.g. local maxima that become points of interest over a preset threshold) and add a quantity collection objective, in which the robot tries to transfer as much data from the transmitters, map as much litter or forest density, etc. We will also try to provide guarantees on convergence to the optima and give an estimation of the number of steps until these optima are found. Some possible extensions to the current method are improving the algorithm so that it works for stochastic objective functions or when no Lipschitz assumptions are made.

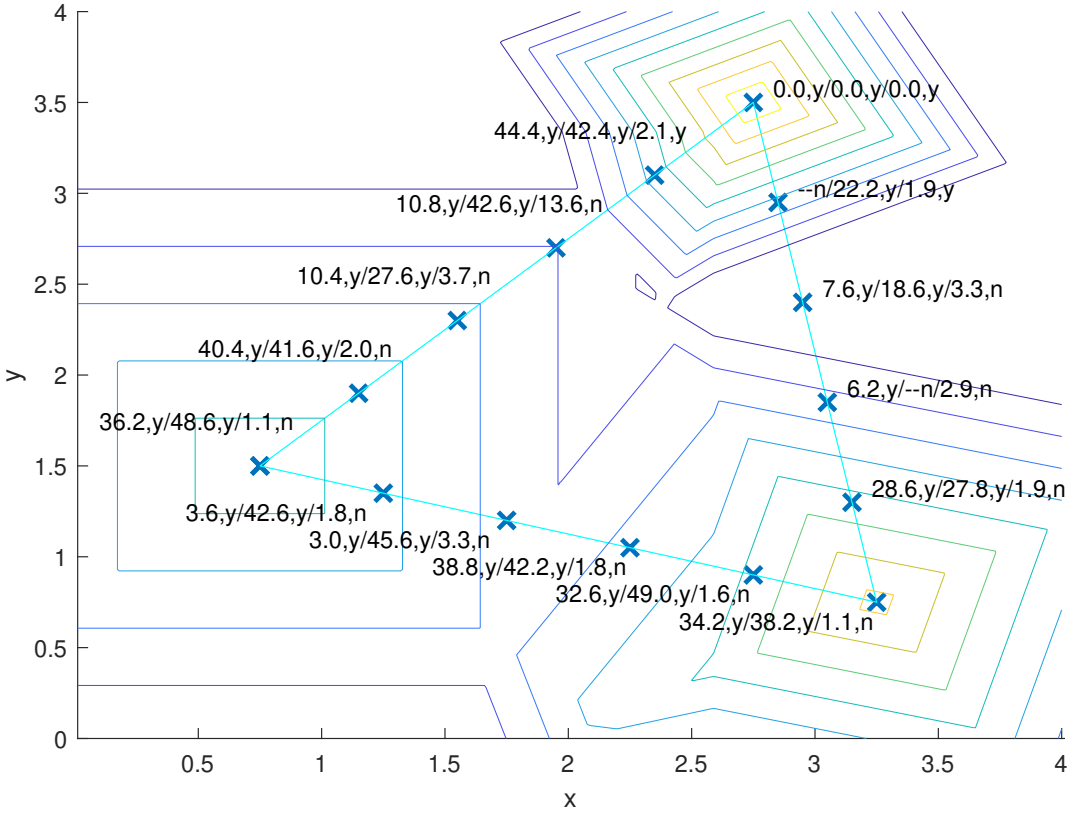


Figure 3.10. Results of the OOPA, CDOO and Gradient Ascent methods using the pyramidal functions setting. When x^* is found, OOPA scores roughly 39% less travelling distance on average compared to CDOO. The Gradient Ascent converges to the global maxima in only 3 out of the 15 runs, however at faster rates. Again, results are similar to the case of the RBF-based function.

4 Conclusions and future work

We developed algorithms for mobile robots to find the optima of a function defined over an operating area, while possibly transferring data over a wireless communication protocol if the optima correspond to antenna transmitters (as in Chapter 2). Optima are learned online from samples during a single trajectory run, and the trajectory length is especially important due to energy and time constraints.

The first task, Multiple Antenna Search, aims to find the antennas position and gather wirelessly the data buffers of the antennas in the least amount of time. We solved this problem using well-known algorithms adapted to our problem: Gradient Ascent, the Travelling Salesman Problem and the Lawnmower method. All these algorithms were extensively tested on deterministic and stochastic transmission rates, all antenna positions being generated uniformly randomly across the searching space. Tests showed that the Gradient Ascent works well when there is no noise on the communication channel, its performance being closer to the TSP's compared to the one of the Lawnmower trajectory. However, the Gradient Ascent performs poorly or even breaks when the transmission signal is drowned by noise. This is expected as the direction of the gradient followed by the robot is highly influenced by the noise. Therefore, advanced path planning algorithms that are more resilient to fluctuations on the communication channel need to be further developed.

To solve the second problem, Path-Aware Global Optimization, whose objective is to find as quickly as possible a global maximum of the function, we developed the Path-Aware Optimistic Optimization (OOPA). This novel optimal control algorithm combines dynamic programming to find the decision of direction taking of the agent at each step (defined as an optimal control problem) and optimistic optimization to build and refine the function's upper bound with each new sample, to quickly focus the search around the optima. Note that unlike the Multiple Antenna Search, this problem does not include any transmission objectives. OOPA was tested against the classical DOO (CDOO) and Gradient Ascent methods in a deterministic setting using different types of functions (radial basis and pyramidal functions) starting from different initial positions. As results show, OOPA proved to outperform CDOO by scoring less distance on average until reaching the optima. Gradient Ascent, however, rarely converges to the function (global) optima due to its local optimization properties, but the convergence happens at faster rates compared to the DOO-based methods.

In future work we will consider more general robot dynamics than simple integrators, e.g. by adding motion related states, additional velocities etc., and then demonstrate the methods on real robots such as aerial drones or ground robots. For this, the stochastic case needs to be studied in depth to find a way to mitigate the noise present e.g. on the communication channel. Other objectives are to provide near-optimality guarantees, bring algorithmic improvements to handle continuous states and actions, and eliminate the need to know the Lipschitz constant. We will also add a collection objective to maximize the integral of the sampled function over the robot trajectory, e.g. map as much forest density or air pollutants as possible, maximize the data transfers between the robot

and the wireless antennas or the collection of ocean litter, etc. Finally, the methods will be extended to the multiagent framework, making them more applicable in real-life multi-robot scenarios.

All in all, the present work represents a strong foundation for future research topics that will be focusing not only on algorithms development, but also on their practical implementation on mobile robots.

5 Bibliography

[1] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäeffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The kuka-dlr lightweight robot arm - a new reference platform for robotics research and manufacturing," vol. 2, Jan. 2010, pp. 1–8.

[2] F. Auat Cheein and R. Carelli, "Agricultural robotics: Unmanned robotic service units in agricultural tasks," *Industrial Electronics Magazine, IEEE*, vol. 7, pp. 48–58, Sep. 2013. DOI: 10.1109/MIE.2013.2252957.

[3] M. Ben-Ari and F. Mondada, "Robots and their applications," in. Jan. 2018, pp. 1–20, ISBN: 978-3-319-62532-4. DOI: 10.1007/978-3-319-62533-1_1.

[4] C. Yuan, Z. Liu, and Y. Zhang, "Fire detection using infrared images for uav-based forest fire surveillance," Jun. 2017, pp. 567–572. DOI: 10.1109/ICUAS.2017.7991306.

[5] J. Fink and V. Kumar, "Online methods for radio signal mapping with mobile robots," May 2010, pp. 1940–1945. DOI: 10.1109/ROBOT.2010.5509574.

[6] 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, Jun. 2017. DOI: 10.1016/j.rse.2017.04.007.

[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, Jan. 2020. DOI: 10.4236/ojap.2020.92003.

[8] J. Nocedal and S. Wright, *Numerical Optimization*. Jan. 2006, ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[9] P. Raica, *Numerical optimization - lecture notes*, Chapter II - Numerical Optimization.

[10] R. Munos, "Optimistic optimization of a deterministic function without the knowledge of its smoothness," vol. 24, Jan. 2011.

[11] T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Jul. 2020, ISBN: 9781108486828. DOI: 10.1017/9781108571401.

[12] R. Munos, *From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning*. Jan. 2014, ISBN: 9781601987679. DOI: 10.1561/9781601987679.

[13] L. Busoniu, *Learning control - lecture notes*.

[14] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 1563–1600, Mar. 2010.

[15] H. Bourel, O. Maillard, and M. S. Talebi, "Tightening exploration in upper confidence reinforcement learning," *CoRR*, vol. abs/2004.09656, 2020. arXiv: 2004.09656. [Online]. Available: https://arxiv.org/abs/2004.09656.

[16] J. Lohéac, V. S. Varma, and I.-C. Morarescu, "Optimal control for a mobile robot with a communication objective," working paper or preprint, Nov. 2019, [Online]. Available: https://hal.archives-ouvertes.fr/hal-02353582.

[17] L. Busoniu, V. Varma, J. Lohéac, A. Codrean, O. Ştefan, I.-C. Morarescu, and S. Lasaulce, "Learning control for transmission and navigation with a mobile robot under unknown communication rates," *Control Engineering Practice*, vol. 100, p. 104 460, Jul. 2020. DOI: 10.1016/j.conengprac.2020.104460.

[18] L. Garcia, L. Parra, J. Jimenez, J. Lloret, P. Mauri, and P. Lorenz, "Dronaway: A proposal on the use of remote sensing drones as mobile gateway for wsn in precision agriculture," *Applied Sciences*, vol. 10, p. 6668, Sep. 2020. DOI: 10.3390/app10196668.

[19] IDESIO, *Mole: An underground soil moisture sensor linked to the cloud*, 2018. [Online]. Available: https://climateinnovationwindow.eu/innovations/mole.

[20] A. Sathyan, N. Boone, and K. Cohen, "Comparison of approximate approaches to solving the travelling salesman problem and its application to uav swarming," *International Journal of Unmanned Systems Engineering (IJUSEng)*, vol. 3, pp. 1–16, Jan. 2015. DOI: 10.14323/ijuseng.2015.1.

[21] M. Venhuis, *The basics of search algorithms explained with intuitive visualizations*, 2019. [Online]. Available: https://towardsdatascience.com/around-the-world-in-90-414-kilometers-ce84c03b8552.

[22] A. Lilienthal and T. Duckett, "Building gas concentration gridmaps with a mobile robot," *Robotics and Autonomous Systems*, vol. 48, pp. 3–16, Aug. 2004. DOI: 10.1016/j.robot.2004.05.002.

[23] R. Horst and H. Tuy, *Global Optimization—Deterministic Approach*. Jan. 1996, ISBN: 978-3-642-08247-4. DOI: 10.1007/978-3-662-03199-5.

[24] E. Lawler and D. Woods, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, Aug. 1966. DOI: 10.1287/opre.14.4.699.

[25] P. Frazier, "A tutorial on bayesian optimization," Jul. 2018.

[26] 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. DOI: 10.1109/JPROC.2015.2494218.

[27] M. Mitchell, *An Introduction to Genetic Algorithms. MIT Press, Cambridge*. Jan. 1998, ISBN: 978-0-262-63185-3. DOI: 10.7551/mitpress/3927.001.0001.

[28] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intelligence*, vol. 1, Oct. 2007. DOI: 10.1007/s11721-007-0002-0.

[29] D. Bertsekas, *Reinforcement Learning and Optimal Control*. Jul. 2019, p. 388, ISBN: 9781886529397.

[30] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[31] 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. DOI: 10.1109/ROBOT.2010.5509714.

[32] 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. DOI: 10.1109/ICRA.2012.6224902.

[33] Z. Lim, D. Hsu, and W. Lee, "Adaptive informative path planning in metric spaces," *The International Journal of Robotics Research*, vol. 35, Sep. 2015. DOI: 10.1177/0278364915596378.

[34] H. Choset, "Coverage for robotics - a survey of recent results," *Ann. Math. Artif. Intell.*, vol. 31, pp. 113–126, Oct. 2001. DOI: 10.1023/A:1016639210559.

[35] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1–5, pp. 11–73, 1997.